Visualisation of Overlapping Sets and Clusters with Euler Diagrams

Paolo Simonetto

Université Bordeaux 1

École doctorale de mathématiques et informatique

Doctorat en informatique

Visualisation of Overlapping Sets and Clusters with Euler Diagrams

Candidate:
Paolo Simonetto

Thesis Directors:
David Auber
Guy Melançon

December 2011

In the loving memory of John,
Giovanna and Sante.

Abstract

In this thesis, we propose a method for the visualisation of overlapping sets and of fuzzy graph clusterings based on Euler diagrams.

Euler diagrams are probably the most intuitive and most used method to depict sets in which elements can be shared. Such a powerful visualisation metaphor could be an invaluable visualisation tool, but the automatic generation of Euler diagrams still presents many challenging problems. First, not all instances can be drawn using standard Euler diagrams. Second, most existing algorithms focus on diagrams of modest dimensions while real-world applications typically features much larger data. Third, the generation process must be reliable and reasonably fast.

In this thesis, we describe an extended version of Euler diagrams that can be produced for every input instance. We then propose an automatic procedure for the generation of such diagrams that specifically target large input instances. Finally, we present a software implementation of this method and we describe some output examples generated on real-world data.

Résumé

Dans cette thèse, nous proposons une méthode pour la visualisation d'ensembles chevauchant et de basé sur les diagrammes d'Euler.

Les diagrammes d'Euler sont probablement les plus intuitifs pour représenter de manière schématique les ensembles qui partagent des éléments. Cette métaphore visuelle est ainsi un outil puissant en termes de visualisation d'information. Cependant, la génération automatique de ces diagrammes présente encore de nombreux problèmes difficiles. Premièrement, tous les clustering chevauchants ne peuvent pas être dessinées avec les diagrammes d'Euler classiques. Deuxièmement, la plupart des algorithmes existants permettent uniquement de représenter les diagrammes de dimensions modestes. Troisièmement, les besoins des applications réelles requièrent un processus plus fiable et plus rapide.

Dans cette thèse, nous décrivons une version étendue des diagrammes d'Euler. Cette extension permet de modéliser l'ensemble des instances de la classe des clustering chevauchants. Nous proposons ensuite un algorithme automatique de génération de cette extension des diagrammes d'Euler. Enfin, nous présentons une implémentation logicielle et des expérimentations de ce nouvel algorithme.

Acknowledgements

I am very grateful to the members of the examination panel of my thesis for their comments and questions. In particular, I would like to thank Stephen Kobourov and Jarke van Wijk, the reviewers, for their invaluable corrections and suggestions, and David Auber and Guy Melançon, my supervisors, for supporting and helping me in many ways, even outside the research environment.

I would like to express my gratitude to two good friends and major co-authors of my articles: Romain Bourqui and Daniel Archambault. I thank Daniel particularly for the invaluable tutoring he provided me on scientific writing and on the English language, that deeply influenced the clarity of my publications.

I would also like to thank the friends who reviewed part of my thesis. Among these, a special thanks to Andrew Collins and Frank Hopfgartner.

Finally, the greatest thanks goes to my girlfriend, Karen Jespersen, for the dedication she put in reviewing and correcting my writing. For this reason, she would really have deserved to be mentioned as an author in every single publication, including this thesis.

Thanks a lot!

Contents

List of Figures

List of Tables

List of Algorithms

Mathematical Notation

G	A generic graph. 30
V	The set of nodes of a graph. 30
E	The set of edges of a graph. 30
u, v, u_i	Generic nodes of a graph. 30
e, e_i	Generic edges of a graph. 30
(u, v)	A directed edge between the nodes u and v. 30
$[u, v]$	An undirected edge between the nodes u and v. 30
$s(e)$	The source node of the edge e. 32
$t(e)$	The target node of the edge e. 32
$\deg_{\text{out}}(u)$	The out degree of the node u. 32
$\deg_{\text{in}}(u)$	The in degree of the node u. 32
$\deg(u)$	The degree of the node u. 32
$\text{dist}(u, v)$	The graph distance between the nodes u and v. 32
C	The set of the connected components of a graph. 34
c, c_i	A generic connected component of a graph. 34
$\text{lev}(u)$	The level of node u in a forest. 34
V^i	The set of nodes at level i in a forest. 34
$\kappa(G)$	The connectivity of the graph G. 34
K_i	The complete graph with i nodes. 35
$K_{i,j}$	The complete bipartite graph with partitions of i and j nodes. 35
F	The set of the faces of a plane graph. 38
f, f_i	A generic face of a plane graph. 38
f_0	The external (unbounded) face of a component. 121
$f_+, f_1, f_2 \ldots$	An internal (bounded) face of a component. 121
$\mathcal{B}(f)$	The nodes and edges constituting the boundary of face f. 38
$\mathcal{B}^n(f)$	The set of nodes in the boundary of face f. 38
$\mathcal{B}^e(f)$	The set of edges in the boundary of face f. 38
$O(\cdot)$	The asymptotic notation. 42
S	The clustering of a clustered graph or Euler diagram. 40
s, s_i	Generic clusters of a clustered graph or Euler diagram. 40, 50
$Q = \mathcal{P}(S)$	The power set of a clustering S. 50
q	A generic set of clusters, that is a generic element of Q. 50
Z^*	The set of all zones of an Euler diagram. 50
Z	The set of the non-empty zones of an Euler diagram. 50
z, z_i	Generic zones of an Euler diagram. 50
$\mathsf{A}, \mathsf{B} \ldots$	Clusters of an Euler diagram. 50
$\mathsf{abd}, \mathsf{de} \ldots$	Zones of an Euler diagram. 50
$\mathcal{Z}(q)$	The zone identified by the set of clusters q. 50

$\mathcal{A}(s), \mathcal{A}(z)$	The associated zones of s or the associated clusters of z. 50
D	A generic Euler diagram. 51
T	The set of curves of an Euler diagram. 51
T_s	The cluster curves of cluster s. 51
t, t_i	Generic curves of an Euler diagram. 51
$\mathrm{int}(t)$	The internal region of t. 51
$\mathrm{ext}(t)$	The external region of t. 51
$s^{\mathcal{R}}, z^{\mathcal{R}}$	The region associated to a cluster s or a zone z. 51
$G^o = (V^o, E^o, S)$	The original graph in Euler representations. 77
$G^z = (V^z, E^z)$	The zone graph in Euler representations. 77
$G^o = (V^g, E^g)$	The grid graph in Euler representations. 77
G^z_s	The subgraph of G^z induced by the zones of s. 82
$m_c(e)$	Connection metric for the zone graph edge e. 83
$m_b(e)$	Brushing metric for the zone graph edge e. 84
$m_i(e)$	Coincidence metric for the zone graph edge e. 84
$m_g(e)$	Global metric for the zone graph edge e. 84
r	Radius of the circles in the grid graph generation. 89
p	Projection of a node into the line of an edge. 89, 106
$\boldsymbol{p}_x$	Position on the plane of a point, node or edge x. 89, 105
$\mathcal{C}$	A Bézier curve. 95
P_0, P_1, P_2, P_3	Control points of a cubic Bézier curve. 95
δ	Optimal edge length of `PrEd` and `ImPrEd`. 102, 111
γ	Optimal node-edge distance of `PrEd` and `ImPrEd`. 102, 111
Cr	Set of crossable edges in `ImPrEd`. 111
Fl	Set of flexible edges in `ImPrEd`. 111
$\mathcal{W}$	The weight of nodes and edges in `ImPrEd`. 111
$\mathcal{F}^r(u, v)$	Repulsive force between node u and node v. 105
$\mathcal{F}^a(e)$	Attractive force exerted by edge e. 105
$\mathcal{F}^e(u, e)$	Repulsive force between node u and edge e. 105
$\mathcal{F}^g(u)$	Gravity attraction force acting on node u. 116
$\mathcal{F}(u)$	Resulting force acting on node u. 105
$\mathcal{A}$	A partitioning of an angle into eight intervals $\mathcal{A}^i$. 106
$\mathcal{M}_u$	Maximal movement for the node u according to $\mathcal{A}$. 106
$\mathcal{S}^e(u)$	The surrounding edges of a node u. 117
$\mathcal{N}^e(u)$	The nearby edges of a node u. 125
$\mathcal{N}^n(u)$	The nearby nodes of a node u. 125

Chapter 1

Introduction

During my master thesis, we worked on a method in which overlapping communities were detected in biological networks. Once the communities were detected, our software would assign a different colour to each of them, and fill the network elements with the colour of the community they belonged to. This visualisation technique clearly presented several critical problems: not only did the user have to identify the communities by distinguishing dozens of colours, but it was also impossible to detect the elements belonging to multiple communities, since these would only have the colour of the larger community.

It is not a surprise if the question "Have you thought of other visualisation methods?" arrived even before the end of my first presentation at the LaBRI. This question was the trigger to the work in this thesis: to provide a visual method for analysing overlapping sets of elements. With such a task, it was inevitable to consider Euler diagrams.

Euler Diagrams. Euler diagrams are probably the most used and intuitive representations for depicting sets and their elements. In Euler diagrams, sets are depicted as regions of the plane. The element inclusion is then encoded by placing the set elements inside the corresponding set regions, and intersection and disjunction between sets are depicted by the presence or absence of overlaps between the corresponding set regions.

Such a common and comprehensible visual metaphor is extremely valuable in information visualisation. An increasing number of data involves overlapping sets. Biological networks, social networks and many others naturally contain classes where elements are shared by two or more of them. Recent clustering algorithms are designed to detect this overlapping structure, rather than providing a classical partitioning where every element belongs to only one class [86].

Euler Diagrams in Visual Data Analysis. The application of Euler diagrams to the visual analysis of data opens a number of non-trivial problems. First, the automatic generation of Euler diagrams is a relatively new field, and the methods proposed so far still present large margins of improvement. Second, Euler diagrams are generally used to depict a small number of sets with few overlaps between them, while real application requirements are typically different. Third, the visual analysis of the data requires the visualisation to be interactively adapted to the needs of the user.

We tried to solve these issues by developing a visual encoding, the Euler Representation, and a software that produces it: EulerView. Euler representations are diagrams similar to the classical Euler ones, but characterised by a less restrictive set of rules, and by the addition of graphical techniques such as textures and colour transparency. EulerView is a plug-in for the graph visualisation framework Tulip that allows the interactive visualisation and exploration of overlapping sets.

The Thesis

This document presents the work related to the main topic of my doctoral studies. The topics discussed here are organised as follows:

Chapter two introduces the visualisation science and the first notions of Euler diagrams. It also discusses how these fields are related to each other, and in particular how and why Euler diagrams can be an extremely powerful visualisation instrument.

Chapter three introduces the graph drawing and Euler diagram theory that is essential to discuss the automatic generation of such diagrams. Graphs and graph drawing are in fact crucial components of all Euler diagram generation algorithms.

Chapter four presents the most important algorithms in the literature for the generation of Euler diagrams. In order to compare our methods with existing ones, we will also outline our generation procedure and discuss the major differences and innovations.

Chapter five details the procedure we developed to generate Euler representations. The procedure is divided in five stages, and for each of them, we describe the results we aim to obtain and we provide the technical details necessary to reimplement the method.

Chapter six describes a graph drawing algorithm that we designed to improve the quality and the computation time of the Euler representation generation algorithm. Despite the reasons for its development, the application of this algorithm is not limited to Euler diagrams.

Chapter seven presents the implementation of the Euler representation generation method we produced, EulerView. The software also features some basic interaction techniques, that can further improve the usefulness of Euler representations as an analysis tool.

Chapter eight contains conclusions and discussion of future work.

Appendix A includes the biographies of the two mathematicians that developed the kind of diagrams used in this thesis: Leonhard Euler and John Venn.

Chapter 2

Visualisation and Euler Diagrams

This chapter presents some background information on the two main topics of this thesis: the field of visualisation and Euler diagrams. By not going too deep into technical details, we hope to both provide a smoother introduction to the readers who are not familiar with these topics, and to allow expert readers to skip this chapter without much harm.

First, we will introduce the science of visualisation. We will start by providing some rudiments of perception and of earlier stages of the visualisation science. Then, we will discuss the present visualisation, its goals and its disciplines.

Second, we will present the graphical representations called Euler diagrams. Again, we will discuss their original form and aim, the evolution into their current state, and the properties and techniques applied to the modern diagrams. Here, we will also present the closely related Venn diagrams.

Third, we will show how the previous topics are related to each other. In particular, we will discuss how and why Euler diagrams can be important tools in visualisation. We will approach this topic both from a technical point of view by discussing their theoretical strengths, and from a practical point of view, by providing a case study example.

Finally, we will state the theoretical limits that hinder the applicability of Euler diagrams in visualisation.

2.1 Visualisation

The use of tools has been considered one of the biggest achievements of mankind and the reason for further development of the human mind. We develop tools to perform tasks that would be difficult or impossible to achieve with our natural abilities: bows allowed us to kill from a distance, musical instruments to produce sounds outside our vocal range, and cars to move faster and farther than we can do by running. In other words, we develop tools to overcome the limitations of the human body.

However, our constraints are not only of a physical nature. Our ability to analyse and understand complex problems with our bare minds is also limited, and can be greatly improved by the use of appropriate tools:

> The power of the unaided mind is highly overrated. Without external aids, memory, thought, and reasoning are all constrained. But human intelligence is highly flexible and adaptive, superb at inventing procedures and objects that overcome its own limits. The real powers come from devising external aids that enhance cognitive abilities. How have we increased memory, thought, and reasoning? By the invention of external aids: It is things that make us smart.
>
> Norman [76, p. 43]

Tasks like studying, decision making or calculation are in fact much more difficult if one cannot scratch or report partial results on paper [10], and can be close to impossible if the problem is not trivial. In these cases, simple tools like pencil and paper allow us to greatly extend our mental abilities.

Not surprisingly, most of the tools that help reasoning come in graphical form, such as drawings, diagrams, pictures, maps and writing. Sight is a very developed sense in humans and is the one most used to interact with the surrounding world:

> All men by nature desire to know. An indication of this is the delight we take in our senses; for even apart from their usefulness they are loved for themselves; and above all others the sense of sight. For not only with a view to action, but even when we are not going to do anything, we prefer sight to almost everything else. The reason is that this, most of all the senses, makes us know and brings to light many differences between things.
>
> Aristotle, *Metaphysics*

The same tools that help us with reasoning provide even greater help when communicating. In fact, communication does not only require both parties to reason on the actual topic, but also to channel the information in the form more convenient to express that idea.

Visualisation might be seen as the study of graphical tools that help reasoning and communication. Visualisation aims to develop efficient methods to imprint and transfer information via graphical means, so that we can benefit from the enormous potentials offered by the human eye (see figure 2.1). It is the purpose of visualisation to design or improve charts and drawings, to investigate their usefulness and to develop systems that help a user to visually investigate and extract information from them. More formal definitions describe the subject as

> *Visualisation*: The use of computer-supported, interactive, visual representations of data to amplify cognition.
>
> Card, Mackinlay and Shneiderman [10, p. 6]

In the next sections we will present the field of visualisation in greater detail. First, we will introduce some concepts of perception, to understand how the human sight works and how to get the most out of it. Second, we will describe how visualisation evolved from visual communication into its current state. Third, we will explain the strong link between visualisation and computer science, mentioned already in the previous definition. Finally, we will discuss the aims and branches of the current discipline.

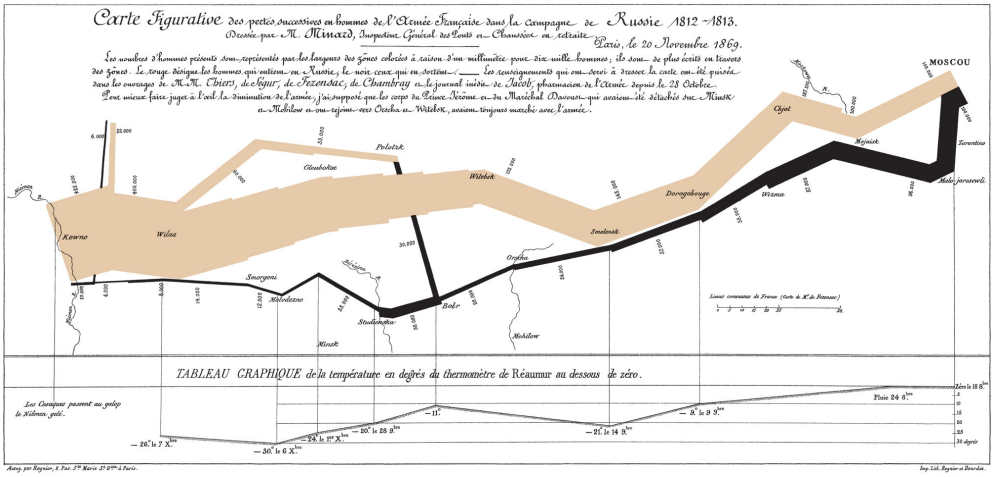


Figure 2.1: One of the most classic examples of a well realised diagram. The drawing, realised by Charles Joseph Minard in 1869, shows the loss of men in the French army in the Russian campaign. The diagram incorporates, in a very intuitive way, information as the army size, the path taken, the retreats and the average temperatures.

2.1.1 The Human Eye and Perception

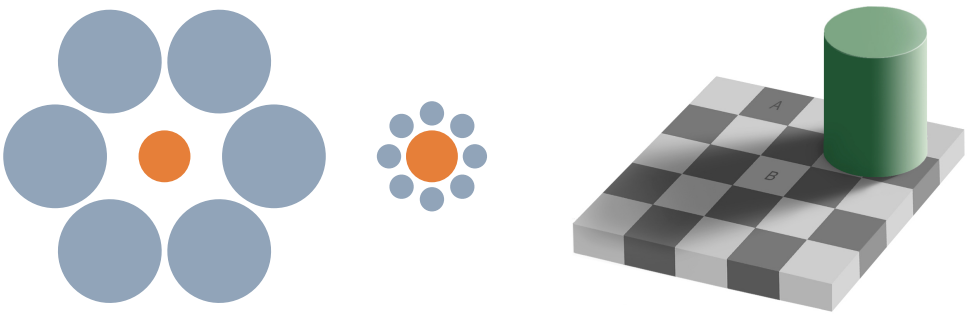
Sight plays a crucial role in the way humans acquire information. The centrality of sight among other senses is due to the enormous amount of data that our nervous system can process and to the efficiency with which we can extract information:

> We acquire more information through vision than through all of the other senses combined. The 20 billion or so neurons of the brain devoted to analysing visual information provide a pattern-finding mechanism that is a fundamental component in much of our cognitive activity.
>
> Ware [111, p. 2]

A Perception Model. We can obtain more insights on weaknesses and strengths of the sight by further investigating the mechanics of vision. Perception is the process of interpreting the stimulus coming from our visual system, that is how our brain understands an observed scene. According to the simplified model of human visual perception described by Ware [111], it is possible to identify three stages in the perception process.

In the first stage, the observed scene is processed by large arrays of neurons in the primary visual cortex in the back of the brain. Here features of objects in the visual field such as orientation, colour, texture and movement patterns are recognised and transferred to the next stage. The processing of this information is very rapid as it is accomplished by billions of neurons working in parallel.

In the second stage, the visual field is divided into regions sharing the same features identified in the previous step, such as continuous regions or regions of the same colour. In other words, we identify shapes and trace the object's borders. This step involves the use of memory and processes the information serially, resulting in a much slower process than the previous.



(a) (b)

Figure 2.2: Optical illusions. **(a)** The central circles have the same dimensions (Hermann Ebbinghaus). **(b)** Squares A and B have the same shade of grey (Edward H. Adelson).

In the third stage, we hold the few objects of our current attention in the visual working memory. The characteristics of these elements are then sequentially acquired through visual search strategies.

Gestalt Psychology and Laws. It is not only important to know how we acquire visual information, but also to understand how we interpret it. As explained above, in the second stage of the visual perception model we recognise regions and shapes. Unfortunately, it is not always the case that what is drawn is what we see, or that what we perceive is acceptable to our mind. Optical illusions are clear examples of these misinterpretations (see figure 2.2).

Gestalt psychology[1] [20, 66] is one of the theories that studies perception and learning. With particular reference to sight, this theory describes the ability of the mind to identify more than what is present in a drawing. For example, certain drawings induce our mind to recognise patterns and shapes that are not physically present (see figure 2.3a). Also, ambiguous pictures might propose two or more valid interpretations, incompatible if considered together but acceptable when considered individually, forcing us to continuously switch between them (see figure 2.3b).

Moreover, Gestalt psychology states that we apply the same criteria to our visual perception as we use when reasoning. Our brain is more comfortable with concepts and memories that are simple, ordered and regular. As a result, we unconsciously perceive relationships between elements based on their similarity, continuity, symmetry and proximity (see figure 2.3c). These tendencies are collected and described by rules named Gestalt laws.

Getting the Most Out of Human Perception. Thanks to this knowledge on perception, we can develop visual communication methods that are more natural and intuitive for the user. The mechanics of our visual system suggests to encode information with orientation, colour, texture and other features that our eyes can easily and rapidly recognise. It also suggests to limit both the number of elements that needs to be simultaneously analysed, and in general the use of memory, as we are not able to store a lot of information over a short time.

[1] In German, *Gestalt* means "essence or shape of an entity's complete form".

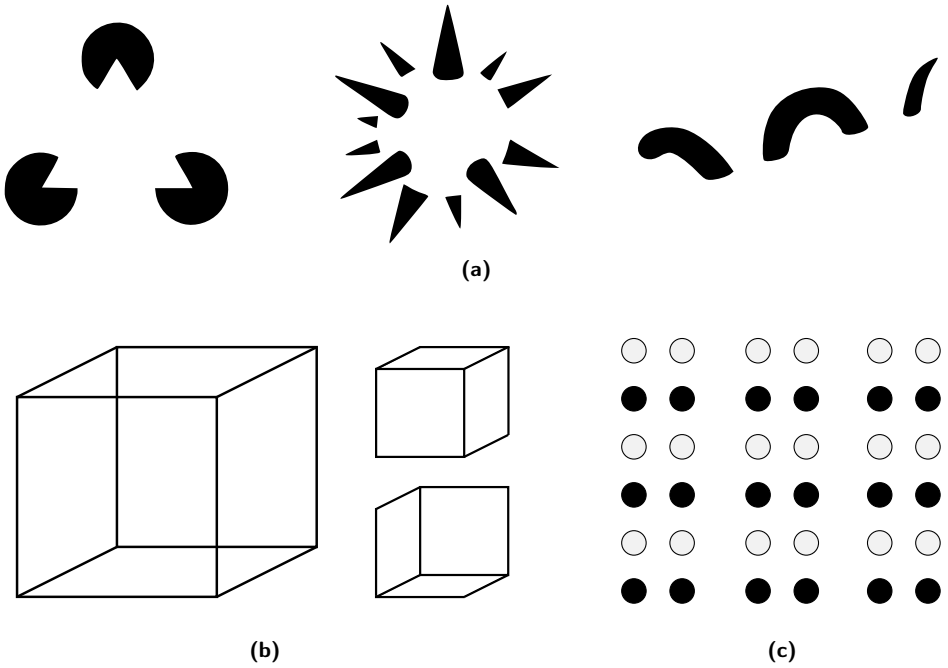


Figure 2.3: Phenomena studied by the Gestalt psychology. **(a)** We perceive shapes not actually drawn, such as the triangle in the first figure, the sphere in the second and the plane in the third. **(b)** The drawing at the left can be interpreted as a cube observed from the bottom-left or from the top-right, as reported in the two smaller pictures. **(c)** In this picture, we tend to unconsciously group the circles in rows due to their common colours, or in couples of columns due to the different vertical spacing between them.

Figure 2.4: Examples of logos featuring effects studied by the Gestalt psychology. **(a)** The IBM® logo, composed of letters not drawn completely (Property of International Business Machines Corporation). **(b)** The MacOS® logo, designed to offer the alternative interpretations of a single face or of two faces facing each other (Property of Apple Corporation).

Figure 2.5: Cave paintings in Lascaux, France.

Gestalt laws are taken into account when creating user interfaces for computer programs, so that options and elements that are logically linked also are perceived as related. Several symbols and logos are also designed according to these findings, featuring letters composed in non-standard ways (see figure 2.4a) or a drawing with multiple interpretations (see figure 2.4b) [93].

However, perception might sometimes conflict with established conventions, so that what is widely used is preferred to alternatives that are better from a perception point of view. Letters, numbers and many of the symbols we use are arbitrary, and they are not necessarily optimal in terms of perception. Even associations such as red/wrong or green/correct strongly depend on the culture, and might not be true for all people. Therefore, the importance of these perception discoveries and of their application to visual communication change with the degree of influence of established conventions:

> This model [p. 5] works well for early-stage sensory processing, where, for example, scientific colour theory has provided the basis for the display devices that are ubiquitous. It works less well at the higher levels of perception, where socially constructed codes are at least as important as innate neural mechanism.
>
> Ware [110]

2.1.2 Evolution of Visualisation

The definition in section 2.1 suggests that visualisation studies the forms of graphical communications that more efficiently transfer information to the user. The techniques used in current visualisation evolved from earlier forms of graphical communication, in the research of better ways to perform this task. We can identify several stages in this evolution.

Visual Arts. The first attempts of communication or expression via graphical representation that still exists are cave paintings (see figure 2.5). The exact purpose of these drawings is still unknown, and it is possible that sharing knowledge and feelings was not their main aim. However, later forms of visual arts developed expression and communication as their main purpose. Art aims to inspire and spread sensations, and visual art obtains this by depicting or representing concrete and abstract objects.

Modern visualisation aims to communicate a different kind of information than art, as it generally represents and expresses measures and values rather than emotions and

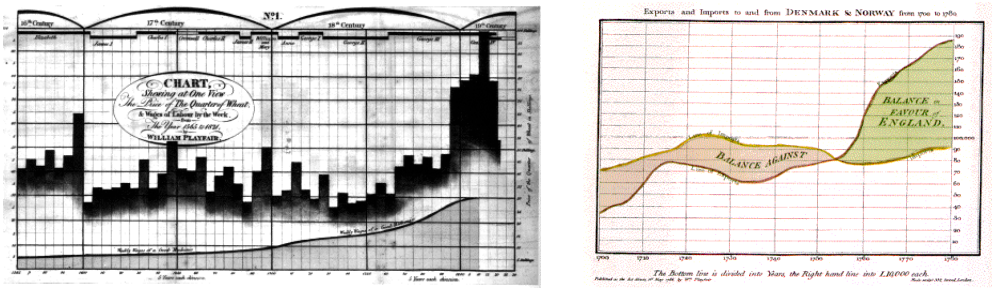


Figure 2.6: Diagrams realised by William Playfair at the end of the 18th century. Playfair introduced visualisation methods such as bar charts, line charts and pie charts, widely used even nowadays.

feelings. Its relationship with this old ancestor is however still evident: In visualisation there is a research of the aesthetics that is generally higher than in other scientific disciplines.

Maps and Diagrams. The first development of the visualisation science is often placed in the modern era, when diagrams and maps started being consistently used for reasoning:

> In the 16th century, techniques and instruments for precise observation and measurement of physical quantities were well-developed. As well, we see initial ideas for capturing images directly, and recording mathematical functions in tables. These early steps comprise the beginnings of the husbandry of visualisation.
>
> Friendly and Denis [44]

Maps and diagrams are forms of graphical communication intentionally developed to show quantities. Although present even before, they start to be used more regularly and efficiently in the modern era. Maps start to become precise enough to be used as reliable navigation instruments. Diagrams start to appear to illustrate mathematical proofs and functions, to support engineering and to describe natural phenomena [43]. A surprisingly high number of visualisation methods and conventions used nowadays have been developed in modern era drawings (see figure 2.6).

Computer Science. During the last decades, visualisation has been increasingly linked to computer science. Computers provided such large benefits in terms of quality, distribution, usability and efficiency of the visualisation methods, that current visualisation has its basis in computer science. The exact nature of the contribution of computers will be discussed more in depth in the next section.

2.1.3 Visualisation and Computer Science

Today's visualisation appears to be inseparable from computer science. The reason for such a close linking of disciplines not necessarily related, becomes clear after briefly investigating the evolution of one form of visual communication: writing.

Figure 2.7: Movable type.

Figure 2.8: Gutenberg Bible, NYPL.

Writing and the Printing Press. Writing allows us to encode information into a graphical form, so that it can be preserved and transferred to anyone able to see and interpret those signs. Writing has been a ground-breaking revolution for mankind: as the information recorded cannot be altered or misreported, they can reach people distant both in terms of space and time.

Printing drastically changed the world of written communication. Although initial printing methods had been developed much earlier in China, the real success of printing came with the independent invention of printing press and metal movable type (see figure 2.7) by Johannes Gutenberg, around 1450 A.D. These printing techniques allowed to replicate documents with very high output quality (see figure 2.8) and many times faster than handwriting, allowing to reach a much wider audience.

The introduction of the European printing press and movable type represents an enormous progress for mankind and one of the most influential inventions of the second millennium:

> What gunpowder has done for war, the printing press has done for the mind.
> Wendell Phillips

The Contribution of Computers to Visualisation. Computer graphics have the same innovative effect on visualisation as the printing press had on writing. In fact, even though drawings and diagrams were already replicable through woodblock printing or lithography, the greatest improvements in terms of distribution, re-use and quality of images came with the introduction of computers. This is probably the main, but not the only, contribution that computers give to visualisation.

First, the constant improvements of computers induces the recording and production of data at drastically increasing rates. This enormous amount of data continually requires new visualisation methods and constantly provides new challenges to solve.

Second, computers supply a much greater computational power than what is provided by the human mind. Therefore, computers can assist the user in the generation and interpretation of the drawing, for instance by aggregating and processing the raw data or by animating the changes in the visualisation.

Finally, computers introduced the possibility of interacting with the given visualisation, opening much greater opportunities than the ones offered by a static drawing. Interaction allows more sophisticated exploration and navigation of the data encoded, it greatly extends the quantity of data that can be represented and allows to generate less cluttered visualisations.

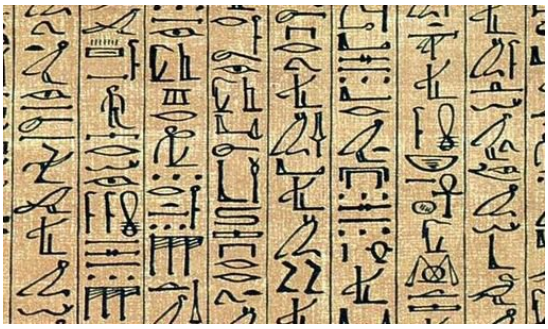


Figure 2.9: Egyptian hieroglyphs.

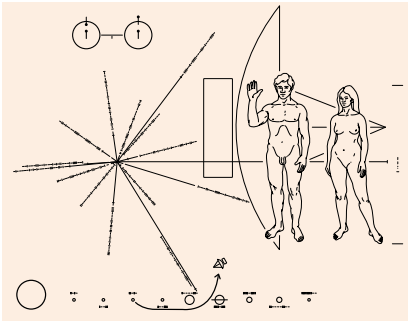


Figure 2.10: The Pioneer plaque.

2.1.4 Reasons and Goals of Visualisation

We already introduced visualisation as the science of presenting information through graphical, computer-based means. We also saw how the main concepts of this definition — visual elements and computers — greatly contribute to the expression of information. However, the concept of information has not been clarified yet. Is visualisation the answer to all kinds of communication? Should information always be presented visually?

Writing Systems. Let us again consider the case of writing. There exist several writing systems, among others logographic and alphabetic systems. In logographic systems, each symbol (logogram) represents or depicts an object or an idea, as in the case of initial Egyptian hieroglyphs (see figure 2.9) and Chinese characters. In alphabetic systems, symbols encode sounds and the writing is the direct conversion of the oral language, as it happens for Greek, Latin and languages derived of those.

Strengths and weaknesses of both writing systems provide very interesting insights into the encoding of information into visual forms. Alphabetic systems are more flexible, since they do not require us to add new symbols when the vocabulary extends, and require less time to be learnt once one knows the corresponding oral language. This happens because alphabetic writing systems strongly re-use consolidated knowledge and existing conventions coming from the spoken language.

On the other hand, logograms (and pictorial symbols in general) are more likely to be understood by people that are not familiar with those conventions (see figure 2.10) and are considered more efficient once mastered. In fact, modern logograms such as the telephone symbol ✆ or male/female restroom symbols 🚹/🚺 often replace the corresponding words when indicating these facilities.

General Versus Specific. This might suggest that there is information that is better expressed in a more explicit and pictorial way, and information that is better expressed in a more implicit way. For example, the telephone symbol is easier to understand than the word 'telephone', but the word 'nightingale' is more direct and efficient than a bird symbol that would probably look like that of many other birds. As a general rule, the more general the concept is, the more helpful becomes a pictorial symbol; the more specific the concept is, the more a precise and concise representation is preferred.

This idea is confirmed by the opinion of other researchers in visualisation. Telea [101] explains how visualisation should and should not be applied when getting insights

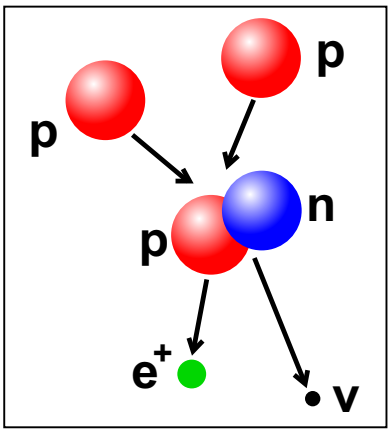


Figure 2.11: Conceptual model of a nuclear fusion process.

Figure 2.12: Physical model of Marina Bay, a newly developed area of Singapore.

on a mathematical function. If the user is interested in knowing the value of the function in a specific point, a text-based answer better achieves the result. If the user is interested in knowing the values of the function in an interval, then a plot of the function works better than a text-based answer:

> There are, indeed, many cases when [...] simple tools such as a text-based query-and-answer system work the best. However, there are also many cases when visualisation works better for answering concrete, quantitative questions. In these cases, the answer to the question is not a single number but typically a set of numbers.
>
> Telea [101, p. 5]

The concept of 'generality' also applies to the knowledge of the user. If the user is not familiar with the case of study, he might not be able to formulate precise questions. For example, a user that is not familiar with the tangent function might wonder why there is not a returned value for $\pi/2$. However, if a plot of the function is provided, he might understand why such a number does not exist and develop a better awareness of the whole function. This additional knowledge might then trigger more precise and meaningful queries.

Visualisation and Modelling. One of the most important reasons for using visualisation is to provide an overview of the case under analysis. To some extent, visualisation differs very little from creating models. Mathematicians, architects and engineers develop conceptual models (see figure 2.11) of their work to acquire a general idea of how it is structured. The latter two also produce physical models (see figure 2.12) to present their work, highlight the strong points and spot eventual parts to be improved.

The main benefit of models is to facilitate the acquisition of new, unsuspected insights. Moreover, models allow to perform top-down analysis, where the attention moves from a general overview of the problem to very specific details.

Most visualisation techniques focus on providing a computer model of the data under analysis. Exactly as it happens with the physical and conceptual models, the nature of the data influences the characteristics of the model. When dealing with

concrete objects, the computer model generally reflects our perception of the reality. When the data refers to abstract objects, the computer model will have a more schematic and structured composition, such as charts or diagrams. The top-down approach mentioned before is also very present in visualisation, as confirmed by the mantra well known by researchers in this field:

> Overview first, zoom and filter, then details-on-demand.
>
> Shneiderman [88]

2.1.5 Disciplines of Visualisation

It is common to subdivide the visualisation field into several disciplines. The two main branches, scientific visualisation and information visualisation, split the visualisation field according to the data used [10] and consequently the output produced. A third branch, visual analytics, is lately evolving from information visualisation because of its different approach to the data analysis. Being relatively new, the boundaries of information visualisation and visual analytics are still not well defined.

Scientific Visualisation. Scientific visualisation [101] is the branch of visualisation that deals with scientific and physically based data, coming from fields such as medicine, geography, physics and astronomy. As this data mainly consists of measures and attributes of concrete objects, the visualisation proposed is typically based on shape and structures of these objects, while colours and other techniques helps to detect the relevant information in the data. Scientific visualisation can be associated with physical modelling, according to the considerations made in the previous section.

Examples of scientific visualisation include the visualisation of 3D objects (see figure 2.13), the identification of objects scanned at security checks (see figure 2.14), and the depiction of atmospheric conditions (see figure 2.15). Active areas of research of scientific visualisation include vector visualisation, flow visualisation and volume rendering.

Information Visualisation. Information visualisation [10, 11, 36] deals with data not related to concrete objects or phenomena. For instance, this data might consist of economical indexes, software metrics, texts and telephone traffic data. The visualisation proposed is therefore more abstract, as was the case of conceptual models.

Examples of information visualisation include the representation of computer networks (see figure 2.16), the study of the relationships between libraries in software projects (see figure 2.17), and the analysis of biological data (see figure 2.18). The fields of information visualisation are, among others, graph and network visualisation, software visualisation and multivariate data visualisation.

Visual Analytics. Visual analytics [56, 65, 102] is characterised by a more general approach to data analysis. In visual analytics scenarios, it is required to analyse a vast amount of non-heterogeneous, dynamic and often conflicting data. Typically, the data is used for a single investigation and the analysis needs to be done in a very short time. For these reasons, visual analytics does not only consider the visual representation of the data, but a broader process that integrates previous and successive steps such as data acquisition and decision making. Thus, visual analytics aims to combine visualisation, human factors and data analysis in a unified process [65].

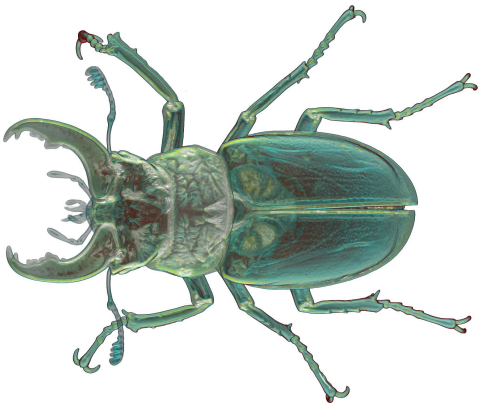


Figure 2.13: Volume rendering of a CT scan of a beetle (courtesy of Bruckner et al. [8]).

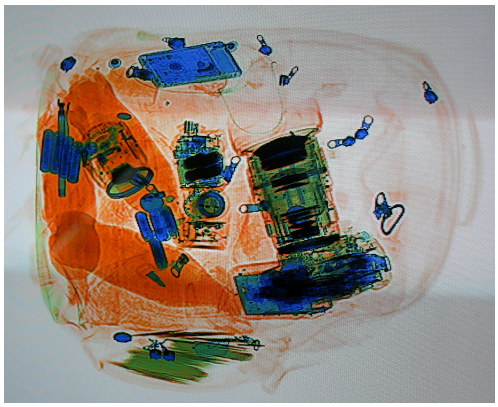


Figure 2.14: X-rays analysis of a bag (courtesy of Duke).

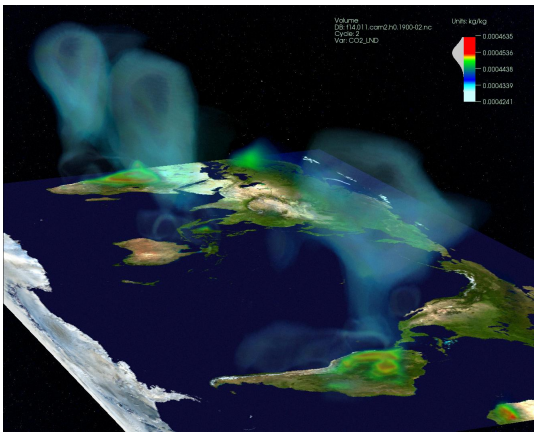


Figure 2.15: CO_2 concentration in the air (courtesy of Hoffman and Daniel).

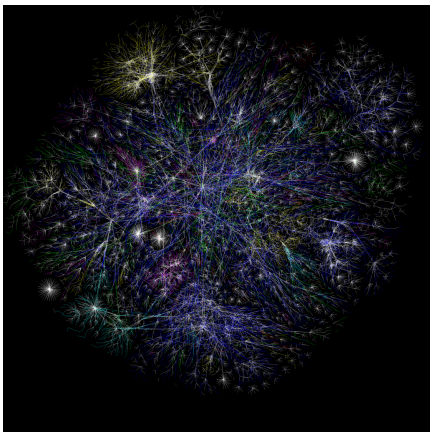


Figure 2.16: Partial map of the Internet (from OPTE project, `www.opte.org`).

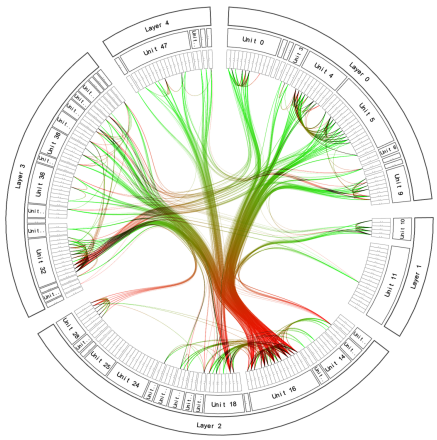


Figure 2.17: Visualisation of software structures using hierarchical edge bundles (courtesy of Holten [60]).

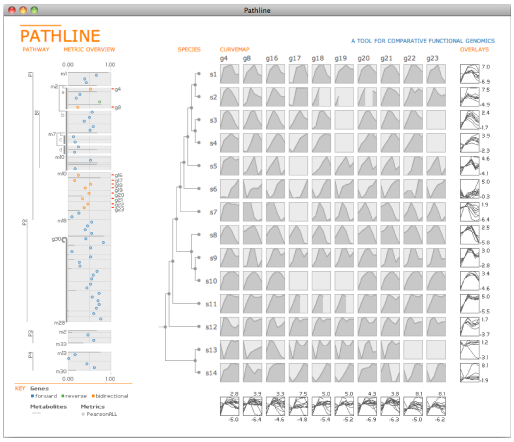


Figure 2.18: Pathline, an interactive tool for the visualisation of comparative functional genomics data (courtesy of Meyer et al. [71]).

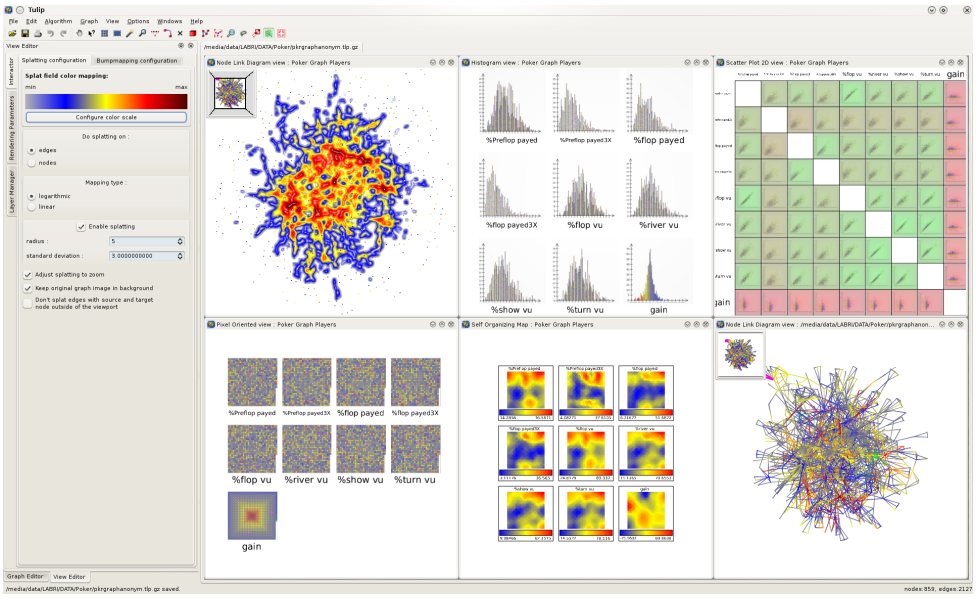


Figure 2.19: Visual analysis of poker player statistics with TULIP (courtesy of Auber [3]).

Examples of application fields of visual analytics are crime investigation and national security, analysis of financial or economical data and visual data mining (see figure 2.19).

2.2 Euler Diagrams

Consider the problem of reordering a box full of buttons. You spread the buttons over a table and you notice that the task is not trivial: the buttons have different colours, shapes, dimension and material (see figure 2.20a). At this point, you would probably proceed by taking the buttons one by one and by collecting them into piles of identical ones.

If the number of piles grows fast, it is a good idea to keep the piles of buttons of the same colour in the same corner of the table, so that finding the right pile for new buttons becomes easier (see figure 2.20b). Also, piles of buttons of the same shape should be closer, so that when you have to allocate a round button, you only need to search the right pile in the "round button" region. Ideally, the same should happen even for size and material (see figure 2.20c).

Considering the problem in more formal terms, we have a collection of elements (buttons) that are grouped into several sets (red buttons, blue buttons, round buttons, metal buttons) possibly overlapping (red round buttons, blue metal buttons). Our actions allowed us to identify regions of a plane (table) that correspond to these sets (red buttons corner, round buttons region).

The idea of dividing a surface into regions collecting homogeneous elements is the base of Euler diagrams. Euler diagrams are graphical representations widely used to depict sets and their intersections. In Euler diagrams, elements are represented by glyphs (usually dots) and sets are represented by the internal region of closed curves. Elements are placed inside the region of each of the sets they belong to. Set

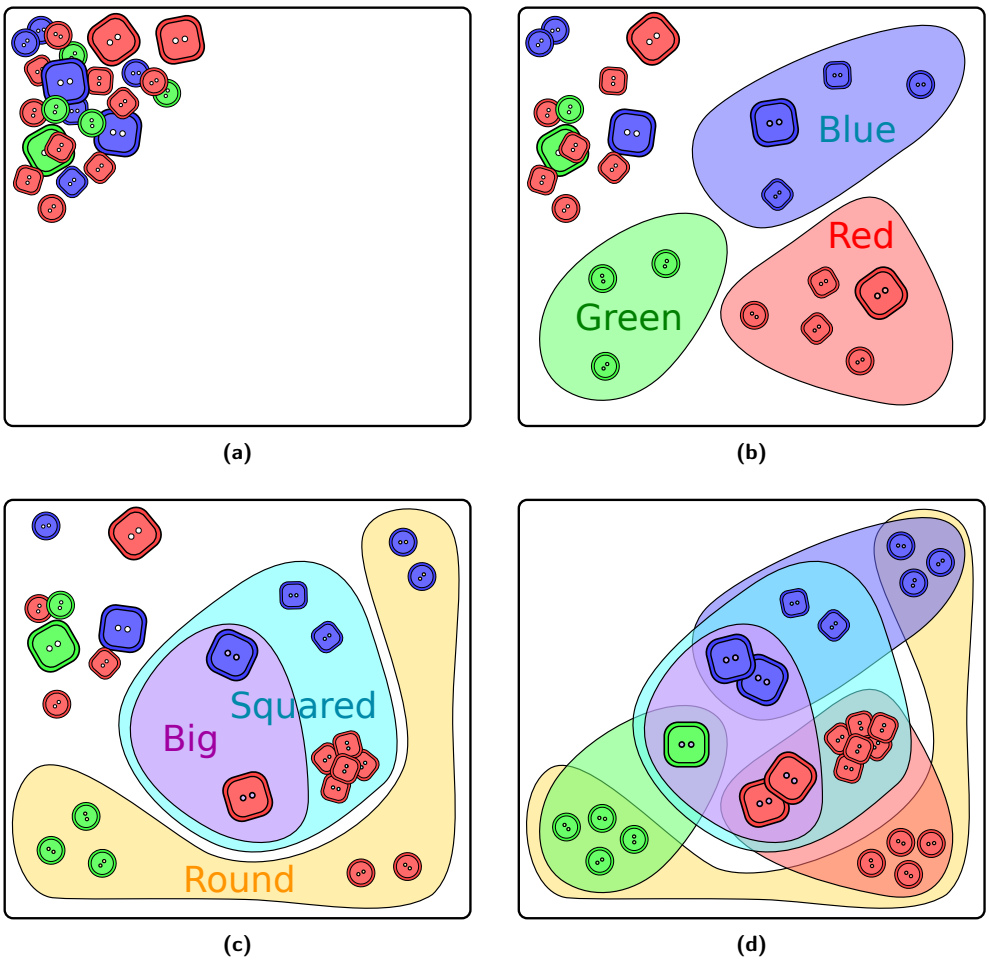


Figure 2.20: An example of an Euler diagram derived from the process of ordering a box of buttons. **(a)** The buttons to be ordered. **(b)** Displacement of the buttons according to their colour. **(c)** Further organisation of the space according to size and shape of the buttons. **(d)** The Euler diagram that can be extracted from the process.

intersections are therefore depicted by overlapping set regions identified by crossing curves.

Figure 2.20d shows an Euler diagram built over the button example described above. In this case, we chose to collect the elements in six sets: red buttons, blue buttons, green buttons, big buttons, round buttons and squared buttons. The diagram provides a very efficient overview of the set relationships. For example, we can clearly see that the set of big buttons is included in the set of squared buttons, meaning that the only big buttons we possess are squared. Also, we can see that the sets of squared and of round buttons do not overlap, as we obviously do not possess buttons that are round and squared at the same time. Finally, we can note how the intersection between the sets of green and of squared buttons is fully included in the set of big buttons, meaning that we do not possess green squared buttons that are not big.

2.2.1 The Original Diagrams

Modern Euler diagrams derive from diagrams developed in the 18th century to reason on syllogisms. These diagrams, called Euler circles, take their name from the Swiss mathematician Leonhard Euler (see bibliography in appendix A), who is commonly considered their inventor.[2] In the 19th century, these diagrams have been formalised and re-elaborated by the English statistician John Venn (see bibliography in appendix A).

Euler Circles. While living in Berlin, Euler was asked by Frederick the Great of Prussia to tutor his niece, the Princess of Anhalt-Dessau. Since they were not able to meet in person for the lessons, Euler wrote over 200 letters to her between 1760 and 1762, covering a very broad spectrum of mathematical and physical disciplines. The letters were collected and published in several languages [7, 16, 33] and became very popular due to the competence of the author and the clarity of the explanation.

In some of these letters [7, vol. 1, letters 102–105], Euler treats the topic of propositions and syllogisms. A proposition is described as a judgement, such as "all men are mortal" or "no man is righteous". Propositions link two concepts, such as men/-mortality and men/rightness, through the judging process: in those sentences, it is stated where or where not mortality or rightness is applicable to men. Propositions can be universal and particular, and can be affirmative or negative:

- "every A is B" is affirmative universal (e.g. "all men are mortal"),
- "no A is B" is negative universal (e.g. "no men are righteous"),
- "some A is B" is affirmative particular (e.g. "some men are learnt"),
- "some A is not B" is negative particular (e.g. "some men are not wise").

Euler proposed the graphical representations shown in figure 2.21 to better understand the relationship between the concepts involved. The diagrams were also used for syllogisms, that is drawing conclusions from certain given propositions. For instance, given the prepositions "all oaks are trees" ("every A is B") and "all trees have roots" ("every B is C"), we can conclude that "all oaks have roots" ("every A is C"), as shown by the diagram in figure 2.22a. If we are instead given the propositions "every A is B" and "some C is A", we can conclude that "some C is B". In this case we cannot fully characterise the relationship between C and B, as C could be totally or partially included in the notion B. Again, the diagrams help to draw the correct conclusions, as shown in figure 2.22b.

Venn Circles. About one century later, John Venn [104, 106] surveyed the existing graphical methods to depict propositions and syllogisms. Some of the main criticisms he raised on Euler circles include the lack of formality and of an algorithmic procedure to produce them.

Venn formalised Euler's representation by linking it to the Boolean logic. Even in Venn circles each of the concepts involved in a syllogism is associated with a circle, but the circles always intersect each other in order to depict all their possible intersections.

[2] The authorship of these diagrams actually dates further back. As mentioned by Hamilton [57, vol. 2, p. 180], the diagrams already appeared in a work by Weise [113], that was published in 1712 some years after his death.

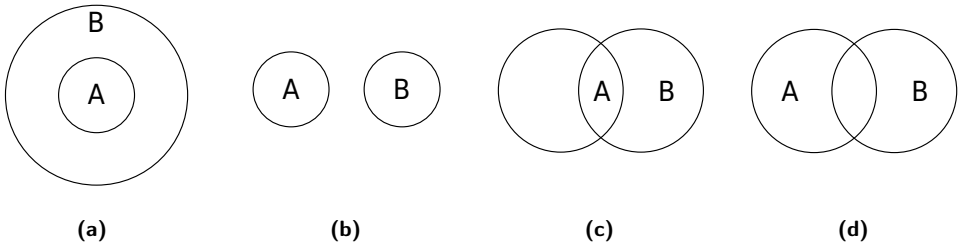


Figure 2.21: Original Euler circles representing the four proposition cases. **(a)** Affirmative universal case. **(b)** Negative universal case. **(c)** Affirmative particular case. **(d)** Negative particular case.

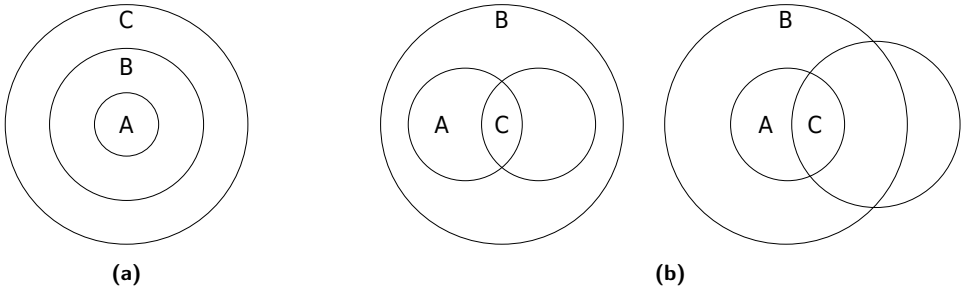


Figure 2.22: Original Euler circles representing two syllogisms. **(a)** The case of two affirmative universal propositions. **(b)** The case of an affirmative universal proposition and an affirmative particular proposition. Since C is not completely defined, there exist two possible scenarios.

These intersections are labelled with Boolean algebra minterms[3] that depend on the circles they belong to. For example, given that a indicates that the intersection is included in circle A and $\bar{\mathsf{a}}$ that it is not, we have

- for one circle, A, two intersections, a and $\bar{\mathsf{a}}$,
- for two circles, A, B, four intersections, ab, $\bar{\mathsf{a}}\mathsf{b}$, $\mathsf{a}\bar{\mathsf{b}}$ and $\bar{\mathsf{a}}\bar{\mathsf{b}}$,
- for three circles, A, B, C, eight intersections, abc, $\mathsf{ab}\bar{\mathsf{c}}$, $\mathsf{a}\bar{\mathsf{b}}\mathsf{c}$, $\mathsf{a}\bar{\mathsf{b}}\bar{\mathsf{c}}$, $\bar{\mathsf{a}}\mathsf{bc}$, $\bar{\mathsf{a}}\mathsf{b}\bar{\mathsf{c}}$, $\bar{\mathsf{a}}\bar{\mathsf{b}}\mathsf{c}$ and $\bar{\mathsf{a}}\bar{\mathsf{b}}\bar{\mathsf{c}}$.

Intersections that are excluded by the proposition or syllogism are eventually shaded out. For example, the proposition "every A is B" implies that there is not an instance of A that is not in B, thus excluding the intersection $\mathsf{a}\bar{\mathsf{b}}$. The proposition "no A is B" instead excludes the intersection ab. Figure 2.23 shows a comparison of the representations proposed by Euler and Venn on these cases.

2.2.2 Modern Euler and Venn Diagrams

In the last decades, the representations developed by Venn and Euler have been intensively used to depict sets rather than propositions and syllogisms. Although the

[3] A minterm is a product of boolean variables in which each variable appears only once, either in complemented or uncomplemented form.

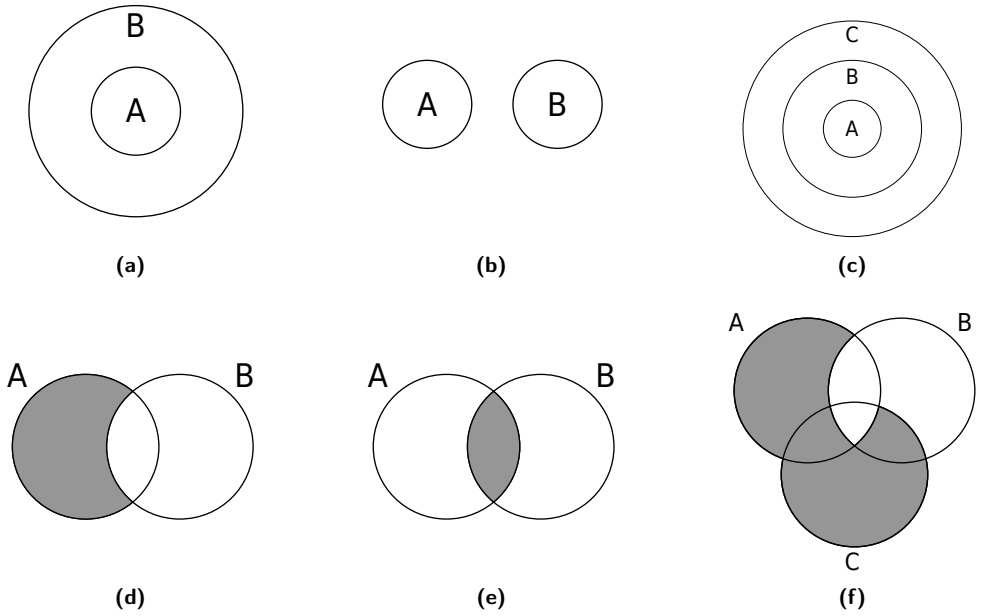


Figure 2.23: Comparison between Euler and Venn circles. **(a-c)** Euler circles of the propositions in figure 2.21 and syllogism in figure 2.22a. **(d-f)** The corresponding Venn circles.

main structure and idea of the initial diagrams have been preserved, the different application field and the usage of computer graphics have introduced some variations in these drawings.

Depiction of the Set Elements. Even when using Euler and Venn diagrams for set representation, set elements may or may not be present in the drawing. When elements are not inserted, the diagrams only show how sets overlap with each other, or in other words, which sets share elements between them and which do not. Thus, Euler diagrams must only depict set intersections that are not empty, and Venn diagrams must shade all set intersections that are empty.

In the case that elements are actually depicted, the previous constraints are sometimes slightly relaxed. Euler diagrams might include set intersections that are empty, because the lack of elements clarify this. The same happens for Venn diagrams, as the diagram does not become equivocal if the empty intersections are not shaded.

The choice of using either the first, strict approach or the second, more relaxed one changes the drawings significantly. Drawing diagrams according to the more rigorous approach is much harder, as it is strictly necessary to avoid undesired overlaps. The rigorous approach also causes higher concurrency and less regular shapes of the set boundaries. On the other hand, the strict approach is necessary when the focus is not on the set elements or when the set elements are not defined (see figure 2.24).

Graphical Improvements. Initial diagrams only relied on closed lines to distinguish the regions of the plane associated with sets or concepts. Modern diagrams are drawn using graphical techniques that aim to improve their clarity:

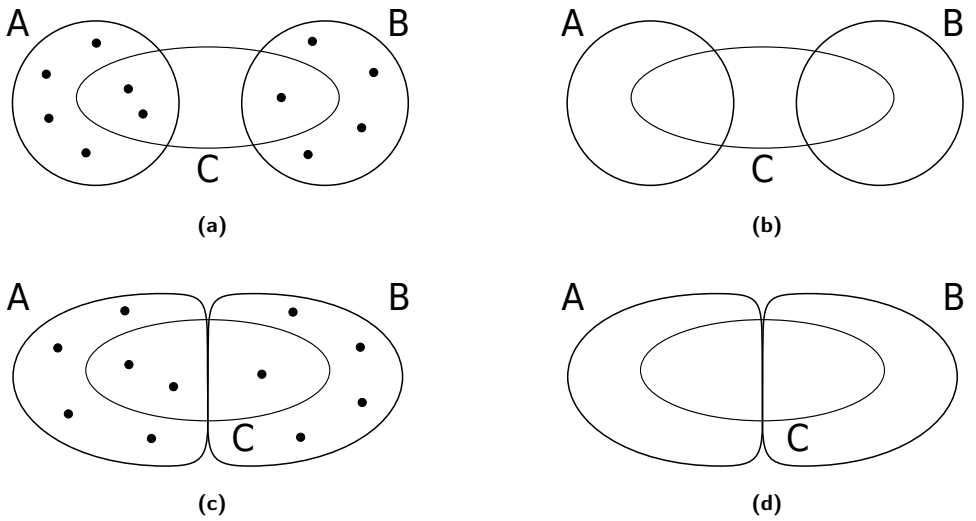


Figure 2.24: Euler diagrams, with and without the depiction of the set elements, generated according to a rigorous and a relaxed approach. **(a-b)** Using a relaxed approach. In the second diagram, it is impossible to realise that the central region of set C is empty. In the first diagram, the depiction of set elements solves the ambiguity. **(c-d)** Using a strict approach. Neither of the diagrams are ambiguous, but it was necessary to modify the set boundaries A and B introducing concurrency and less regular shapes.

- *Colours.* By using different colours for the set boundaries, the identification of the region associated with a set is made easier (see figure 2.25b). Colours become particularly important when the shape of the sets is not regular, when sets intersect in intricate ways and in the case of concurrency between the set boundaries.

- *Shaded Regions.* The clarity of the diagram is further improved by colouring the regions identified by the set boundaries (see figure 2.25c). By using semi-transparent colours, the overlapping regions will assume a colour that is a blend of the original ones, helping to retrieve the sets involved. Shaded regions are particularly important when very irregular boundary shapes make it difficult to identify the region associated with the set.

- *Area-proportionality.* When depicting sets, it is important to provide information such as the number of elements contained inside each zone. In some modern Euler and Venn diagrams, we tend to associate regions with a high number of elements to a larger area of the drawing, and regions with low number of elements to a smaller area (see figure 2.25d). This is generally desired in all the cases where the number of elements contained in each region is available, since this kind of drawing provides interesting information even when the focus is not on the set elements. Unfortunately, area-proportional drawings are difficult to obtain and might require less regularly shaped sets.

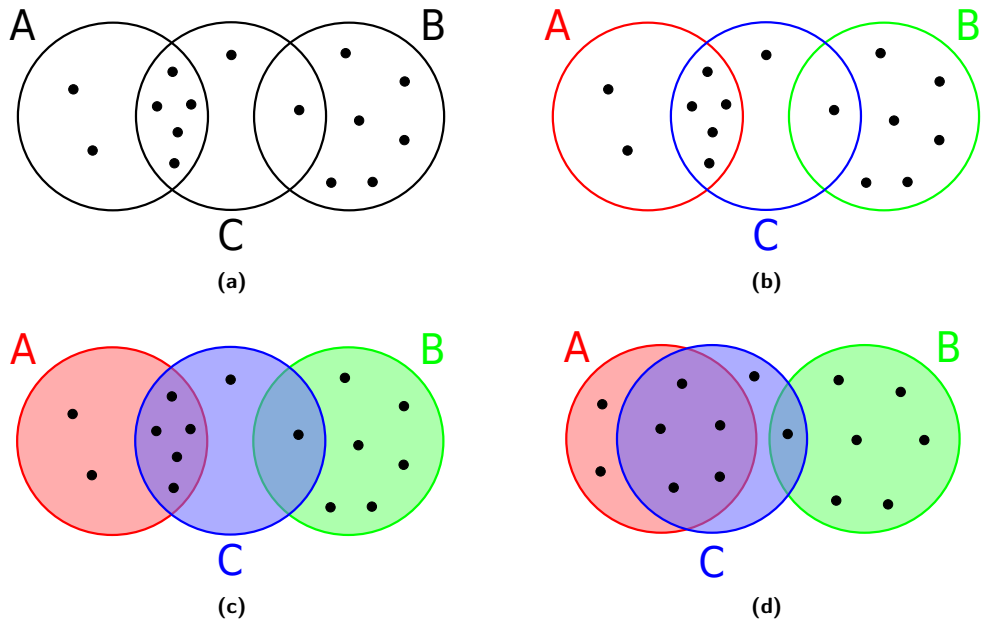


Figure 2.25: Graphical improvements present in modern Euler diagrams. **(a)** A plain Euler diagram. **(b)** Adding coloured set boundaries. **(c)** Adding semi-transparent coloured set regions. **(d)** Adding area-proportionality.

2.3 Data Visualisation with Euler Diagrams

Euler diagrams are extremely valuable tools in information visualisation and visual analytics: not only do they make good use of our perception system, but they can also be used to easily retrieve non-trivial information from complex data.

In this section, we first detail the technical reasons that motivate the previous statement. Then, we provide an example of visual data analysis that benefits from this kind of diagrams. Finally, we state some of the limitations of Euler and Venn diagrams.

2.3.1 Euler Diagrams and Perception

By analysing Euler diagrams from a perception point of view (see section 2.1.1), we can detect their theoretical benefits and drawbacks:

- *Gestalt Laws.* The concepts at the base of Euler diagrams is the division of the plane into continuous regions that contain similar objects. When the shape of the boundaries are relatively simple and regular, we will respect the Gestalt laws of continuity of the set boundaries and proximity of the set elements.

- *Use of Memory.* Compared to an alternative visualisation where sets are drawn individually and placed apart from each other, Euler diagrams require a much lower use of the memory. In fact, independently drawn sets would require to replicate the shared elements, increasing the amount of information to elaborate. Euler diagrams represent a lower bound in terms of objects represented, using a single graphic for each set and for each set element.

More importantly, in independently drawn sets the identification of intersections, inclusions and exclusions requires to continuously switching sight between the set representations. Euler diagrams instead provide all this information locally, limiting the need to observe different scenes at once.

- *Established Conventions.* Euler diagrams are so widely spread that they are the universal *de facto* standard for the visual representation of sets. This virtually eliminates the problems related to the presence of multiple standards and the need of explaining how to interpret the visualisation.

- *Region Identification.* A possible drawback of Euler diagrams is in the identification of the set regions. According to the perception model presented in section 2.1.1, the reconstruction of objects (sets) by merging the regions that compose them is a slow and serial process. There is not a lot that we can do from this point of view, other than further discriminate related from unrelated regions with colours and other graphical techniques (see section 2.2.2). As this represents a main limitation of Euler diagrams, we will discuss the problem in greater detail in section 2.3.3.

As long as the diagram is not too complicated, the last point is negligible and the benefits stated at the first two points make Euler diagrams highly intuitive. The third point is an advantage of Euler diagrams, but needs to be taken into serious consideration when proposing extensions that significantly change their concept or functioning.

2.3.2 An Example of Data Exploration with Euler Diagrams

Most of the diagrams shown so far depicted a very limited number of sets and elements. This might legitimise the idea that Euler diagrams can only show trivial information, making them unhelpful when it is necessary to acquire insight from more complex data. In this section, we provide an example of how visual data investigation can be greatly supported by Euler diagrams.

Case Study. The *Group of Six* (G6) is a forum created in 1975 by some of the most industrialised democracies of the world: France, West Germany, Italy, Japan, United Kingdom, and United States. The forum later expanded by including Canada (1976, becoming G7) and Russia (1997, becoming G8). The *Group of Twenty* (G20) is instead the forum of the twenty more developed economies of the world: Argentina, Australia, Brazil, China, European Union, India, Indonesia, Mexico, Saudi Arabia, South Africa, South Korea, Turkey and all the G8 countries.

We built an Euler diagram that shows the behaviour of these countries with respect to several important aspects of developed countries and democracies (see figure 2.26). The diagram contains the set of the G8 countries, the set of the G20 countries and sets containing the best performing countries for each of the following indicators:[4]

- *Research and development.* We considered the percentage of GDP invested in research and development in year 2008 [78], year 2010 [51] and the forecast of this indicator for 2011 [51]. For each year, we collected the countries that performed better than the G20 country average in a set. The three sets are equivalent and are depicted in the diagram with the set labelled "R&D".

[4]The European Union has been excluded as it includes some of the G20 countries and because the aggregate indicator for the European Union is often missing in the considered surveys.

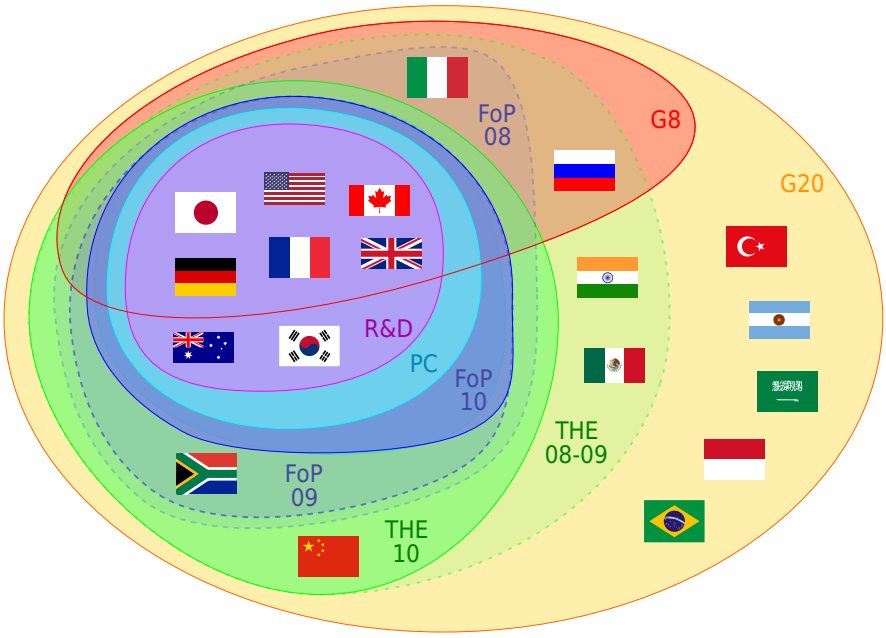


Figure 2.26: An Euler diagram showing the G20 countries and the best performing countries in terms of research and development (R&S), education (THE), press freedom (FoP) and corruption (PC) for years 2008–2010.

- *Higher education.* We considered the Times Higher Education's university ranking for years 2008–2010 [99]. For each year, the surveys include a list of the top 200 universities. By grouping the G20 countries that have at least one university in the top 200 chart, we obtained the set "THE 08–09" (years 2008 and 2009) and the set "THE 10" (year 2010).

- *Freedom of the press.* We considered Freedom House's survey on press freedom for years 2008–2010 [34]. In this survey, countries are marked either free, partially free or not free according to the degree of press freedom. We collected the G20 press free countries in the sets "FoP 08" (year 2008), "FoP 09" (year 2009) and "FoP 10" (year 2010).

- *Perceived corruption.* We considered Transparency International's corruption perception index for years 2008–2010 [100]. The index measures the level of public sector corruption perceived by the citizens of the country. For each year, we grouped the countries with an index lower than the G20 average. The resulting sets are equal and correspond to set labelled "PC".

Data Exploration. The diagram shows four indicators, each for three years, and two sets showing the G8 and G20 countries, for a total of 14 classifications and 9 displayed sets (five sets coincide with others and are therefore merged). Despite the relatively high volume of data, each set of the drawing can easily be identified. Moreover, by using increasing transparency for older surveys we can depict the temporal evolution of the sets, while giving more emphasis on the recent data.

The diagram allows to both confirm expected behaviours and to extract significant and unexpected information. As expected, most of the G8 countries have excellent results for all the chosen indicators, while less developed countries have lower performance for most of them. The most evident unexpected insights are regarding Italy and Russia: contrarily to the rest of the G8 countries, they have poor performance for all the indicators in year 2010. This is particularly surprising for Italy, that behaves very differently from culturally and geographically close countries as France, Germany and United Kingdom.

The set evolution over time also shows that the gap between the distinguished countries (Japan, USA, Canada, UK, Germany, France, South Korea and Australia) and the remaining ones is increasing for some of the indicators. This result could either derive from better performances of the first countries or worse performances of the second ones.

Euler Diagrams as an Information Visualisation Tool. In conclusion, the diagram allowed to easily deduce significant insights by visual inspection. Complex and very different information such as political relationships, country average behaviour and the temporal evolution of statistical indicators is presented in a natural and intuitive way. The representation is compact, requires little memory usage and respects the standard established conventions for set representation. Thanks to the generality of Euler diagrams, these benefits are automatically extended to every application that involves overlapping collections of elements.

2.3.3 Limitations of Euler and Venn Diagrams

Despite their high flexibility, Euler diagrams have limitations that are difficult to overcome. The most significant one is related to the intrinsic complexity of the set intersections to be represented.

When sets overlap in a very intricate way, the number of different regions to be displayed tends to increase very rapidly. When all the possible intersections need to be depicted, Euler diagrams correspond to Venn diagrams and the number of regions is exponential with respect to the number of sets. As mentioned in section 2.3.1, the region identification is also a perception bottleneck for Euler diagrams. Thus, as even John Venn stated, Venn diagrams are helpful only when depicting very few sets:

> Beyond five terms [sets] it hardly seems as if diagrams, of the particular kind here described, offer much help, but we have seldom occasion to trouble ourselves with problems which would introduce more than that number.
>
> Venn [105, p. 117]

Therefore, when Euler diagrams are used to depict more than a few sets that overlap in a very intricate way, the representation will necessarily be unintuitive and confusing (see figure 2.27). Note that this is not necessarily related to the number of sets or the number of elements to be represented: we might not be able to draw a nice six sets Venn diagram, and still be able to draw clear and informative Euler diagrams for hundreds of sets and thousands of elements.

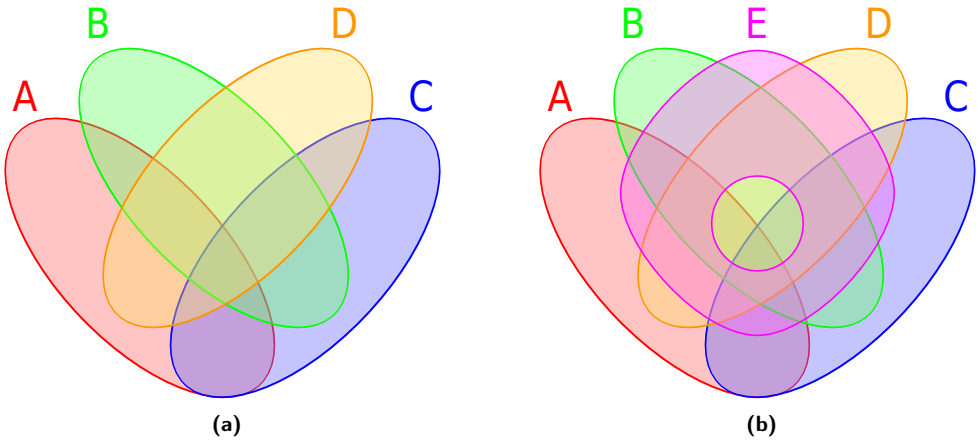


Figure 2.27: Diagrams proposed by John Venn [105] to depict four and five sets. He discouraged the production of these diagrams for more than five sets. **(a)** The diagram for four sets. **(b)** The diagram for five sets. In order to be drawn using simple shapes, set E contains a hole in the region inside its main boundary.

Chapter 3

Graph and Euler Diagram Theory

This chapter presents the background theory required to understand the work discussed in this thesis, the notation used, and references for further information.

First, we introduce the mathematical entities called graphs and the related theory. We put particular emphasis on the concepts associated with planarity, as they are strictly related to the generation of Euler diagrams.

Second, we introduce some rudiments of algorithmics and the field of graph drawing. In particular, we focus on the the subfields of planar graph drawing and force-directed drawing, presenting their evolution and their most known algorithms.

Finally, we continue the discussion on Euler diagrams, this time from a formal point of view. After introducing the basic definitions, we investigate the relation between mathematical sets and their representations in Euler diagrams, the diagram properties and the issues that might occur when drawing them.

In the following, the concepts are first introduced in an informal way, in order to provide the readers that are not familiar with the terms a smoother introduction. Then, in the boxed paragraphs, the definitions and the notation are explained in a more rigorous and formal way. Also, the mathematical notation is collected in the list on page xx and the definitions are inserted in the index on page 179.

3.1 Graph Theory

In this section, we introduce the foundations of graph theory. Graphs are mathematical tools used to represent relational data, that is, information on the relationship between entities. Each graph is composed of two kinds of elements: the nodes, that represent the entities, and the edges, that are associated with two nodes and express the presence of a relation between the two entities.

For instance, a graph can be used to represent friendship relations between a group of people. In this case, each node represents a person, and each edge indicate the friendship between the two associated people. Therefore, people who do not know each other will not have an edge between them, and a close group of friends might have an edge connecting any two individuals in the group.

Graphs can also be used to represent the street map of a city. Here, nodes might be used to indicate junctions or locations and edges might be used to represent streets.

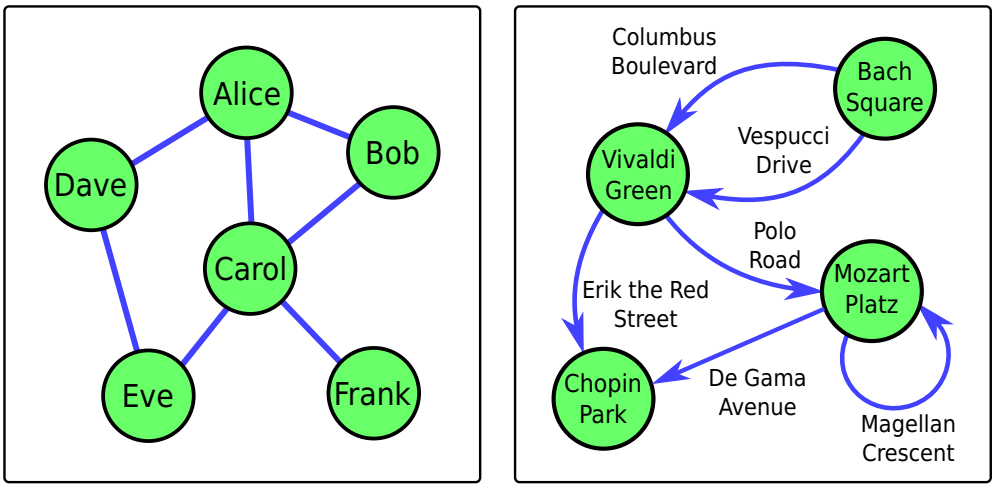


Figure 3.1: Examples of graphs and relative node-link diagrams. **(a)** A graph depicting the friendship relations in a group of people. **(b)** A graph showing the connections between locations in a city.

Thanks to the high flexibility of these tools, there are no problems in taking particular cases into consideration, such as multiple streets connecting the same two squares, streets that start and leave from the same location and one-way streets.

Node-link Diagrams. When dealing with graphs, it is natural to work with one of their diagrammatic representations. The most common representations, *node-link diagrams*, depict the nodes with glyphs (usually circles) and the edges with arcs connecting the involved nodes. Using these conventions, we can depict the graph examples mentioned above in a very direct and intuitive way (see figure 3.1).

In this thesis, we will also refer to a node-link diagram as the *drawing* of a graph. Drawings are characterised by the *drawing style* chosen to represent the edges. In particular, in a *straight-line drawing* the edges are straight segments connecting the centre of the nodes, in a *polyline drawing* the edges are polygonal chains, and in a *Jordan drawing* the edges are Jordan arcs.

The Origin of Graphs. Graphs are another of the many inventions attributed to Leonhard Euler (see biography in appendix A). In 1735 he published a paper about the problem of the Seven Bridges of Königsberg. The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel river and in correspondence with two major islands on the river. The four landmasses were connected by seven bridges (see figure 3.2a).

It was speculated that there was a way to cross every bridge of the city without crossing the same bridge twice. Euler proved this was wrong by abstracting the configuration of the city into a graph-like structure. As the internal movement in each of these land masses is irrelevant, each land mass can be thought of as a single entity. As the bridges connect two different land masses, they can be thought of as links between these entities. Nowadays, we would abstract the land masses into

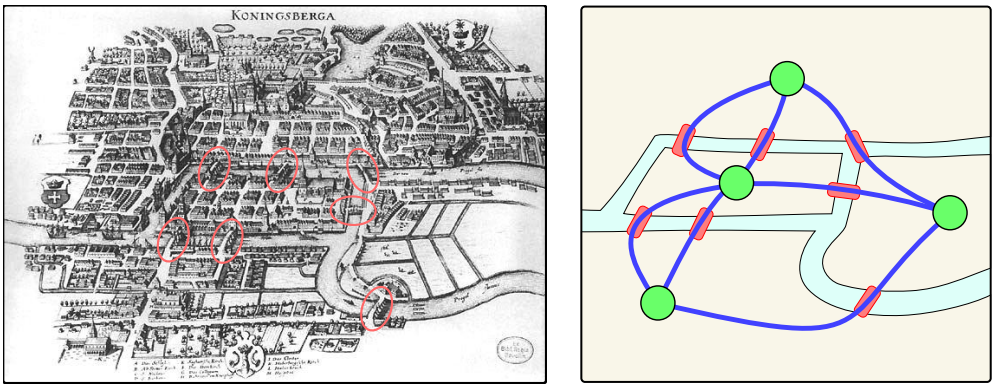


Figure 3.2: The problem of the Seven Bridges of Königsberg. **(a)** A map of Königsberg as it was in 1651 (by Merian-Erben), with highlights on the seven bridges. **(b)** The graph that abstracts the connections between the land masses of the city.

graph nodes, the bridges into graph edges, and we would depict the problem using the node-link diagram in figure 3.2b.

Euler realised that whenever you cross a bridge to reach a land mass, you also need to cross a bridge to leave it. Therefore, except at most for two land masses (the one where you start your walk and the one where you finish it), all the landmasses need to have an even number of bridges to be able to cross them all. As Königsberg had four land masses, all connected by an odd number of bridges, the problem does not have a solution. In modern graph theory, a path that uses once and only once all the edges of a graph is still named after Euler.

Theoretical Foundations. In the remaining part of this section, the graph theory concepts used in the thesis will be presented and discussed in greater detail. Due to this focus, we refer to dedicated graph theory textbooks [25, 63] for a more complete survey on the matter.

3.1.1 Sets, Multiset and Tuples

A collection of elements can be ordered and unordered, and can allow or forbid element repetitions. The sequence in which the elements are listed is irrelevant in unordered collections, while it is typifying in ordered collections. At the same time, element repetition can be forbidden, so that each element can be listed only once, or allowed, so that the same element can be listed several times in the collection.

Sets, multiset and tuples are collections of elements that mainly differ from each other for their behaviours regarding these aspects:

- *sets*: unordered, forbid repetitions. Identified by curly brackets, e.g. $\{a, b, c\}$.
- *multisets*: unordered, allow repetitions. Identified by squared brackets, e.g. $[a, b, b]$.
- *tuples*: ordered, allow repetitions. Identified by round brackets, e.g. (b, c, a, b).

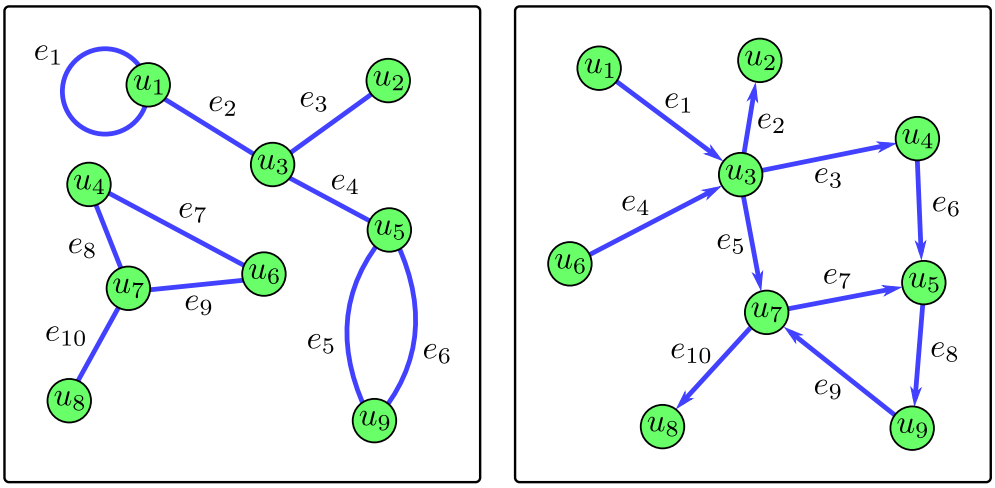


Figure 3.3: Examples of undirected and directed graphs and their notation. **(a)** An undirected graph. The edge e_1 is a loop, and e_5 e e_6 are multiple edges. The graph is therefore a multigraph. **(b)** An example of a directed graph. As it does not contain loops or multiple edges, the graph is said to be simple.

3.1.2 Graphs and Their Classification

Graphs are mathematical entities that extend the concept of sets to allow the expression of relationships between the set elements. To obtain this aim, a mathematical set is augmented by additional information on eventual relations between its elements. In standard graphs, the nature of these relationships is binary and only involves two elements at a time: it is then possible to express the presence or the absence of a relation between any two elements of the original set.

Graphs can either be directed or undirected. In directed graphs, all the edges have an orientation: each edge implies a relationship between the first element and the second, but not between the second and the first. In undirected graphs, each edge express a reciprocal relationship between the nodes involved.

Graphs can also differ for the kind of edges they contain. Multiple edges are edges that appear more than once in a graph. Loops are edges that depict a relationship between a node and itself. Graphs that do not contain these kinds of edges are said to be simple, while graphs that allow them are called multigraphs. Examples of these kinds of graphs are shown in figure 3.3.

Graphs. A *graph* is a tuple $G = (V, E)$ composed of a set V of *nodes* or vertices and a multiset E of *edges* or links, where the graph edges are either multisets or tuples of size two containing graph nodes.

Nodes and edges of a graph are said to be the *elements* of the graph. We indicate with u, v, or with u_i, $i \in \mathbb{N}$, generic nodes of a given graph $(u, v, u_i \in V)$ and with e, or with e_i, $i \in \mathbb{N}$, generic edges of the same graph $(e, e_i \in E)$.

Directed and Undirected Graphs. A graph can be directed or undirected. In a *directed graph*, each edge is a tuple of two nodes, $e = (u, v)$. In an *undirected graph*, each edge is a multiset, $e = [u, v]$, where the order of the nodes does not matter. Therefore, $(u, v) \neq (v, u)$ and $[u, v] = [v, u]$. In this thesis, when not specified, graphs should be considered undirected.

Simple Graphs and Multigraphs. When a graph contains two or more equal edges, these are called *multiple edges*. An edge (u, u) or $[u, u]$ is instead called a *loop*. Graphs with no loops and no multiple edges are called *simple graphs*. Graphs where loops and multiple edges are explicitly allowed are called *multigraphs*.

Graph Dimension. The *dimension* of a graph is a measure related to the number of elements of the graph. In this thesis, we associate the dimension of a graph with the number of nodes it contains, indicating as *small* a graph with less than approximately one hundred nodes, *medium* a graph with a hundred to a thousand nodes, and *large* a graph with more than a thousand nodes.

Indications on the number of edges will be expressed as a function of the number of nodes. We call *sparse* a graph where $|E|$ is approximately equal to or smaller than $|V|$, and *dense* a graph where $|E|$ is in the order of $|V|^2$.

3.1.3 Subgraphs

Subgraphs are graphs formed by a portion of the elements contained in another graph. Subgraphs are generally obtained by deleting some of the nodes and edges of an original graph. As the new collection of nodes and edges must still be a graph, it is not possible to keep the original graph edges whose extremities have been discarded.

A particular kind of subgraphs are induced subgraphs. These graphs are formed by a subset of the original graph nodes and by all the original edges that join these nodes.

Subgraphs. A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$. An *induced subgraph* of an undirected graph G is a subgraph G' where $E' = \{e = [u, v] \in E : u \in V' \wedge v \in V'\}$. The induced graph of a directed graph is defined similarly.

3.1.4 Relations Between Nodes and Edges

We can refer to nodes and edges with terms that indicate a particular relationship with other graph elements. For instance, nodes can be the extremities of an edge and adjacent to or neighbours of other nodes. We also introduce the concept of source and target of an edge, that better suits directed edges but that can be used even in the case of undirected graphs.

The degree of a node indicates the number of edges that are related to that node. In directed graphs, it is possible to obtain different degree measures by considering incoming and outgoing edges, while for undirected graphs there exists a single measure for the node degree.

Appellations of Nodes and Edges. Given a directed edge $e = (u, v)$ or an undirected edge $e = [u, v]$, we call u and v *extremities* of e and we say that e is *incident* to u and v. In a graph G, two nodes incident to the same edge, as well as two edges incident to the same node, are said to be *adjacent*. The nodes adjacent to u in G are called *neighbours* of u.

Given a directed edge $e = (u, v)$, the first node is called the *source* of e and it is identified by $s(e) = u$. The second node is called the *target* of e and it is identified by $t(e) = v$. In the case of undirected edges, source and target can still be used to refer to one and to the other of the two nodes, but neither term refers to a particular extremity.

The Degree of a Node. Given a directed graph G, the *out degree* of a node u indicates the number of times u is a source in the graph, thus $\deg_{\text{out}}(u) = |\{e \in E : u = s(e)\}|$. The *in degree* is instead the number of times a node is a target, thus $\deg_{\text{in}}(u) = |\{e \in E : u = t(e)\}|$.

For both directed and undirected graphs, the *degree* of a node u is the number of edges having u as extremity, counting loops twice. In directed graphs it corresponds to the sum of the out degree and in degree, $\deg(u) = \deg_{\text{out}}(u) + \deg_{\text{in}}(u)$.

A node with degree zero is identified as *isolated*.

3.1.5 Walks, Paths, Cycles and Distance

We can consider the edges of a graph as links that allow us to move from one node to another. Directed edges can be considered as one-way links, that only allow to move from the source to the target of the edge. Instead, undirected edges can be considered as links that allow movement in both directions. It is then possible to define the concepts of walks, paths and cycles as particular sequences of movements in the given graph.

Conceptually, a walk is a sequence of movements that leads from one node of the graph to another. A path is a walk where you never pass the same node twice, except in the case where your final movement brings you to the initial node. A cycle is a path where this latter condition occurs, or in other words, a path that brings you back to the initial node. A chain is a maximal path in which there is no possibility to decide the movement, since all nodes except the extremities only have two edges: one to access them, and one to leave them. Examples of walks, paths, cycles and chains are shown in fig. 3.4.

Walks. A *walk* is an alternating sequence of nodes and edges $u_0 e_1 u_1 \dots e_l u_l$ of a graph G, beginning and ending with nodes, in which each edge has the previous node in the sequence as source and the following node as target. The length of the walk is l, and corresponds to the number of edges in the sequence. When not ambiguous, a walk can be indicated by listing only the nodes $u_0 u_1 \dots u_l$ or only the edges $e_1 \dots e_l$ of the sequence. When the first and the last node coincide, $u_0 = u_l$, the walk is said to be *closed*.

Paths, Cycles and Distance. A *path* is a walk where all the nodes of the sequence are distinct, except possibly u_0 and u_l. A closed path is a *cycle*. In

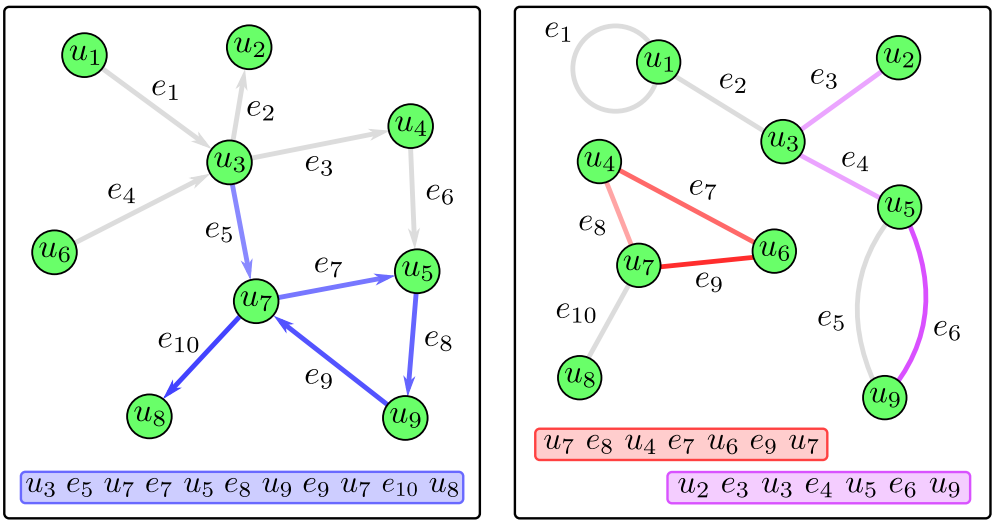


Figure 3.4: Examples of walks, paths and cycles in directed and undirected graphs. The direction of the walks is rendered with decreasing values of luminosity on the coloured graph edges. **(a)** An example of a walk in a directed graph, coloured blue. The walk is not a path, since it passes through the node u_7 twice. **(b)** Examples of paths and chains in an undirected graph. Since the red path starts and ends with the same node, it is a cycle. The red path is also a chain, since $\deg(u_7) \neq 2$ and $\deg(u_4) = \deg(u_6) = 2$.

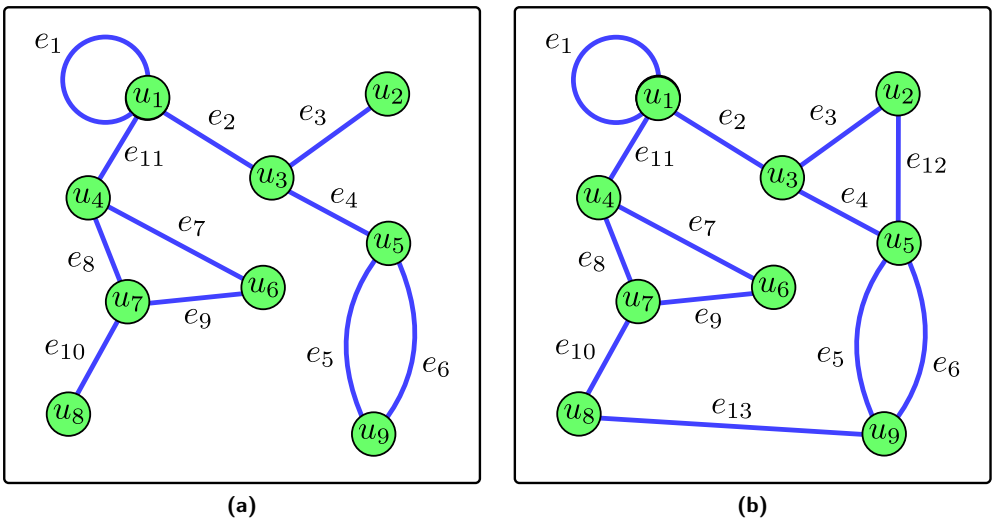


Figure 3.5: Examples of graphs with different connectivity. The original graph from which these graphs are derived (figure 3.3a) is disconnected. **(a)** The graph is made connected by adding the edge e_{11}. The graph can still be disconnected by removing only one node (u_1, u_3 or u_4 for example), therefore its connectivity is one. **(b)** The graph is made 2-connected by adding edges e_{12} and e_{13}. The connectivity of the graph is two, since the graph can be disconnected by removing, for instance, the pair u_3, u_8.

a graph G, the *distance* $\text{dist}(u, v)$ between two nodes u and v is defined as the length of the shortest path between u and v in G.

Chains. A *chain* is a path in which the nodes u_0 and u_l have degree different from two, and all the other nodes have degree two.

3.1.6 Connectivity

The connectivity of a graph is a property that expresses how strongly connected its nodes are. A graph is connected if every node can be reached from any other by moving over the edges. When this does not happen, the graph is disconnected and the connected portions of the graph are called connected components.

A graph is k-connected when it is necessary to remove at least k nodes to disconnect the graph. A k-connected graph can be thought of as a graph where each node is simultaneously connected to the others in k or more different ways. Examples of graphs with different connectivity are shown in figure 3.5.

Connectivity. A graph G is *connected* if there is a path for any two distinct graph nodes u and v. A graph that is not connected is *disconnected*. When a graph is disconnected, its maximal connected subgraphs are called *connected components*. The set of the connected components of a graph is indicated with C, and a generic connected component is indicated with c, or with c_i, $i \in \mathbb{N}$.

The *connectivity* $\kappa(G)$ of a graph G is the minimum number of nodes that needs to be removed to make G disconnected. A graph is k-connected when $\kappa(G) \geq k$.

3.1.7 Trees and Forests

Trees and forests are particular kinds of undirected graphs that have several important applications. Trees are connected graphs without cycles. When a node of a tree is selected as root, the graph nodes are automatically organised into a hierarchy dominated by the chosen node. Forests, an extension of trees, are simply graphs composed of one or more independent trees.

Since hierarchical organisations are quite common, these graphs developed specific appellations for their elements. The root is the top node of a tree. A node that is controlled by another node, higher in the hierarchy, is called descendant. The controlling node is instead called ancestor. When this relationship is direct, that means there are no intermediate nodes between ancestor and descendant, the two nodes are called parent and child. Examples of forests, trees and of the related notions are shown in figure 3.6.

Trees and Forests. A *forest* is an undirected graph without cycles. A *tree* is a connected forest. A *rooted tree* is a tree where one of the nodes is distinguished from the others.

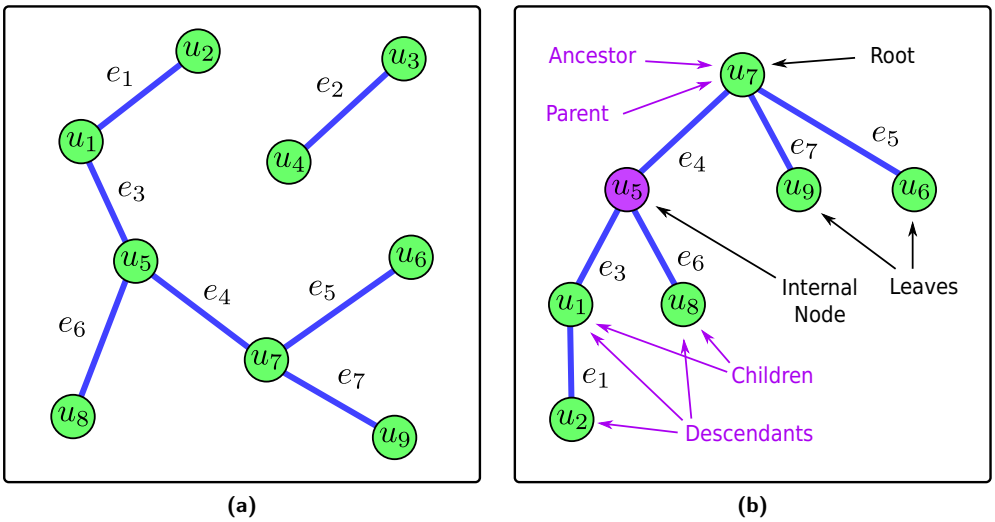


Figure 3.6: Examples of trees and forests. **(a)** A forest composed of two trees. **(b)** The biggest tree of the previous picture rooted on the node u_7. In purple, the hierarchical notions with respect to the highlighted node. In black, the notions that do not depend on the choice of a node.

Appellations of Tree Nodes. The *root* is the distinguished node of a rooted tree. Every node v that appears after a node u in a path that starts from the root is a *descendant* of u, while u is an *ancestor* of v. When two of such nodes are consecutive in the paths, then u is the *parent* of v and v a *child* of u.

A node with no children is called a *leaf*. A node with children is instead called an *internal node*.

Levels. The *level* $\text{lev}(u)$ of a node u is its distance from the root. When $G = (V, E)$ is a forest, we identify with V^i the set of nodes that have level i.

3.1.8 Complete, Bipartite Graphs and Subdivisions

Complete and bipartite graphs are other particular and important kinds of graphs. Complete graphs are simple graphs where all the nodes are directly connected to each other. Bipartite graphs are instead graphs where the nodes can be divided into two groups, such that all the edges only connect a node of a group with a node of the other group.

Subdivisions are operations that substitute an edge of a graph with a new sequence of nodes and edges. Subdivisions of complete and bipartite graphs will be used in a further characterisation of graphs. Examples of these graphs and of subdivisions are presented in figure 3.7.

Complete and Bipartite Graphs. A *complete graph* is a graph where every two different nodes are adjacent. A complete graph with i nodes will be identified

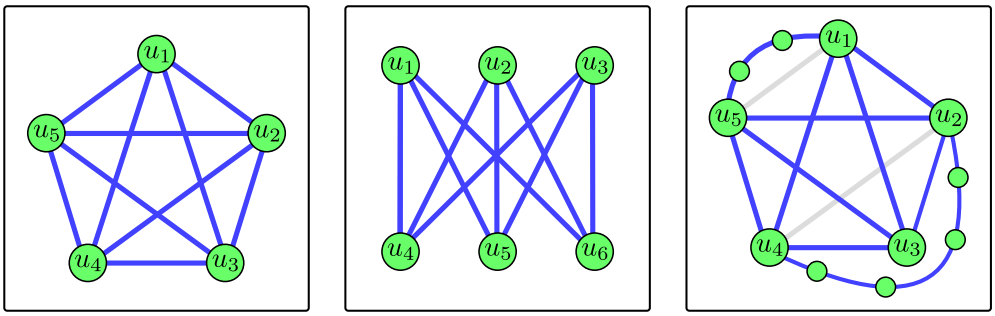


Figure 3.7: Examples of complete graphs, complete bipartite graphs and of the subdivision operation. **(a)** The complete graph with five nodes, K_5. **(b)** The complete bipartite graph $K_{3,3}$. **(c)** The application of subdivisions on the first graph: the shaded edges have been replaced by sequences of new nodes and edges.

with K_i. A *bipartite graph* $G = (V, E)$ is a graph where the nodes can be grouped in two disjoint sets $V', V'' \subseteq V$ so that every graph edge is incident to a node in V' and a node in V''. A *complete bipartite graph* is a bipartite graph where every two nodes u, v, with $u \in V'$ and $v \in V''$, are adjacent. The complete bipartite graph with $i = |V'|$ and $j = |V''|$ will be identified with $K_{i,j}$.

Subdivisions. A *subdivision* is the operation of removing a graph edge (u, v) or $[u, v]$ and adding new nodes $u_1 \dots u_l$ and edges $e_0 \dots e_l$ so that all the new nodes have degree two, and so that they create the path $u, e_0, u_1 \dots u_l, e_l, v$.

3.1.9 Planar Graphs

For some graphs, called planar, it is possible to provide drawings where the graph edges do not cross each other. An embedding of a graph on the plane, or planar drawing, defines these drawings by listing the position of nodes and edges on the plane.

For non-planar graphs, instead, it is not possible to obtain these drawings as the graphs contain some critical structures that cannot be represented without crossings. The two basic critical structures, that are also the smallest non-planar graphs, are K_5 and $K_{3,3}$ (see figures 3.7a and 3.7b). Non-planar graphs are all linked to these structures, as all of them and only they contain a subdivision of K_5 or $K_{3,3}$.

A planar drawing of a graph divides the plane into regions bounded by nodes and edges. These regions, called faces, allow us to define the concepts of equivalent embeddings and planar maps. Equivalent embeddings are drawings that are theoretically different, but substantially similar. For instance, a drawing where all the nodes and edges are shifted right for the same distance is only theoretically different, as it is practically equivalent to the original. We will then say that two embeddings are equivalent if they produce faces with the same boundaries. The classes of equivalent embeddings, that are the groups that collect all the embeddings equivalent between them, are called planar maps. Examples of faces and of equivalent and non-equivalent embeddings are reported in figure 3.8.

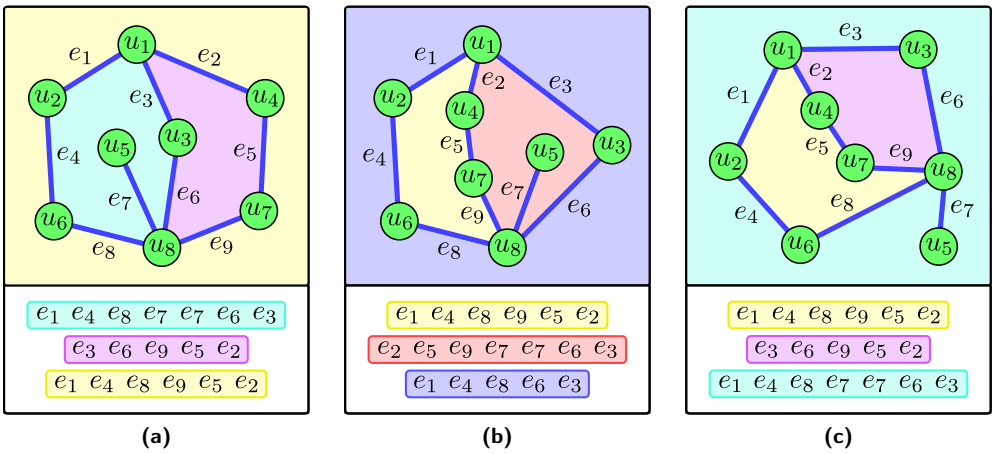


Figure 3.8: Examples of graph faces and embeddings, both equivalent and non-equivalent. **(a)** A planar drawing of a graph with faces highlighted in different colours. The sequences at the bottom of the drawings indicate the boundary edges of each face. **(b)** A non-equivalent embedding of the same graph. The yellow face of the first drawing is preserved, but the other two do not correspond. **(c)** An equivalent embedding of the first graph. All the faces of the first drawing correspond to the current faces.

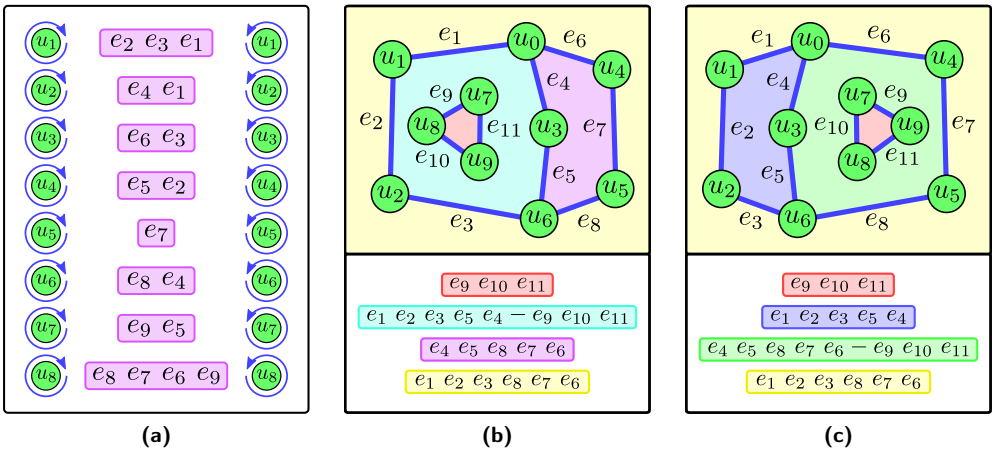


Figure 3.9: Examples of combinatorial embeddings and planar maps. **(a)** The combinatorial embeddings of the first and third drawings of figure 3.8. All the nodes have been scanned in clockwise order in the first case, and anticlockwise in the second case. As the graph is connected and the embeddings are equivalent, the combinatorial embeddings of the drawings correspond. **(b)** A planar drawing of a disconnected graph. In this case, one of the faces is identified by two separated sequences of edges. **(c)** A non-equivalent embedding of the same graph. The faces in the two drawings do not correspond even though the combinatorial embedding has not changed.

When a planar graph is connected, it is possible to characterise a planar map by providing a combinatorial embedding. A combinatorial embedding is a data structure that collects the ordering of the incident edges of each graph node, all clockwise or all anticlockwise.

When a graph is instead disconnected, this one-to-one correspondence is missing, as the same combinatorial embedding might correspond to different planar maps. Disconnected graphs contain two or more connected components, and these components can be moved to different faces of other components. This operation changes the boundary of the faces, but leaves the combinatorial embedding unaltered. Examples of combinatorial embeddings and of their relationships with planar maps are shown in figure 3.9.

Planar and Plane Graphs. An *embedding* of a graph G on a surface Σ is a representation of G where the nodes are associated with points of Σ, the edges with arcs that start and end at the coordinate of the relative nodes, and where the arcs do not cross with other arcs or nodes except that at their endpoints. In this thesis, we will always consider Σ to be a plane and we will call an embedding on such a surface a *planar drawing*.

A graph that admits a planar drawing is called *planar graph*. A graph is not planar if and only if it contains subgraphs that are subdivisions of K_5 or $K_{3,3}$ (*Kuratowski's Theorem*, 1930). A planar graph where a planar drawing has been fixed is called *plane graph*.

Faces. A planar drawing of a graph divides the plane in topological connected regions called *faces*. We indicate with f, or with f_i, $i \in \mathbb{N}$, a face of a plane graph and with F the set of all the faces in the drawing. Each face is enclosed by a boundary $\mathcal{B}(f)$ of nodes and edges. We denote the set of the boundary nodes of f with $\mathcal{B}^n(f)$ and the set of boundary edges with $\mathcal{B}^e(f)$.

Equivalent Embeddings and Planar Maps. Two embeddings of G are *equivalent* if the boundary of each face in an embedding corresponds to the boundary of a face in the other. The *planar maps* of G are classes of equivalence of the planar embeddings of G.

Combinatorial Embeddings. A *combinatorial embedding* of G is a data structure that lists in order the edges incident to the graph nodes. When G is connected, its planar maps correspond to its combinatorial embeddings. When G is disconnected, the same combinatorial embedding might correspond to different planar maps.

Properties of Planar and Plane Graphs. In a connected plane graph, we have that $|V| - |E| + |F| = 2$ (*Euler's Formula*, 1750). A *maximal planar graph*, or triangulated graph, is a connected graph where no other edge can be added without breaking the planarity. Any embedding of these graphs have each face bounded by exactly three edges. As a consequence of the Euler's formula, in any planar graph with more than three nodes, we have that $|E| < 3|V| - 6$.

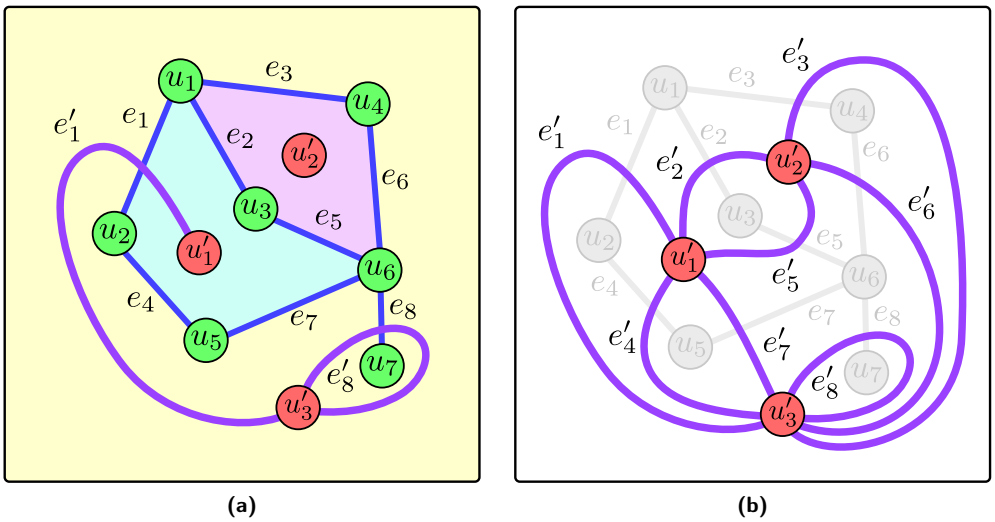


Figure 3.10: Example of a dual graph. **(a)** Positioning of the dual nodes and of two dual edges. As e_8 belongs only to one face, its dual e'_8 is a loop. **(b)** The dual graph.

3.1.10 Dual Graphs

The dual graph is a connected plane graph built over another connected plane graph. Let G be the original graph. The dual graph G' have a node for each face of G, and an edge for each edge of G. The edges of G' connect the faces whose boundaries contain the corresponding edge in G. For example, if the edge e appears in the boundaries of the faces f_1 and f_2 of G, then the dual edge e' will connect the dual nodes u'_1 and u'_2 associated with those faces. When e appears only in the boundary of one face of G, then e' will be a loop on the node related to that face. Each node u'_i is placed inside the corresponding face f_i, and each edge e'_j is routed to cross once the corresponding edge e_j and only that.

Non-equivalent embeddings of the same graphs leads to different dual graphs. However, we obtain the original graph by applying the dual transformation twice, so the dual of G' is G. Also, the dual graph is generally a multigraph, even when the original graph is simple. An example of dual graph is shown in figure 3.10.

Dual Graph. The *dual graph* of a connected plane graph $G = (V, E)$, with set of faces F, is a plane graph $G' = (V', E')$ so that

- for each $f_i \in F$, there is a node $u'_i \in V'$ $(V' = F)$,
- for each $e_i \in E$, where $e_i \in \mathcal{B}^e(f_j), \mathcal{B}^e(f_k)$, there is an edge $e'_i = [u_j, u_k]$,
- each u'_i lies in the region of f_i,
- each e'_i crosses once e_i and only that edge.

Properties of Dual graphs. If G' is the dual of G, then G is the dual of G'. Also, for a plane graph G and its dual G', we have that $|V'| = |F|$, $|E'| = |E|$ and $|F'| = |V|$.

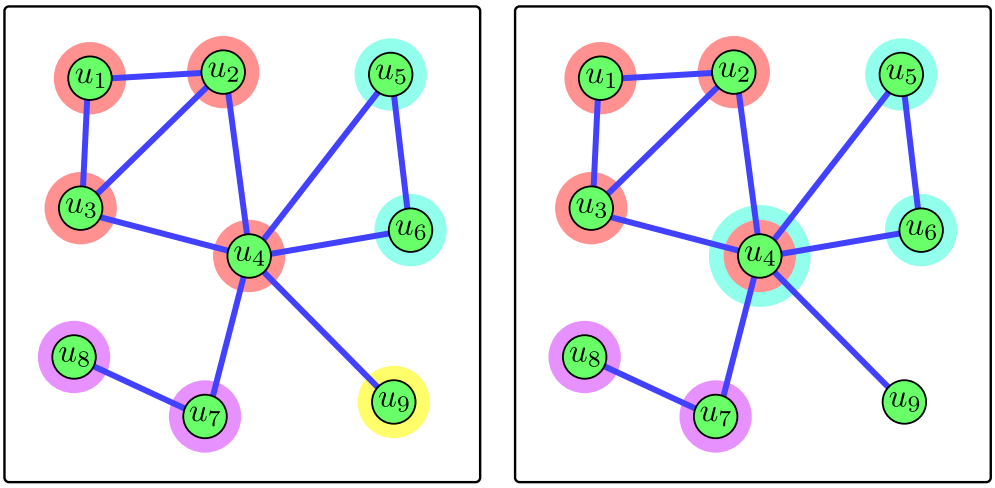


Figure 3.11: Example of a clustered graph, with different clusterings. The clusters are identified by coloured crowns around the nodes. **(a)** A strict clustering of the graph, where all nodes belong to exactly one cluster. **(b)** An overlapping clustering, where u_4 belongs to two clusters and u_9 to zero clusters.

3.1.11 Clustered Graphs

In clustered graphs, the nodes are assigned to sets that collect similar elements. Each of these sets is called a cluster and might collect, for example, nodes that share the same property or nodes that are tightly connected by graph edges. We call clustering the set of all the clusters identified in a graph, so that we can refer to different clusterings when two or more attempts of grouping the graph nodes generates non-corresponding clusters.

Traditionally, the clustering is a partitioning of the graph nodes, where each graph node is assigned to exactly one cluster. In recent years, this strict classification of the graph nodes has been considered too limiting, because some nodes might not naturally belong to any cluster or might fit into multiple ones. We refer to strict clustering in the first case, and to clustering or overlapping clustering in the second case (see figure 3.11).

Clusters and Clustered Graphs. A *cluster* is a set of nodes $s \subseteq S$ of a graph $G = (V, E)$. A *clustered graph* is a graph $G = (V, E, S)$ where $S = \{s_1, s_2 \ldots\}$ contains one or more clusters.

Strict and Overlapping Clustering. The set S of clusters of $G = (V, E, S)$ is called *clustering*. A *strict clustering* is a partitioning of V, or in other words a clustering where

- $s_i \cap s_j = \emptyset$ for all $s_i, s_j \in S$ and $i \neq j$,
- $\bigcup_{s_i \in S} s_i = V$.

A clustering where these conditions are not enforced, and in particular where the graph nodes can belong to multiple clusters, is called an *overlapping clustering* or a fuzzy clustering.

3.2 Graph Drawing

Having a good drawing is essential to easily understand the structure of a graph. The difference between a clear and an unclear drawing mostly resides in the choice of the drawing style and in the positioning of the nodes and edges. It is in fact very difficult to understand the structure of a graph, even when small and simple, if these aspects are not chosen carefully.

The determination of a good drawing becomes increasingly challenging when considering larger and more complex graphs. Moreover, when dealing with graphs of hundreds or thousands of elements, the manual placement of the graph elements is no more an option. For these reasons, a very broad and active branch of graph theory, called graph drawing, is dedicated to the automatic generation of visual representations of graphs.

In this section, we will first introduce some rudiments of algorithmics. Then, we will present a few graph drawing algorithms and concepts that are relevant to the work presented in this thesis. For a broader and more complete overview of this discipline, we suggest some dedicated textbooks [25, 75] or literature reviews on the subject [24, 31, 59].

3.2.1 Foundations of Algorithmics

When finding a solution to a problem, humans use, to various extents, intuition, judgement and common sense instead of a strict mathematical approach. However, there are situations in which having a well defined sequence of operations to execute, called an algorithm, is helpful if not necessary: we are not likely to follow a strict procedure when ordering a coffee, but we do follow a well defined sequence of steps when multiplying two large numbers on a piece of paper. Computers, that are not provided with the talents mentioned above, only rely on algorithms to solve problems.

Not all the problems have or will have an algorithm that solves them. There are problems, even common and very important, for which it is proven that an algorithm cannot exist. Luckily, this is not true for the vast majority of the problems we are asked to deal with.

Algorithms. An *algorithm* is a finite and well defined sequence of instructions that solves a specific problem.

Complexity. It is generally possible to provide many different algorithms for a problem. These algorithms might be very different in the way they approach the problem and, more importantly, in the computational effort required to provide the solution. For example, efficient algorithms to evaluate the graph planarity generally use a fraction of a second to compute the answer, while an algorithm based on the Kuratowski's Theorem, on the same graph, might require a computation time estimated in centuries.

Since algorithms are independent from the particular input provided and the structure of the computer where they run, it is necessary to define some abstractions in order to evaluate their efficiency. As a first abstraction, all the operations that require a very limited and constant time, such as the numerical operations of sum and multiplication, are identified as basic and considered all alike. As a second abstraction, we express the computation time with regard to the input provided. In fact, this measure of efficiency should take into account that it takes ten times longer to retrieve the greatest number in a set if we provide a set that is ten times bigger than the previous.

The reasons why there might be such a difference in the execution time as that mentioned above are rarely related to how many basic operations are performed, but instead to how many times a sequence of operations is repeated. For instance, in the case of an algorithm based on the Kuratowski's Theorem, we would need to control whether or not a subgraph of the input graph is a subdivision of K_5 and $K_{3,3}$. Here the problem is not on the number of basic instructions required to perform this check, but on the number of possible subgraphs to be controlled: there are in fact $2^{|E|}$ possible subgraphs, which is a huge number even for relatively small graphs.

The asymptotic notation $O(\cdot)$ takes into account all of these concepts, as it ignores constant factors (such as the number of basic operations), it focuses on how many times a computation is repeated and it expresses the complexity with respect to the size of the input. This information is collected in the argument of the notation. For instance, a graph algorithm with complexity $O(|V|)$ is likely to execute some basic operation for each node present in the graph, while an algorithm with complexity $O(|V|^2 + |V| \cdot |E|)$ probably executes basic operations for each node-node and node-edge pair of the graph. Note that in the second case, an eventual term $|V|$ would not appear in the notation as it is dominated by the other terms.

Asymptotic Notation. For a pair of real positive functions $f(n)$, $g(n)$, with $n \in \mathbb{N}$, we will use the *asymptotic notation* $f(n) \in O\big(g(n)\big)$ if there are two positive constants c_1, c_2 such that $f(n) \leq c_1 g(n) + c_2$ for every n.

Complexity. The *complexity*, or more specifically time complexity, of an algorithm is the number of basic operations required by the algorithm to provide the solution, expressed as a function of the size of the input, and reported in asymptotic notation.

Letting n be the size of the input, we will say that the complexity of an algorithm, or simply that an algorithm itself, is *logarithmic* if the complexity is $O(\log n)$, *linear* if it is $O(n)$, *quadratic* if it is $O(n^2)$, *polynomial* if it is $O(n^i)$, $i \in \mathbb{N}$, and *exponential* if it is $O(a^n)$, $a \in \mathbb{R}$, $a > 1$.

The previous definitions apply similarly to quantities other than the number of basic instructions, when these are explicitly indicated.

3.2.2 Aesthetics of a Graph Drawing

It is very difficult to define or measure the quality of a drawing. An aspect of a drawing that might help the comprehension of a particular drawing might actually be undesired in other cases. For instance, even the choice of the drawing style, among those stated at page 28, is not trivial: there are cases where polyline drawings are preferable to Jordan drawings, and cases where the opposite is preferable.

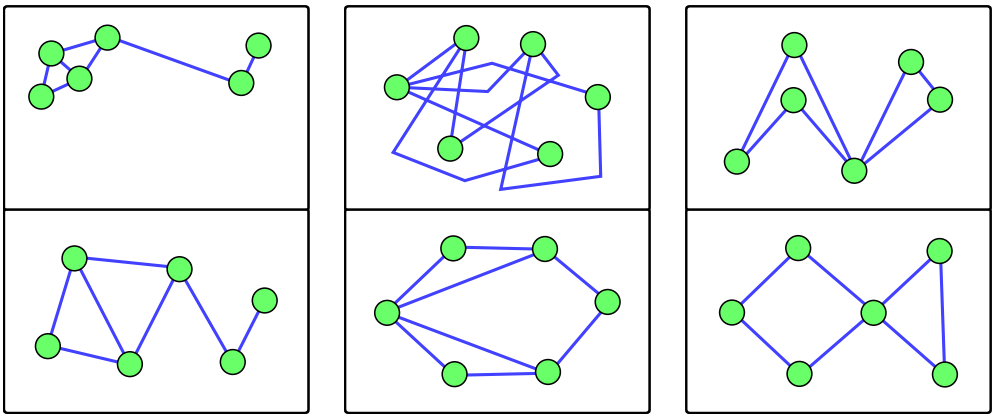


Figure 3.12: Examples of drawings that do and do not respect the aesthetics rules. At the top, drawings that do not present some aesthetic properties. At the bottom, drawings of the same graphs that do respect them. **(a)** Area, aspect ratio and edge length. **(b)** Bends and crossings. **(c)** Shape of the faces, symmetry and angular resolution.

However, there are properties that are widely recognised to be increasing the clarity of the graph:

- *Area*: a graph drawing should make good use of the space, avoiding too cluttered and too empty portions of the drawing. This allows to obtain a compact drawing without using excessively high zoom values.
- *Aspect ratio*: a drawing should have little difference between the vertical and the horizontal dimension. This discourage unevenly compressed drawings.
- *Edge length*: the length of the edges should present little variation with respect to the average edge length.
- *Bends*: in a polyline drawing, both the total number of bends and the number of bends for a single edge should be kept as low as possible to make edges easier to follow.
- *Crossings*: crossings between the edges should be reduced as much as possible, as they can generate confusion and disturb the identification of the edges.
- *Shape of faces*: the faces of the drawing should be as close as possible to regular polygons or other simple shapes.
- *Symmetry*: the presence of symmetries in the structure of the graph should be shown as much as possible in the graph drawing.
- *Angular resolution*: the angle between the edges incident to a graph node should be as high and as regular as possible, to reduce possible confusion between the edges in proximity to the graph nodes.

Examples of drawings that do and do not respect these properties are shown in figure 3.12.

Several authors have investigated the understanding of graph drawings in relation to these aesthetics properties. Some studies confirmed the Gestalt laws (see section 2.1.1) in terms of perceived importance and the relationship between the graph nodes due to their position or symmetry in the drawing [19, 69]. Other studies analysed the importance of some of these properties in the correct comprehension of the drawing [79, 80]. The results prove the importance of crossings reduction, which was the most important factor, and of bends reduction and symmetry exposing, that were also decisive. The other factors were either not taken into account or were found not statistically significant.

3.2.3 Graph Drawing Algorithms

Despite having standard rules that characterise the aesthetics of a graph drawing, the generation of these visual representations is a problem that cannot be considered uniformly. The problem changes drastically with the nature of the input graph and the expected output. In fact, a general algorithm might not enforce a planar drawing for a planar graph or a hierarchical drawing for a rooted tree. As a consequence, the graph drawing literature contains a very high number of algorithms that differ in the approach chosen or for the specific case they aim to deal with.

Planar Drawing Algorithms

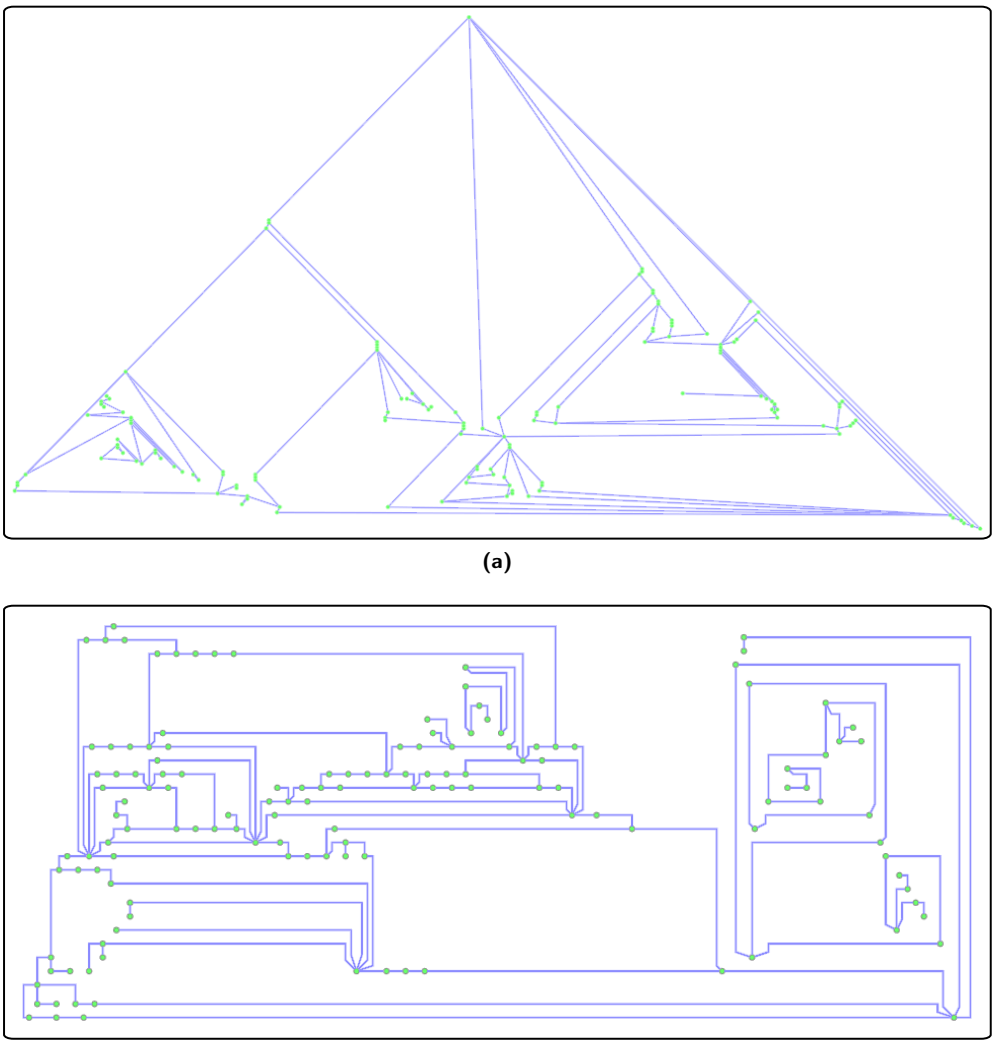
The class of *planar drawing algorithms* collects the algorithms that draw planar graphs without edge crossings. Due to the vast field of applications of planar graphs (among others electronic circuit design, vehicle routing or service planning), there exist many subcategories of this class [75]. Here, we will only discuss straight-line and polyline drawings, that are the most general class of drawings and the most significant from an information visualisation point of view.

The entire field of planar graph drawing, however, seems to be of little interest in information visualisation:

> In information visualisation applications, it only makes sense to check for planarity when dealing with a small and sparse graph [...]. In general, we can safely say that planarity is not a central issue in information visualisation.
>
> Herman, Melançon and Marshall [59]

The reason for this lack of interest is probably related to the high specificity of the problem. Huge graphs are likely not to be planar, and small and sparse graphs might be understandable even when there are a few crossings in their drawings. As a result, most algorithms neglect comprehensibility aspects and focus more on theoretical issues, such as complexity or space used, or are developed to solve application specific problems. There are however a few exceptions, as shown by some of the algorithms below.

Straight-Line Planar Drawings. The proof of the existence of a planar straight-line drawing for every planar graph, that can lead to efficient algorithms to draw them, dates back to before the 1950s [35, 98, 108]. In the following years, new algorithms were developed to reduce the complexity and the area necessary to draw the graph. At the current state, two approaches [18, 87] are mainly used to obtain straight-line



(a)

(b)

Figure 3.13: Output comparison of the presented planar drawing algorithms. **(a)** Drawing generated by `FPP`. **(b)** The same graph drawn by `Mixed-Model`.

planar drawings. The approaches are distinct but actually related to each other [74], and both obtain compact graph drawings in linear time.

In this thesis we will use the approach of De Fraysseix, Pach and Pollack [18], and in particular its linear implementation [13], whenever we need to obtain a first straight-line, planar drawing of a graph. We will refer to the algorithm as `FPP`.

The algorithm focuses more on theoretical results rather than on graph aesthetics. The drawings produced are affected by a very low angular resolution and very inhomogeneous edge length, as shown in figure 3.13a. In the general case, it is possible to produce straight-line drawings that present much higher aesthetics than those generated with `FPP`. However, when dealing with very complex graphs, the margin of improvement is drastically reduced and the drawings produced are close to the best obtainable.

Polyline Planar Drawings. To overcome the low aesthetics of complex straight-line drawings, there is no choice but moving to a different drawing style. Gutwenger and Mutzel [53] use the polyline drawing style to achieve high aesthetics in planar graph drawings. The algorithm proposed `Mixed-Model` produces drawings characterised by very high angular resolution, a very limited number of bends and of space used, and runs in linear time with respect to the graph nodes.

As shown in figure 3.13b, the typical drawings generated by this algorithms are much clearer than those produced by `FPP`. This is particularly evident on difficult input instances, where a higher degree of freedom on the edge shape is essential to produce readable graphs.

Force-Directed Algorithms

The class of *force-directed* algorithms collects the algorithms that generate a graph drawing by simulating the evolution of a physical system. According to Di Battista et al. [25], the first algorithm that belongs to this family has been developed by Tutte [103] in 1963. The algorithm computes a planar layout with only convex faces for any planar, 3-connected graph. The result is obtained by selecting an opportune position for a first set of nodes, and by driving the positioning of the remaining nodes by forcing them to the barycentre of their neighbours. However, as the author does not provide a physical interpretation of the layout process, other methods are generally considered the progenitors of the force-directed algorithm class.

Classical Force-Directed Algorithms. Eades [30] developed the first algorithm that interprets the graph layout as the evolution of a physical system. In this method, the graph edges are thought of as springs, and the graph nodes as rings to which the springs are connected. Starting from a non-optimal initial state, the system will evolve under the effect of tensed or compressed springs to a stable position, where each spring is as close as possible to its optimal length.

In the work of Kamada and Kawai [64], the spring model is extended to consider all the other nodes of the graph, rather than only a node's neighbours. The force system aims to place any pair u, v of nodes at at distance $l \cdot \text{dist}(u, v)$ from every other node, where l is the optimal distance between neighbour nodes. In other words, the algorithm links the theoretical graph distance to the Euclidean distance in the graph layout. The authors also explain how to compute the optimal displacement of a node, when only one node is moved at each iteration.

Fruchterman and Reingold [45] also improved the work of Eades [30], proposing a more sophisticated force system. The algorithm, was also inspired by the use of simulated annealing in graph drawing [17], and it models the graph nodes as charges that repel each other, and the edges as zero-length springs that attract the linked charges. In this case, there exist two kinds of forces rather than one: a first, that is always repulsive (see figure 3.14a), and a second, that is always attractive (see figure 3.14b). The stable state is therefore reached once these forces are balanced for all the nodes in the graph (see figure 3.14c).

Newer Force-Directed Algorithms. These methods inspired a long series of algorithms based on the same principles. In newer algorithms, the structure and the force system of the original works are generally much preserved, and the contribution is typically oriented towards overcoming or mitigating the weaknesses of the force-directed approach. In `Gem` [42], the focus is on improving the efficiency of the

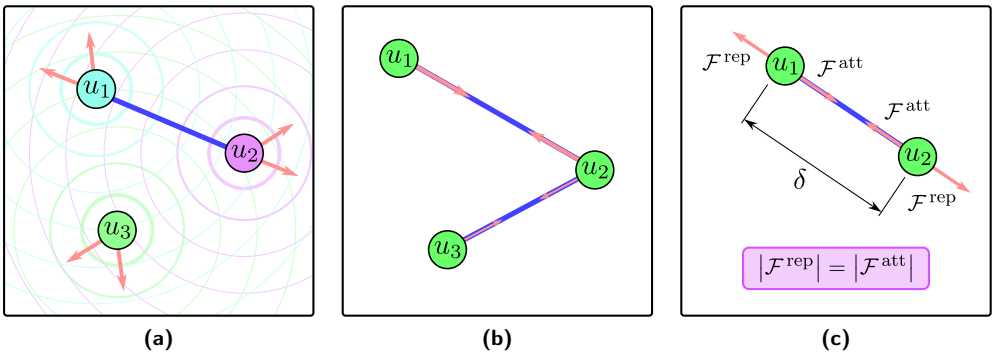


Figure 3.14: Typical forces considered in a force-directed algorithm. Thicker field lines and vectors indicate higher force magnitudes. **(a)** The repulsive force between nodes. **(b)** The attraction force between adjacent nodes. **(c)** Attractive and repulsive forces balance at the optimal edge length δ.

algorithm by spending more computational effort on nodes that present higher stress and higher chances of reducing it.

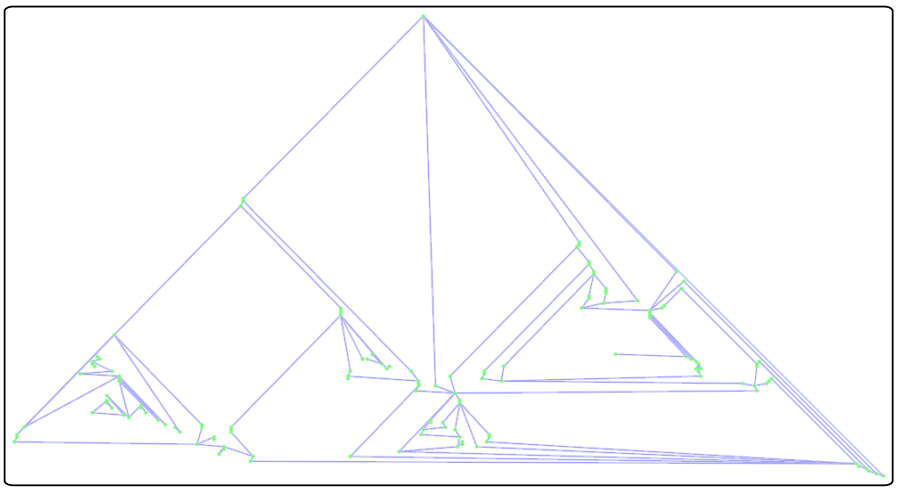
Cohen [14] and Hadany and Harel [55] introduced the concept of multilevel layout, that consists of computing the layout by stages, starting from very few nodes and gradually increasing the number of nodes taken into consideration. This approach is used in many force-directed algorithms that aim to layout very large graphs, such as `GRIP` [46, 47], `LGL` [1], and that of Walshaw [109] and of Harel and Koren [58].

A different approach to reduce the computational time and to scale to larger graphs has been proposed by Fruchterman and Reingold [45] and by the authors of `FADE` [81]. The algorithms decompose the plane in cells, generating groups of geometrically close nodes. The nodes inside each cell are then represented with a pseudo-node, a single entity that approximates the position and the effect of the cell's nodes when computing the forces. With this system, the number of forces to be computed is greatly reduced without changing significantly the resulting forces on the nodes. `FM`^{3} [54] further improves these methods by combining it with a multilevel approach.

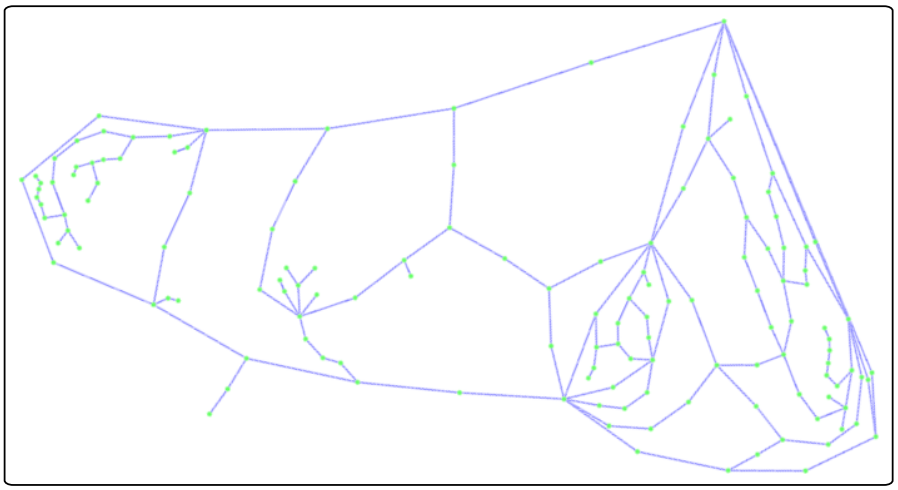
Constrained Force-Directed Algorithms. Standard force-directed algorithms use forces to drive the positioning of the nodes to a desired configuration. However, the force system itself cannot enforce the presence of any particular structure or property in the final layout. For this reason, force-directed algorithms that aim at particular configurations of the graph layout implement additional constrains on the node movement.

`PrEd` [6] extends the method of Fruchterman and Reingold [45] to grant that no new edge crossings will be created and no existing edge crossings will be undone at any stage of the layout computation. This means that the algorithm can be used to optimise a planar layout while preserving its planarity and its embedding (see figure 3.15), or to improve a graph that has a meaningful initial set of edge crossings. To achieve this result, `PrEd` adds a phase where the maximal movement of each node is computed, and adds a repulsive force between nodes and edges. Since `PrEd` is used in our method for the generation of Euler diagrams, the algorithm will be explained in greater detail in section 6.1.

Dwyer et al. [26–29] developed a series of algorithms that enforce several different



(a)



(b)

Figure 3.15: Example of layout improvement with `PrEd`. **(a)** A sub-optimal planar drawing (see figure 3.13a). **(b)** The drawing improved by `PrEd`.

characteristics on the final layout of a graph. In these algorithms, a method that improves the computation of the optimal position proposed by Kamada and Kawai [64], called stress majorisation [49], is used to transform the computation of a graph layout into a problem of mathematical programming. It is then possible to provide additional constrains to the problem to exclude drawings that do not present the desired characteristics, such as a particular orientation of directed edges, the displacement of nodes according to a hierarchy or their positioning into particular shapes.

Advantages and Disadvantages. The algorithms that implement a force-directed approach typically present the same common advantages and disadvantages:

- *Simple and intuitive.* Force-directed algorithms are generally very simple to understand and implement.

- *High aesthetics for small graphs.* For graphs up to some hundred nodes, the layout produced generally has a good distribution of the nodes, uniform edge length and high angular resolution. Also, symmetries in the graph structure are shown in the final layout.
- *Preserving the mental map.* Force-directed algorithms can be used to improve an initial, sub-optimal, layout of a graph. When the initial configuration is sufficiently stable, the resulting layout will retain the general graph structure and the reciprocal positions of the nodes, partially preserving the insight acquired by the user in the previous drawing [32, 73].
- *Lower aesthetics for bigger graphs.* The aesthetics of the graph typically decreases with the increase of the graph size, since the computation is more likely to reach configurations of local optima. When this happens, the evolution of the system converges to a final layout that may be much worse than the best obtainable.
- *High computational cost.* Early force-directed algorithms were typically characterised by a high algorithm complexity, which limited their applicability for large graphs. Newer algorithm overcome this limitation.

3.3 Euler Diagram Theory

In this section, we complete the introduction to Euler diagrams started in the previous chapter by approaching the topic from a more formal point of view. We start by presenting the basic terminology, such as clusters, zones and regions, along with their notation.

Towards the end of the section, the discussion shifts to properties of Euler diagrams, to the different conventions used and to the drawability issues that they present.

3.3.1 Clusters and Zones

Due to the aim of this thesis, we assume that behind every Euler diagram there exist a set of all elements V and a set system S to be represented. Each set $s \in S$ of the set system collects a certain number of elements of V, so that different sets s might share some of their elements. For instance, in the example in figure 2.20 on page 16, V is the set of all buttons and S is the set system to be depicted that is formed by the overlapping sets: green buttons, blue buttons, round buttons, etc.

In section 3.1.11 on page 40, we already described a situation in which there existed mathematical entities V and S, having the same properties described above: the case of overlapping clustered graphs. In clustered graphs, V is the set of nodes of the graph and S a collection of sets that groups some of them. Since graphs extend sets, and since we can always consider the case $E = \emptyset$ if we are not interested in relations between elements, we will assume that the set of all elements V is actually the set of the nodes of a graph.

This assumption allows us to unify the formalisation of Euler diagrams into a single case, making it independent from the set system being an overlapping collection of elements or a fuzzy graph clustering. Therefore, even when the elements in an Euler diagram are not part of an explicitly defined graph, the set system will be called a clustering, and the sets to be represented will be called clusters. To refer to different clusters $s \in S$, we will use upper case letters in sans-serif font, such as A, B, C.

In section 2.2.1 on page 17, we saw how the clusters generate a number of different intersections. For instance, in the case of three clusters $S = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, we used the minterm $\mathsf{ab\bar{c}}$ to indicate the intersection $\mathsf{A} \cup \mathsf{B} \cup \bar{\mathsf{C}}$. We now call those intersections zones and we indicate them with a sequence of lower case letters in sans-serif font, that omits all the complemented clusters. For example, abd and de are the zones included respectively in the clusters $\{\mathsf{A}, \mathsf{B}, \mathsf{D}\}$ and $\{\mathsf{D}, \mathsf{E}\}$, and external to all the other clusters of S.

Since a zone can be either included or excluded in each cluster, there are a total of $2^{|S|}$ possible combinations. In fact each zone corresponds to an element p of the power set of S, and in particular to that zone that is included in all the clusters in p, and excluded by all the others. The zones identified in this way are all the zones that can possibly exist in a diagram with such S, and are collected in the set Z^*. However, since some zones cannot exist due to some particular cluster configurations (for instance, the zone $\mathsf{\bar{a}b}$ cannot exist if $\mathsf{A} \supset \mathsf{B}$), we usually consider the set of the non-null zones Z, that is the set of the zones actually expressed in the diagram.

We also define the concept of associated zones and clusters, that allow us to switch from clusters to their related zones, and vice versa. The associated zones $\mathcal{A}(s)$ of a cluster s are all the non-empty zones contained in the cluster, or in other words, all the non-empty zones that contain the cluster letter in their label. For instance, $\mathcal{A}(\mathsf{A}) = \{\mathsf{a}, \mathsf{ab}, \mathsf{ac}\}$. The associated clusters $\mathcal{A}(z)$ of a zone z are the clusters that contain the zone, or in other words, the cluster whose letter is in the zone's label. For instance $\mathcal{A}(\mathsf{abd}) = \{\mathsf{A}, \mathsf{B}, \mathsf{D}\}$.

Finally, we introduce the concept of wildcard characters to easily refer to the set of zones whose labels share the same group of letters. For instance, with $\mathsf{ab}*$ we indicate the set of the zones that contain the letters a and b in their label. Clearly, we have that $\mathsf{a}* = \mathcal{A}(\mathsf{A})$.

Clusters. For uniformity of notation, we always assume that an Euler diagram is associated with an overlapping clustering S of a clustered graph $G = (V, E, S)$. Therefore, we define a *cluster* $s_i \in S$ as a set to be represented in an Euler diagram, and we indicate it with an upper case letter in sans-serif font, such as A, B, C. As a consequence, the set system of the diagram corresponds to the *clustering* S.

Zones. Let $Q = \mathcal{P}(S)$, where $\mathcal{P}(S)$ denotes the power set of S. A *zone* z is a set defined by a certain $q \in Q$ as

$$z = \mathcal{Z}(q) = \bigcap_{s_i \in q} s_i \cap \bigcap_{s_i \in S,\ s_i \notin q} \bar{s_i} = \bigcap_{s_i \in q} s_i \setminus \bigcup_{s_i \in S,\ s_i \notin q} s_i$$

A zone is indicated with a sequence of lower case letters in sans-serif font, such as ab and c, corresponding to the labels of the clusters in q. With Z^* we indicate the set of all possible zones of a diagram, $Z^* = \{\mathcal{Z}(q) : q \in Q\}$, and with Z the set of non-empty zones, $Z = \{z \in Z^* : z \neq \emptyset\}$.

Associated Zones and Clusters. The *associated clusters* of a zone z are the clusters that fully contain z:

$$\mathcal{A}(z) = \{s \in S : z \subseteq s\}$$

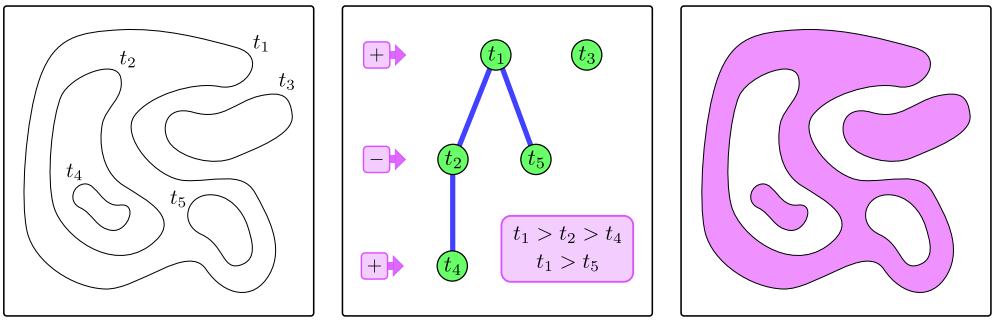


Figure 3.16: Identification of a cluster region. **(a)** A set of cluster curves. **(b)** The inclusion hierarchy depicted as a forest. The levels of the forest are marked alternating as positive and negative regions. **(c)** The coloured surface is the resulting cluster region.

When $z = \mathcal{Z}(q)$, $\mathcal{A}(z)$ correspond to q. The *associated zones* of a cluster s are the non-empty zones that are fully contained in s:

$$\mathcal{A}(s) = \{z \in Z : z \subseteq s\}$$

Wildcards. When using the zone labels, we use Greek letters, such as α and β, to indicate a sequence of lower case Latin letters. For instance, $\alpha = \mathsf{abd}$ or $\alpha = \mathsf{ce}$. With the notation $\alpha*$ we indicate the set of zones

$$\alpha* = \mathcal{A}(s_1) \cap \mathcal{A}(s_2) \ldots \qquad \text{where } \alpha = s_1 s_2 \ldots$$

3.3.2 Euler Diagrams and Regions

In Euler diagrams, the portion of the plane assigned to each cluster is called a cluster region and it is typically identified as the area enclosed by a simple closed curve, such as a circle or an ellipse. The area assigned to each zone z is then automatically identified as the portion of the plane included in the cluster regions of the clusters in $\mathcal{A}(z)$, and excluded from those of all the remaining clusters.

Here, we present a more general definition of the cluster regions. Each cluster is associated with one or more simple, closed curves (Jordan curves), that do not intersect between them, but that can intersect with those of other clusters. For cluster s, we indicate the set of these cluster curves with T_s. When a curve contains other curves of the same cluster, these are intended alternating as positive or negative regions according to the order of inclusion.

For instance, if we have only two cluster curves t_1 and t_2, and t_1 includes t_2, then t_2 acts as a negative region (a hole) and its internal area is removed from the cluster region. If a third cluster curve t_3 is contained in t_2, that is a hole, then the internal area of t_3 acts as a positive region and it is added to the resulting cluster region, that is now composed of the portion of the plane between t_1 and t_2 and by the internal region of t_3 (see figure 3.16).

Euler Diagrams. An *Euler diagram* D is a pair $(T, \mathcal{T})$, where T is a set of Jordan curves, $\mathcal{T} : T \to S$ is a function that associates a curve with a cluster and where all the curves associated with the same cluster, called *cluster curves*, $T_s = \{t \in T : \mathcal{T}(t) = s\}$, do not intersect between them:

$$t_1, t_2 \in T_s,\ t_1 \neq t_2 \ \Rightarrow\ t_1 \cap t_2 = \emptyset$$

Relations Between the Cluster Curves. A Jordan curve t divides the plane into two regions: one enclosed by the curve, $\text{int}(t)$, and one external to the curve, $\text{ext}(t)$ (*Jordan Curve Theorem*). Since two different cluster curves $t_1, t_2 \in T_s$ do not intersect, t_2 lies either completely in $\text{int}(t_1)$ or completely in $\text{ext}(t_1)$:

$$t_2 \subset \text{int}(t_1) \quad \text{or} \quad t_2 \subset \text{ext}(t_1)$$

The inclusion relationship is a partial ordering on the cluster curves in T_s, where $t_1 > t_2$ if $t_2 \subset \text{int}(t_1)$. We can therefore organise them into a hierarchy, that assumes the shape of a forest. The roots of the forest are the cluster curves that are not contained in the internal region of any other curve. The children of each curve t are the curves directly contained in the internal region of t, and the leaves are the curves with no other curves in their internal region.

Cluster Regions. A *cluster region* $s^{\mathcal{R}}$ is a portion of the plane associated with a cluster s. In Euler diagrams, it is identified by the cluster curves T_s. Let $G = (V, E)$ be the forest that represents their inclusion hierarchy, and $i_{\max}$ be its maximum level. The cluster region corresponds to $s^{\mathcal{R}}_{i_{\max}}$, with $s^{\mathcal{R}}_i$ inductively defined as

$$s^{\mathcal{R}}_i = \begin{cases} \bigcup\limits_{t \in V^0} \text{int}(t) & \text{if } i = 0 \\ s^{\mathcal{R}}_{i-1} \setminus \bigcup\limits_{t \in V^i} \text{int}(t) & \text{if } i > 0,\ i \equiv 1 \mod 2 \\ s^{\mathcal{R}}_{i-1} \cup \bigcup\limits_{t \in V^i} \text{int}(t) & \text{if } i > 0,\ i \equiv 0 \mod 2 \end{cases}$$

Zone Regions. A *zone region* $z^{\mathcal{R}}$ is a portion of the plane associated with a zone z. It is identified as

$$z^{\mathcal{R}} = \bigcap_{s_i \in \mathcal{A}(z)} s^{\mathcal{R}}_i \cap \bigcap_{s_i \in S,\ s_i \notin \mathcal{A}(z)} \overline{s^{\mathcal{R}}_i} = \bigcap_{s_i \in \mathcal{A}(z)} s^{\mathcal{R}}_i \setminus \bigcup_{s_i \in S,\ s_i \notin \mathcal{A}(z)} s^{\mathcal{R}}_i$$

Other Definitions of Euler Diagrams in the Literature. The above cluster region definition allow us to obtain a much broader range of configurations, introducing concepts already used by the creators of these diagrams, such as holes or disconnected cluster regions (see figure 2.27b on page 25, that depicts a Venn diagram with a hole). This definition is also coherent with most definitions already proposed in the literature.

Stapleton et al. [97] propose the same notation, but use the concept of *winding numbers* to determine the actual cluster region from the cluster curves. Since we con-

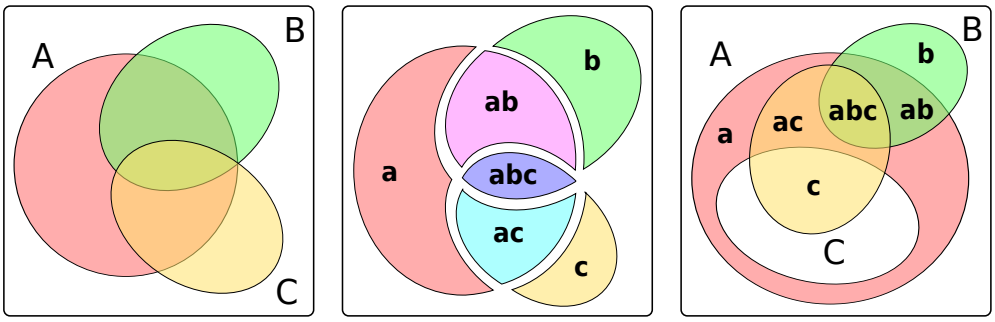


Figure 3.17: Identification of zone regions. **(a)** A diagram with three clusters. **(b)** The zones of the previous diagram have been labelled and split. **(c)** Labelling of the zones of another diagram, containing a hole.

sider only simple and not intersecting cluster curves, the two definitions are equivalent. Verroust and Viaud [107] propose a notation that marks explicitly each curve as positive or negative, and the cluster region is identified as the intersection of the internal area of positive curves and the external area of negative curves. This definition is less general than the one proposed in this thesis, since it cannot represent a configuration with a positive region inside a hole (see figure 3.16).

The definition proposed in this thesis is also compatible with those that admit a single curve for each cluster, such as the definitions of Chow [12] and Flower, Fish and Howse [39]. In this case, it is sufficient to restrict the cardinality of T_s to one for each cluster $s \in S$.

Although our definition could be further generalised by considering non-simple curves, they are not strictly necessary to enlarge the number of clusterings that can be depicted and may lead to ambiguities that can be difficult to untangle [37, 38]. Therefore, we chose to consider only simple curves in our definition.

The definition of zone regions is similar in all papers, although their identification is influenced by the actual formulation of the cluster regions (see figure 3.17).

3.3.3 Properties of Euler Diagrams

An Euler diagram can be characterised by patterns that typically decrease its readability. Stapleton et al. [97] identified several patterns, such as multiple crossing points, concurrent curves, disconnected zones and clusters (see figure 3.18). Benoy and Rodgers [5] evaluated the effect on the diagram comprehension of different criteria, finding evidence that diagrams with more regular shapes, evenly spaced curves and zones with adequate area can be more easily understood.

The presence or absence of these patterns has been defined as a diagram property by Stapleton et al. [97]. A diagram can therefore be characterised by the property of having only one curve per cluster, or not having points where more than two curves intersect. These properties are not only mere aesthetics aspects: even the definition of an Euler diagram, given by different authors, differs depending on these properties. Moreover, several generation methods only work with diagrams that have a specific set of properties, and might not be able to draw clusterings that do not admit this kind of diagram.

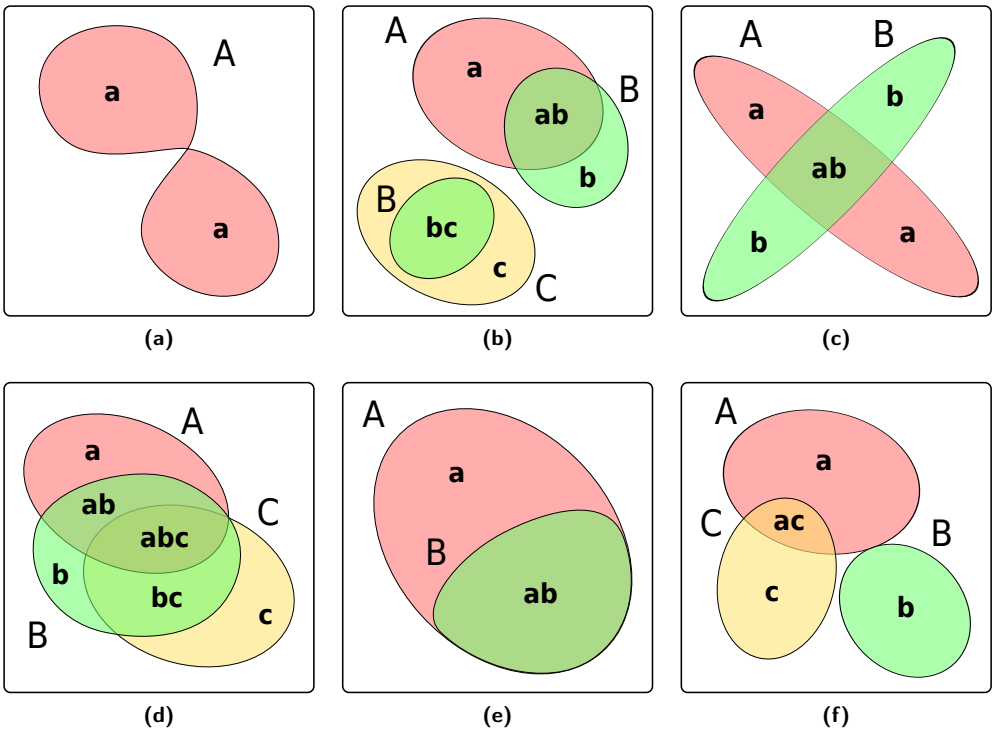


Figure 3.18: Examples of patterns considered by the Euler diagram properties. **(a)** A cluster bounded by a non-simple curve. **(b)** A cluster region, $\mathsf{B}^{\mathcal{R}}$, is not connected. All zone regions are instead connected. **(c)** The zone regions $\mathsf{a}^{\mathcal{R}}$ and $\mathsf{b}^{\mathcal{R}}$ are disconnected. The cluster regions are instead connected. **(d)** A multiple crossing point in correspondence with the intersection of the three cluster curves. **(e)** Concurrency between the cluster curves. **(f)** Brushing point between the curves of cluster A and B.

Properties of Euler Diagrams. We define the following *properties* that might be associated with an Euler diagram:

- *Simple curves*: all the curves of the Euler diagram are simple (assured by the above definition of an Euler diagram):

$$\forall t \in T,\ t \text{ is a simple curve}$$

- *Only expressed zones*: the zones present in the diagram are all and only the non-null zones expressed by the clustering:

$$z^{\mathcal{R}} \neq \emptyset \ \Leftrightarrow\ z \in Z$$

- *Single cluster curve*: all the clusters in the diagram have exactly one associated curve:

$$\forall s \in S,\ |T_s| = 1$$

- *No disconnected clusters*: all the clusters in the diagram have a connected cluster region:
$$\forall s \in S,\ s^{\mathcal{R}} \text{ is a connected region}$$
- *No disconnected zones*: all the zones in the diagram have a connected zone region:
$$\forall z \in Z,\ z^{\mathcal{R}} \text{ is a connected region}$$
- *No multiple crossings*: there are no points where more than two curves cross, that means no points belong to three or more curves:
$$t_1, t_2, t_3 \in T,\ t_1, t_2, t_3 \text{ different } \Rightarrow t_1 \cap t_2 \cap t_3 = \emptyset$$
- *No concurrent curves*: there are no curves that share a portion of their lines, that means no curves share an interval of points:
$$t_1, t_2 \in T,\ t_1 \neq t_2 \ \Rightarrow\ \nexists\, p_1, p_2 \in t_1, p_1 \neq p_2 : [p_1, p_2] \in t_2$$
- *No brushing points*: there are no tangent points between the curves, that means no curve t_1 share a point or an interval with another curve t_2 without moving from $\text{int}(t_2)$ to $\text{ext}(t_2)$, or vice versa:
$$t_1, t_2 \in T,\ t_1 \neq t_2,\ [p_1, p_2] \in t_1 \cap t_2 \ \Rightarrow$$
$$\nexists \varepsilon \in \mathbb{R},\ \varepsilon > 0 : \quad (p_1 - \varepsilon, p_1) \in \text{int}(t_2) \ \oplus\ (p_2, p_2 + \varepsilon) \in \text{ext}(t_2)$$
- *Convex clusters*: all the clusters in the diagram have a convex cluster region:
$$\forall s \in S,\ s^{\mathcal{R}} \text{ is a convex region}$$
- *Given cluster shape*: all the clusters in the diagram have a cluster region of a given shape, such as a circle or an ellipse:
$$\forall s \in S,\ s^{\mathcal{R}} \text{ is of the given shape}$$

3.3.4 Validity of an Euler Diagram

The properties of Euler diagrams are used to discriminate between correct and incorrect representations of a given clustering. Only when an Euler diagram respects the chosen set of properties, it is possible to say that the diagram is a valid representation of that set system. Two of these properties are considered necessary by the vast majority of the authors, and will therefore be taken as granted: simple curves, since the inclusion relationship is much clearer under this condition, and depicting only expressed zones, to avoid ambiguity of the diagram (see section 2.2.2).

Euler himself [33] did not introduce the diagrams in a formal way, characterising which properties a diagram should have. His examples, however, never present multiple cluster curves, disconnected zones or clusters. Therefore, most authors [12, 84, 90] depict a valid *standard Euler diagram* using these properties.

Type	Definition	Properties
Restricted	Well-formed Euler diagrams	• Simple curves • Only expressed zones • No disconnected zones • No disconnected clusters • Single cluster curve • No multiple crossings • No concurrent curves • No brushing points
Standard	Standard Euler diagrams	• Simple curves • Only expressed zones • No disconnected zones • No disconnected clusters • Single cluster curve
Generalised	Extended Euler diagrams	• Simple curves • Only expressed zones • No disconnected zones • No disconnected clusters
	Relaxed Euler diagrams	• Simple curves • Only expressed zones • No disconnected zones

Table 3.1: Properties associated with different definitions of Euler diagrams. We marked the properties shared by all definitions with a yellow bullet, the properties shared by standard and restricted Euler diagrams with an orange bullet, and the additional properties enforced by well-formed diagrams with a red bullet.

Flower, Fish and Howse [39] worked on a further restricted class of diagrams, called *well-formed Euler diagrams*, that aim for the maximal clarity of the representation. These diagrams must avoid multiple crossings, concurrent curves and brushing points, in addition to the properties of standard Euler diagrams.

Other authors [12, 84, 91, 107] proposed representations that are less restrictive than standard Euler diagrams. Typically, these diagrams are characterised by non-disconnected zones, but accept multiple curves per cluster (holes and disconnected clusters). We will call these diagrams *generalised Euler diagrams*.

Table 3.1 summarises the properties associated with the different kinds of diagrams proposed in the literature.

Validity. An Euler diagram is a *valid* representation of the clustering S if each zone $z \in Z$ has a corresponding region $z^{\mathcal{R}}$ and if the diagram respects a given set of properties.

In this thesis, we use the set of properties of relaxed Euler diagrams, that comprises the properties: simple curves, only expressed zones, and no disconnected zones.

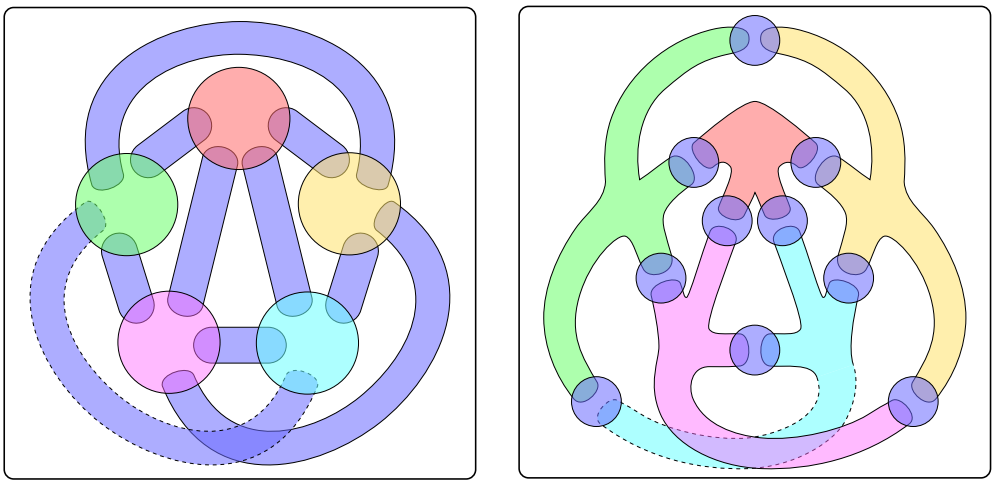


Figure 3.19: Example of an undrawable Euler diagram, according to the standard set of properties, derived from the graph K_5. All cluster regions relative to the graph edges have been filled in dark blue, but are all to be considered distinct. **(a)** A first attempt to draw the graph fails as we cannot avoid overlaps between the dashed cluster region and other edge cluster regions. **(b)** Shaping the node cluster regions, an advantage that we have compared to the case of graphs, does not solve the problem. We cannot avoid overlaps between the dashed portion of the region and other unrelated ones.

3.3.5 Drawability of an Euler diagram

The choice of a set of properties deeply influences the existence of a valid Euler diagram for a given clustering, as respecting all the selected properties at the same time might be impossible. Thus, the drawability of a clustering is related to the existence of a valid Euler diagram for that clustering.

Well-formed Euler diagrams are the most limited class. These diagrams have a high number of undrawable instances, even between very common clusterings. For instance, the case of a clustering containing two identical clusters, $S = \{s_1, s_2\}$, $s_1 = s_2$, is not drawable, as it is both needed to represent all and only the expressed zones and to have not concurrent cluster curves. Figure 3.18d shows other example of diagram that cannot be drawn under the well-formedness conditions.

Standard Euler diagrams can depict a much larger class of inputs, but it is still possible to generate clusterings that do not have a valid representation. Let us consider a diagram built over a non planar graph, so that each node and edge of the graph is associated with a different cluster, and two clusters overlap only if one is associated with an edge and the other with its source or target. To draw the diagram we would need either to have a disconnected cluster or to extend a cluster region over a non related one. Both actions are forbidden, since disconnected clusters are not allowed and we can only depict expressed zones (see figure 3.19).

Extended Euler diagrams, proposed by Verroust and Viaud [107], are characterised by non-disconnected zones, non-disconnected clusters and multiple cluster curves (holes). In their article they propose a generation method and prove that clusterings up to eight sets are all drawable, but they show an instance of undrawable clustering composed of nine sets.

Rodgers, Zhang and Fish [84] and Simonetto, Auber and Archambault [91] discussed how disconnected cluster regions are necessary to remedy the planarity issues described above. Therefore, a class of generalised diagrams that aim to draw all input instances must allow them. Trivially, disconnected clusters are also a sufficient condition, since we would be able to place each zone in its own isolated region and still obtain a valid representation.

Drawability. A clustering is *drawable* if it has a valid Euler diagram. Every clustering is drawable according to the conditions specified in this thesis for a valid Euler diagram.

Chapter 4

Algorithms for the Generation of Euler Diagrams

This chapter presents the methods and the algorithms proposed for the automatic generation of Euler diagrams.

First, we describe and review the generation algorithms currently proposed in the literature. For each method, we discuss the kind of diagram that the authors aim to generate, the benefits and drawbacks of the choices that have been taken, and the technical implementation of the approach.

Then, we introduce the generation method we have developed. The method will be presented very briefly, since in this chapter we only aim to compare the approach with that of other methods in the literature. A detailed explanation of the method will be presented in the following chapter.

4.1 Related Work

During the last decade, several algorithms for the automatic generation of Euler diagrams have been proposed. These algorithms present significant differences on the technical aspects and the generated output, but all share the same general approach to the Euler diagrams generation: first, generate and draw a plane graph that represents the backbone of the final diagram, then, use it to draw the cluster curves.

The generation of Euler diagrams is therefore seen as a graph drawing problem that involves, among others, graph planarity, layout aesthetics and optimisation, and routing of curved edges.

The algorithms presented in this section are grouped on the basis of the diagram they generate, according to the definitions reported in table 3.1. Then, we describe a number of algorithms having special characteristics, or that focus on very specific tasks in the diagram generation. The section is finally closed by a few InfoVis methods that do not produce Euler diagrams as defined in this thesis, but a visualisation with clear analogies to this kind of diagrams.

4.1.1 Well-Formed Euler Diagrams

The first method for the automatic generation of Euler diagrams was sketched by Flower and Howse in a paper [40] that dates back to the year 2002, and was fully

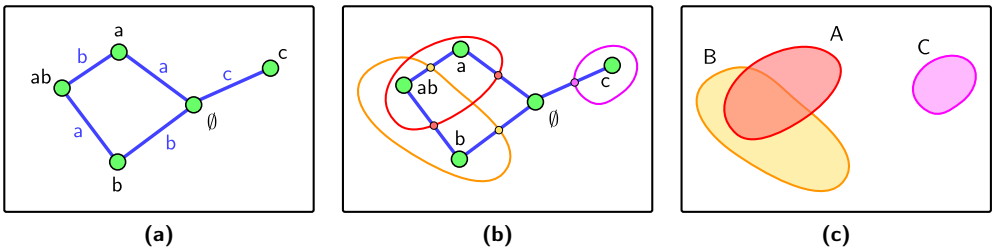


Figure 4.1: Generation of well-formed Euler diagrams. **(a)** Generation and drawing of the super-dual graph. **(b)** Routing of the cluster curves. The algorithm detects the points were the curves must pass. **(c)** The resulting diagram.

formalised some years later [39]. The method uses the procedure mentioned above, and is able to generate well-formed Euler diagrams only.

The backbone graph is called *super-dual* G_d, and is composed as follows. The graph nodes are inserted according to the zones of the diagram to be built, since there must be a one-to-one correspondence between the nodes of G_d and the elements of Z. The edges are inserted between nodes whose zone labels differ from each other for the presence/absence of only one letter. For instance, the node ab would be directly linked to the node abd, since only the letter d is present in one label and absent in the other. However, the same node would not be directly connected with ace, as the labels differ for multiple letters (see figure 4.1a).

Since a valid graph G_d must be plane, it might be necessary to remove edges from its initial configuration until the graph becomes planar. At the same time, it is necessary to observe a second condition, called connectivity condition: for each cluster $s \in S$, both the subgraph induced by nodes associated with the zones $\mathcal{A}(s)$ (the set of zones whose label contains the letter of s) and the subgraph induced by the remaining nodes, $\bar{\mathcal{A}}(s)$, must be connected (see figure 4.2).

Once the edges of G_d have been decided, the graph is embedded on the plane. Here some additional conditions, called face conditions, are verified to ensure that no unwanted zones are created when routing the cluster curves. In the case that some of the conditions fail, it is necessary to produce an alternative planar drawing of G_d and reiterate the control of the face conditions, until they are all respected.

Starting from a correctly drawn super-dual graph, the cluster curves are depicted so that each cluster $s \in S$ has a single curve t that includes all the nodes associated with the zones $\mathcal{A}(s)$ and none of the remaining nodes. To correctly route the curves, the algorithm detects some points that the curve must cross and, in the case of finite faces, traces the curves trough the middle of the face (see figure 4.1b). By eliminating the graph and considering only the cluster curves, we obtain the final diagram (see figure 4.1c).

Remarks. The authors have the merit of presenting a procedure that has been largely used in the automatic generation of Euler diagrams. All following algorithms, in fact, compute the cluster curves only once the general shape of the diagram is defined by drawing a planar graph. Furthermore, the authors presented a large number of theoretical results and definitions that have also been adopted by other authors.

Unfortunately, since the paper is mostly theoretical, there is not a lot of detailed information on the choice of the edges to be removed and on the routing of the cluster

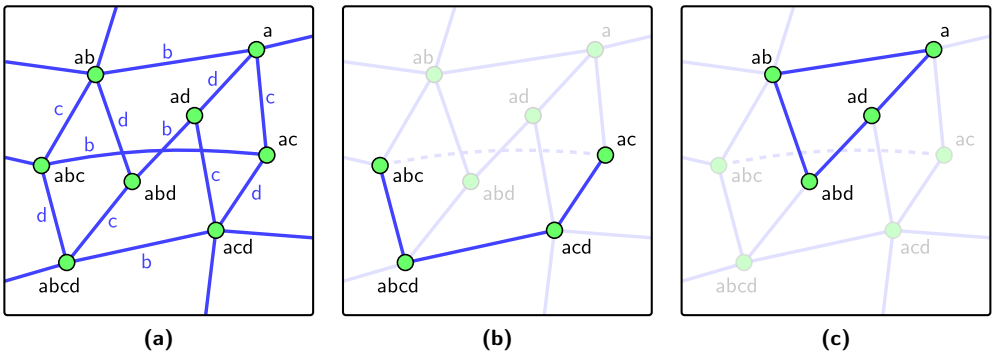


Figure 4.2: Removing edges of a super-dual graph to meet the planarity and connectivity conditions. The figures show only a portion of the super-dual and simulate on it the removal procedure and the relative checks. Clearly, these controls are done on the whole graph and at a topological level, therefore before drawing the graph. **(a)** A portion of the initial configuration of the graph G_d. **(b)** After deciding to remove the edge [abc, ac], we check if the subgraph induced by the nodes associated with the zones $\mathcal{A}(s)$, for each $s \in S$, is connected. Here we check the nodes for $s = \mathsf{C}$, that are connected. **(c)** Also the subgraph induced by the nodes of $\bar{\mathcal{A}}(s)$, for each $s \in S$, must be connected. Again, we check $s = \mathsf{C}$ and the subgraph is connected.

curves. Since the specific implementation of these tasks deeply influences the aesthetics of the final diagram, it is not possible to discuss the quality of the resulting diagrams.

Furthermore, the method is designed to produce only well-formed Euler diagrams. Although these diagrams present higher readability than general ones, a large number of very common configurations are undrawable under these conditions (see section 3.3.5).

Routing the Cluster Curves in Well-Formed Euler Diagrams

Rodgers et al. [85] presented an algorithm for routing the cluster curves in a diagram generated with the previous method.

In order to obtain reference points for drawing the curves, the algorithm triangulates the graph, meaning that additional edges are inserted to obtain only triangular faces. The unbounded face is clamped by a frame and is also triangulated. Unlike the standard edges of the super-dual, the newly inserted edges might connect nodes that differ for more than one letter. For example, in order to triangulate the graph, we might insert an edge between a and abc, that differ for two letters (see figures 4.3a and 4.3b).

Since the edges are crossed only by the cluster curves that differ in the label of their extremities, the newly inserted edges might be crossed by zero or multiple curves. When more than one curve cross an edge, a greedy policy assigns the ordering of the edge-curve crossings so that the final diagram does not contain undesired overlaps and all the zones are correctly represented (see figures 4.3c and 4.3d).

The authors provided other methods to further improve the quality of the resulting diagram, such as inserting additional elements for a finer triangulation, optimising the layout of the triangulated graph with a force-directed algorithm and smoothing the cluster curves by shifting the crossing points along the graph edges.

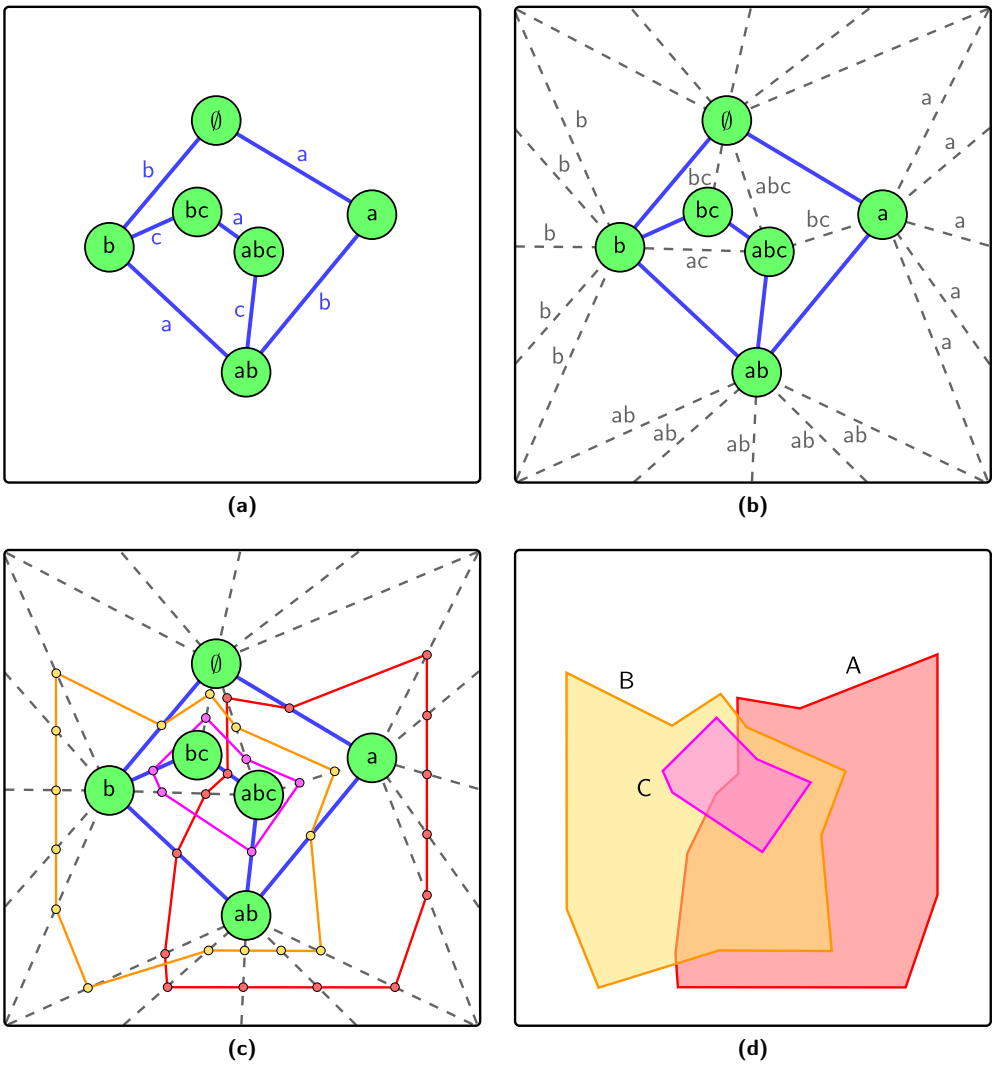


Figure 4.3: Routing the cluster curves in well-formed Euler diagrams. **(a)** The original super-dual graph. **(b)** The graph is triangulated by adding the additional dashed edges and by considering additional 16 points in a frame around the graph. The edges have been labelled with the letters that differ in the label of their extremities. **(c)** Since some edges are labelled with multiple letters, it is necessary to assign the order in which the cluster curves cross each edge. **(d)** The resulting diagram. This drawing can be further improved through a force-directed optimisation of the triangulated graph.

Remarks. This method provides very convincing results on small diagrams, especially once the mentioned optimisation procedures are applied to the triangulated graph. The results on large diagrams present lower aesthetics, possibly due to the fact that the super-dual quickly grows in complexity and leads to a triangulated graph that is difficult to optimise.

We have a couple of minor observations on this method. First, general force-directed algorithms cannot formally ensure that no crossings will be inserted, even when starting from a very regular and stable initial configuration such as that of the triangulated graph. To formally avoid incorrect layouts, it is possible to use specific force-directed algorithms such that of Tutte [103] or of Bertault [6].

The second observation regards the choice of nesting portions of the diagram. The algorithm independently draws and merges the portions of the diagram that have no curve intersections. Since the merge operation consists of inserting a portion in the correct face of another, there could be issues with the dimension of the nested component, which might appear undersized compared to the rest of the diagram.

On the other hand, this choice presents a significant benefit. Splitting the diagram generation into smaller sub-instances could greatly decrease the complexity of the problem, and with that the computation time required. Also, it might be possible to readjust the size of the cluster regions with post-generation optimisation algorithms, that will be presented in section 4.1.5.

4.1.2 Standard Euler Diagrams

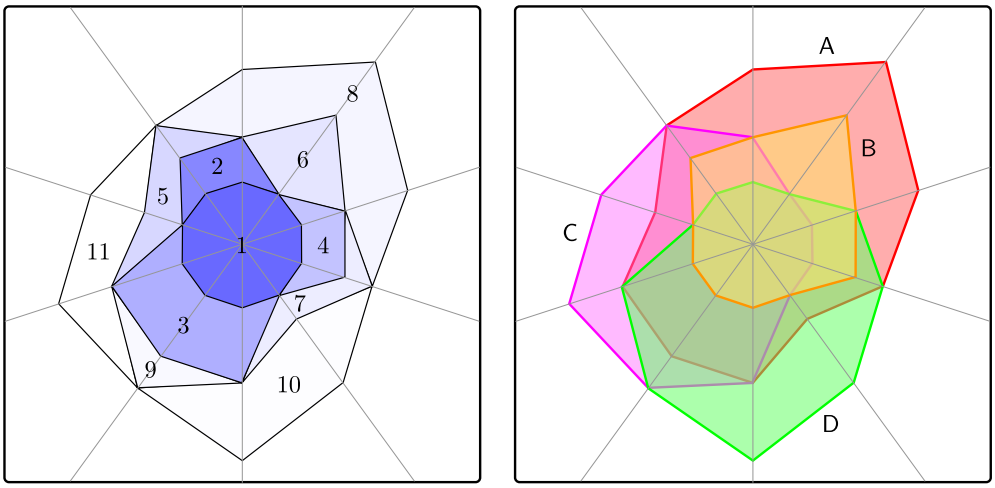
In his doctoral thesis, Chow [12] studied the generation of different kinds of Euler diagrams in great detail. The approach chosen for the diagram drawing is similar to that of Flower, Fish and Howse [39], since it is also based on the detection and drawing of a backbone graph, called *Euler dual*, that drives the depiction of the cluster curves. However, the author extensively studied the properties and characteristics of the Euler dual, their effect on the final diagrams, the classes of drawable instances and the complexity of several problems linked to the generation of Euler diagrams.

The real focus of Chow's thesis is the generation of area-proportional Euler and Venn diagrams. In area-proportional diagrams the area of the zones must be proportional to a weight that each zone has assigned as for instance the number of elements contained.

At first, he considered the problem of generating area-proportional Euler and Venn diagrams of three clusters, using only circular, elliptical, or rectangular cluster curves. Here, the author analysed which shapes always allow to obtain correct area-proportional diagrams, and proposed approximations for those that cannot be correctly drawn.

Then, the author proposed an approach for generating or re-drawing a particular sub-class of Euler diagrams to fit the area-proportionality constraints. In this method, the diagram is drawn by adding one zone at a time, starting from the central ones and moving towards the external ones in a spiral fashion. Since each zone is attached to the outer part of the diagram drawn so far, the area of the zone can be regulated by acting on its radial dimension (see figure 4.4).

Remarks. The thesis contains a large number of very interesting theoretical contributions. Also, the author puts great effort into covering different definitions of Euler diagrams, since the theoretical results highly depend on the Euler diagram properties requested for the drawing.



(a) (b)

Figure 4.4: Generation of area-proportional Euler diagrams. The method requires to order and iteratively add the zones to the current drawing, and works only for a particular class of Euler diagrams. **(a)** The order in which the zones are added. At the stage i, only the zones with an index lower than or equal to i are present in the drawing. The area of each zone can be controlled through its thickness: compare, for example, zone 3 and zone 7. **(b)** The final diagram generated with this method.

Also, the algorithms described seem to produce very pleasant diagrams, with the additional benefit of the area-proportionality property. However, the method proposed for diagrams with more than three clusters only works on a specific class of inputs, and the document does not include diagrams featuring more than half a dozen of clusters. Therefore, it is not possible to discuss the utility of the method for larger diagrams.

4.1.3 Extended Euler Diagrams

Verroust and Viaud [107] approached the Euler diagram generation problem from a practical point of view. They were asked to produce a method for visualising the relationships between categories of video and audio recordings, and decided to rely on Euler diagrams. The only diagrams having a generation algorithm available at that time, well-formed Euler diagrams [40], were unfortunately too restrictive, as they precluded the generation of a diagram for many input instances.

Therefore, the authors defined the less constrained class of the extended Euler diagrams, and developed a method to generate them. The construction of the diagram is very similar to the other approaches. First, the backbone graph is identified and drawn. Again, the graph must be planar and the subgraph induced by the nodes of $\mathcal{A}(s)$, for each $s \in S$, must be connected. Then, the cluster curves are identified by triangulating the graph and routing the curves through the resulting faces.

In the paper, it is proven that such a method works for every input clustering having eight or less clusters, and undrawable instances that exceed this number are provided.

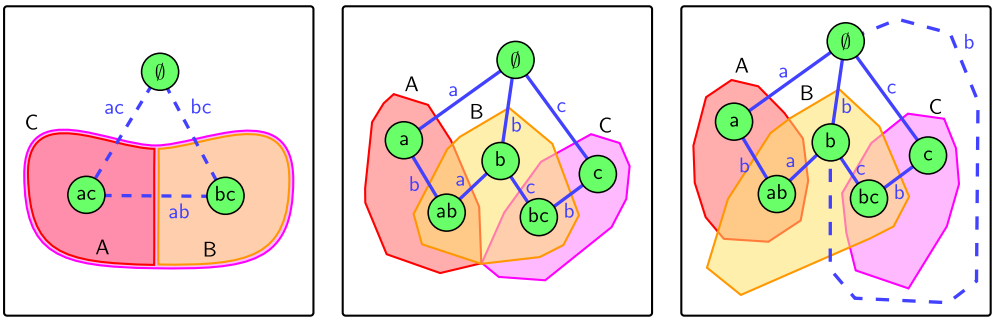


Figure 4.5: Extensions of the well-formed diagram method to draw standard Euler diagrams. **(a)** Edge insertion for the concurrency enforcement. In this first example, the dashed edges, that are not inserted by the original method, are inserted as the diagram requires concurrent curves. **(b)** Edge insertion for the elimination of brushing points. In this second example, we have an unnecessary contact between the clusters A and C. **(c)** The insertion of the dashed multiple edge allows to remove the brushing point from the previous configuration.

Remarks. The most interesting aspect of the paper regards the drawability of Euler diagrams. Even though the authors developed the method to overcome the drawability issues of the previous method, and consequently extended the standard Euler diagram definition, the possibility of generating a valid diagram was insured only up to eight clusters. This shows the complexity of the Euler diagram generation problem, as well as the necessity of violating the connected cluster region property to provide a drawing for every input instance.

Also, the paper has the merit of sketching the first concrete method for the depiction of the cluster curves, since the method of Rodgers et al. [85] was proposed a few years later. However, this second method has the benefit of reducing the concurrency of the cluster curves, that is largely used in the method of Verroust and Viaud.

4.1.4 Relaxed Euler Diagrams

The method for generating well-formed diagrams [39] has been extended by Rodgers, Zhang and Fish [84] in order to draw relaxed Euler diagrams. The extended algorithm differs from the previous for a number of features that adapt the generation of the super-dual to the non-welformedness conditions, such as concurrent curves and disconnected clusters. It also differs for a slightly modified cluster curve routing.

Starting from the initial configuration of the super-dual graph, the algorithm proceeds with the edge removal phase to meet planarity and connectivity conditions, as in the previous method. However, if the edge removal leads to a graph that respects the planarity but not the connectivity condition, the graph is not discarded as an undrawable input and the generation continues with the following steps.

When the graph is not well-formed, the algorithm inserts additional edges between the super-dual nodes that differ for two or more letters. This is done to enforce the concurrency on the cluster curves when the diagram requires it (see figure 4.5a). Then, as in the previous method, the super-dual is embedded on the plane taking care of placing the node associated with the null zone ∅ on the external face.

Once the graph is drawn, the algorithm proceeds with a second edge addition

phase, where some multiple edges are routed on the embedded graph to avoid the generation of unnecessary brushing points (see figures 4.5b and 4.5c). Then, the graph is triangulated and the cluster curves are drawn in a way similar to the previous method [85].

Remarks. The algorithm presents most of the advantages and disadvantages of the method it extends. In particular, relatively small diagrams are nicely drawn, as the algorithm manages to assign the cluster regions so that the intersections are clear and readable. The choice of independently drawing and merging portions of the diagram is also preserved, along with the advantages and drawbacks it induces.

Unfortunately, as with previous methods, very large diagrams seem not to be the target of the algorithm, as the ability to deal with large input instances is not discussed in the paper.

4.1.5 Specialities

Significant efforts have been done to improve specific phases of the diagram generation, or to produce diagrams with a different approach.

In this section, we first present some methods that improve the aesthetics of already generated diagrams, in the attempt of making them more readable and comprehensible.

Then, we present two methods that generate the diagrams by adding a cluster curve at a time. This can be used both to modify an existing diagram, and to generate new diagrams from scratch.

Finally, we describe a method that utilises very readable cluster curves, but at the expense of a heavy use of disconnected cluster regions and/or duplicated elements.

Optimisation of the Cluster Curves

Flower, Rodgers and Mutton [41] proposed a method to optimise the shape of the cluster curves, and with that the aesthetics of the diagram. The method requires to interpret the cluster curves as a path of nodes and edges, and to evaluate a pool of metrics. The metrics evaluate characteristics such as the area of the diagrams, cluster regions and zone regions, the smoothness of the curves, and several others.

The values calculated for the metrics are optimised through a hill climbing approach. The nodes are therefore moved only in directions that improve the value of the metrics, until no further progress is possible. The nodes are moved both individually and in groups, in order to accelerate the convergence to an optimal configuration.

Micallef and Rodgers [72] also proposed to interpret the cluster curves as paths, but they used a force-directed algorithm to optimise them. In addition to the standard attraction and repulsive forces, other forces were inserted to ensure a correct configuration of the resulting diagram.

Remarks. The optimisation of the cluster curves is, in our opinion, a very important step of the generation of Euler diagrams. We believe that it is not convenient to overload the construction phase with non-structural aesthetics considerations. For example, in the first phases it is better to focus on the general shape of the final diagram rather than on the actual spacing between the curves or the cluster elements. The latter aspects can then be optimised in a post-processing computation, as proposed by the above methods.

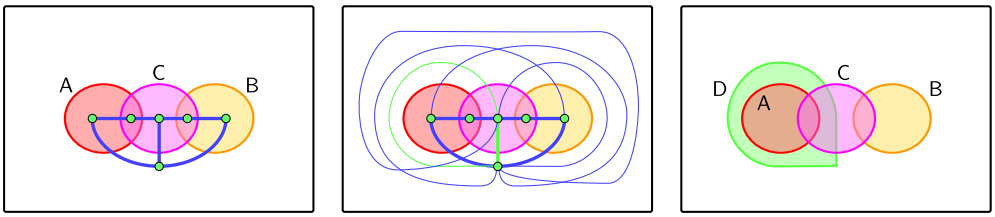


Figure 4.6: Inductive generation of Euler diagrams. **(a)** The initial configuration of the diagram and its backbone graph. **(b)** In order to insert a new cluster curve, we insert multiple edges to the backbone graph (drawn thinner). The cycle chosen to route the cluster curve is highlighted in green. **(c)** The final diagram, containing the additional cluster D.

The results produced with the hill-climbing optimisation are very aesthetically pleasing, especially when all the metrics are considered and are well balanced between each other. Unfortunately, according to Micallef and Rodgers [72], the method is computationally expensive and impractical for large diagrams.

The force-directed optimisation of the cluster curves, also used in our method, seems to be able to scale to larger diagrams. Unfortunately, the control over the diagram shape is much less reliable, since all the characteristics considered by the hill-climbing optimisation need to be translated into appropriate forces on the graph nodes.

Inductively Generated Diagrams

Some authors studied the problem of inserting new cluster curves, corresponding to additional clusters, to an existing diagram. The technique can also be used to generate new diagrams from scratch, if we start with an empty initial diagram and we insert one curve at a time.

The first method proposed [94] requires to insert additional edges to the backbone graph, and to use the graph elements to route the cluster curves to be inserted (see figure 4.6). The new edges are inserted between nodes already adjacent, and are therefore meant to provide new ways of routing the cluster curves.

The second method [95] extends the first by constructing and using a more complex graph, called a hybrid graph. The authors also explain how to enforce the properties that characterise the well-formed Euler diagrams, such as curve simplicity, absence of concurrency and of multiple crossings.

In both cases, if s is the new cluster to be inserted and t_s is its curve, the graph is studied in order to find a cycle that crosses the intersecting clusters of s, and that splits the zones in the desired way. If such a cycle exists, t_s is routed over the nodes and edges of the cycle and added to the current drawing. This operation clearly requires the recomputation of the backbone graph at every iteration, since the insertion of new curves generates new zones (and therefore new nodes in the backbone graph) and alters their adjacency relations (the edges of the backbone graph).

Remarks. The insertion of new cluster curves in an existing diagram could be very useful in a data investigation scenario, where the user interacts with the drawing in order to discover new insight on the data. In fact, the technique greatly preserves the

mental map of the user (see page 49), that would be lost if the diagram was generated from scratch at each modification.

However, not all diagrams can be generated with this method, since some diagrams would require to pass through invalid configurations in order to obtain a valid final drawing. The authors mention the case of a Venn diagram with five clusters: to be drawn correctly, we would need to add the fifth cluster curve to an invalid four-clusters Venn diagram, characterised by disconnected zones. Since a Venn diagram with four clusters has a valid drawing, the algorithm would produce this configuration, making the addition of the fifth curve impossible.

Inductive Pierced Diagrams

Stapleton et al. [96] studied the problem of inductively generated diagrams from another point of view. They defined a *single piercing curve* as a curve that intersects exactly one other curve, and a *double piercing curve* as a curve that intersects exactly two other curves. They then considered diagrams that have only circular cluster curves, and that can be generated by inductively adding one single or double piercing curve at a time.

The generation of piercing diagrams requires to first identify whether the input clustering can or cannot be generated with this method. Then, at each insertion, the algorithm needs to identify the curves that contain or intersect the one to be inserted, and to compute the position and dimension of the curves in order to respect these relationships.

Since the algorithm can only draw a restricted class of diagrams, the authors aim to combine this method with general inductive methods to be able to draw a larger input class. They also plan to provide a method to re-adjust size and position of the curves to increase the aesthetics of the final diagram.

Remarks. This method is very interesting for the properties that are enforced in the final diagram. Circular shapes greatly help to identify the cluster regions, providing a very clear representation of containment and overlap relationships. Also, the method identifies the essential parameters for the positioning of the curves (the intersection and containment relationships), allowing to design an eventual aesthetics optimisation that has a more direct control on the diagram. Force-directed curve optimisation methods, for example, are much more difficult to control, as there is a greater number of variables that come into play (all the forces acting on each point of the cluster curves).

On the other hand, these choices present significant drawbacks. Because of the generation process and the circular shapes imposed, not all input instances can be drawn. Also, the circular shapes might cause a considerable wasting of space and inequality on the cluster dimensions when dealing with complex or large clusterings (see figure 4.7). Finally, the extension of the method to higher levels of piercing, such as triple or quadruple piercing curves, might cause an excessive increase of the problem complexity.

However, considering that this approach is relatively new and that some of the above problems might be corrected by combining it with other algorithms, the piercing method can have an important impact on the future methods for the generation of Euler diagrams.

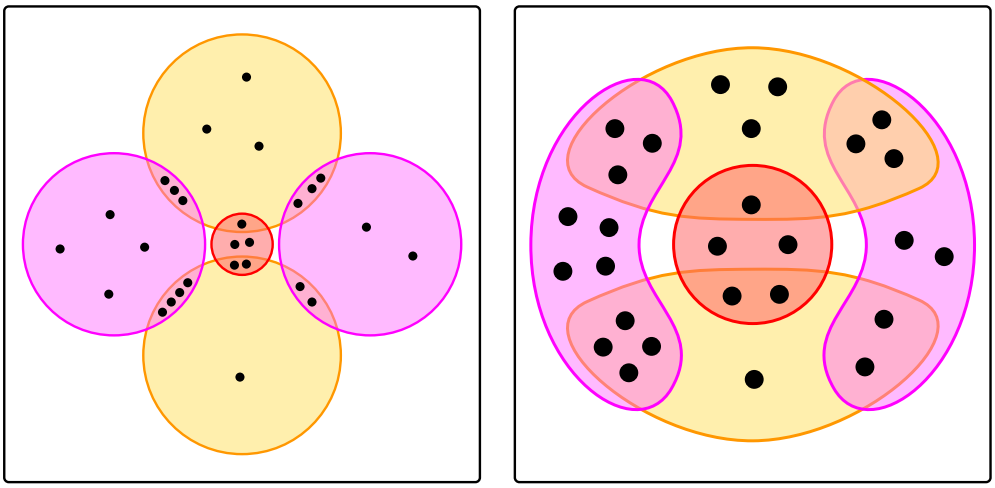


Figure 4.7: Problems related to enforcing circular shapes in Euler diagrams. **(a)** The diagram style forces an imbalance in the zone sizes. The problem worsens with the increase of the diagram size and complexity. **(b)** A diagram drawn without enforcing circular shapes. The diagram can be much more compact and can better utilise the space, without excessive consequences for the comprehension of the cluster curves.

Untangled Euler Diagrams

Riche and Dwyer [83] proposed two methods to generate Euler diagrams with more readable overlaps.

According to the authors, Euler diagram comprehensibility would greatly benefit from simple and convex cluster regions. In order to enforce this property without restricting the input class, the authors decided to make a large use of disconnected cluster regions and to use only rectangular regions.

The first method, ComED (see figure 4.8a), splits the cluster intersections into a strict hierarchy, so that the diagram can be depicted without partial overlap between the rectangular regions. For instance, for a diagram with $S = \{\mathsf{A}, \mathsf{B}\}$ and $Z = \{\mathsf{a}, \mathsf{ab}, \mathsf{b}\}$, we could build a hierarchy in which ab is child of a, and b is isolated, which leads to a diagram in which a and b are associated with disjoint rectangular regions, and ab is associated with a rectangular region fully contained in that of a.

The second method, DupED (see figure 4.8b), aims to avoid any kind of overlap between the cluster regions by duplicating the original elements contained in a cluster. The resulting diagram will feature a disjoint rectangular region for each cluster, containing all the cluster elements. In both ComED and DupED, links are used to identify the disconnected portions of a cluster region or the duplicated elements.

The authors present a software that allows to interact with such visualisations, that also features the possibility of moving from ComED to DupEd. They also present the results of a user study which evaluates the benefits of both styles and compare them with hand-drawn Euler diagrams, and which provides some evidence of the validity of the visualisation.

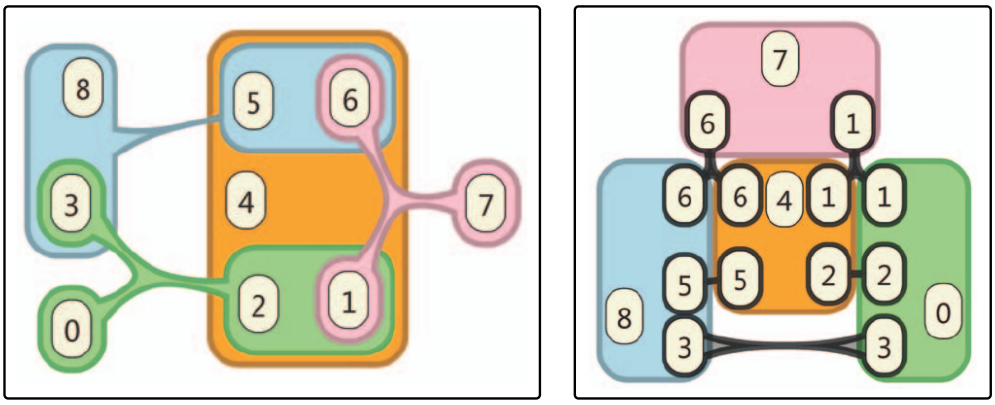


Figure 4.8: Examples of untangled Euler diagrams (courtesy of Riche and Dwyer [83]). **(a)** The diagram drawn according to the ComED style. **(b)** The diagram drawn according to the DupED style.

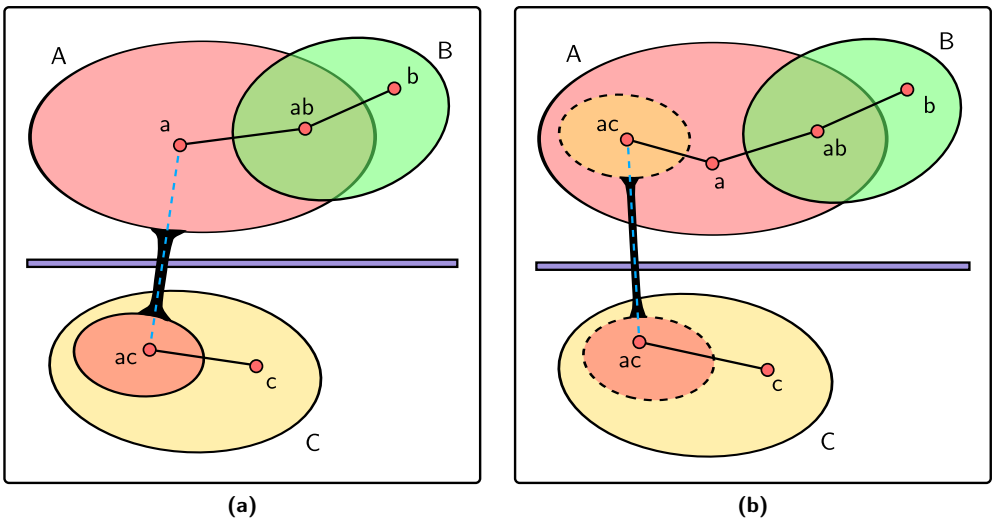


Figure 4.9: Solutions suggested to improve the readability of complex diagrams. **(a)** Linking of disconnected portions of a cluster region. **(b)** Duplication of a shared zone region.

Remarks. This method is extremely interesting for its different approach to the diagram generation, and for the quality of the resulting drawings. We already identified, in a previous publication [90], the two strategies behind ComED and DupED (the insertion of links between the disconnected components of the cluster regions and the zone duplication) as potential ways of making an Euler diagram more readable (see figure 4.9). However, the authors have the merit of including these strategies into a well realised generation procedure, which is able to generate informative and readable diagrams even for complex input.

The interaction implemented, as well as the possibility of switching from one style to another by dragging the zone regions, are also very interesting and well designed.

4.1.6 Methods with Analogies to Euler Diagrams

To conclude, we present a couple of information visualisation methods that have high similarities with the generation of Euler diagrams, but that do not comply entirely with this task. Both methods involve the identification of regions that encompass a set of elements, as it happens with the cluster regions of Euler diagrams.

However, since the methods do not alter the initial positions of the cluster elements, they can hardly be used to generate Euler diagrams as defined in this thesis. They can instead provide meaningful ideas for the identification and the depiction of the cluster curves and regions.

GMaps

Gansner, Hu and Kobourov [48] developed Gmap, a method for visualising a non-overlapping clustering of a graph as a geographical-like map (see figure 4.10). To generate Gmaps, a graph is drawn in the plane and clustered with an algorithm that fits the characteristics of the drawing, so that the clusters detected collect elements at a close geometrical distance. Then, the mapping algorithm identifies the borders of the clusters and the map colouring algorithm assigns colours to the clusters so that nearby clusters have colours that greatly differ from each other.

The mapping algorithm inserts a great number of dummy nodes at random positions in the drawing, with the only constraint of not being too close to an original graph node. Then, the Voronoi diagram for the graph is computed, and the cells containing nodes that belong to the same cluster are merged to form the cluster region. The method also takes into consideration the glyph or the label associated with a node, generating cluster regions that fully contain them as long as they do not overlap with those of other nodes.

The colouring algorithm receives as input a number of colours equal to the number of clusters, and assigns them to the cluster regions so that chromatically close colours are preferably assigned to clusters at great distance in the drawing.

Remarks. From an Euler diagram point of view, the most interesting parts of the method are the mapping and the colouring algorithms. The mapping algorithm could be used to efficiently detect the cluster regions, but it would need to handle overlapping clusters and to ensure that we would not generate non-desired zones in the diagram.

The colouring algorithm could instead be directly applied to the generation of Euler diagrams, since we could compute distant colours for the overlapping clusters as it is done in Gmaps for the clusters that share a boundary.

Bubble Sets

Collins, Penn and Carpendale [15] proposed a technique to display overlapping clusters that collects elements with an assigned position. Since bubble sets do not alter the current element placement, they can be used to enhance existing diagrams and visualisations, such as, for instance, scatter plots (see figure 4.11).

With bubble sets, we dynamically create regions that aim to enclose all the elements of a cluster and no other ones. To do so, edges might be inserted between the nodes of one cluster and routed to avoid the nodes that are not contained in it. The area around the nodes and edges is then modelled as an energy field, and the cluster boundaries are computed as the points of the resulting energy field having the same value (isocontours).

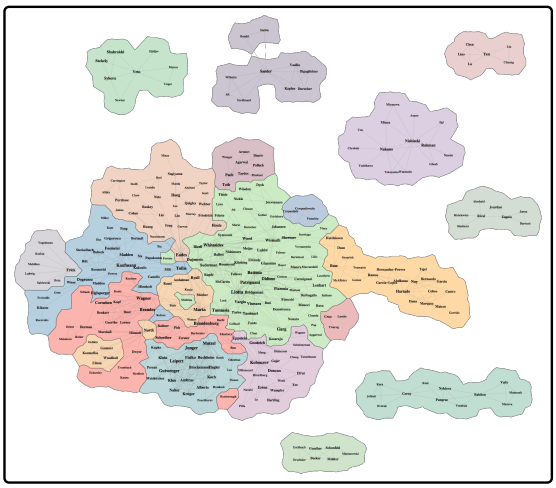


Figure 4.10: Gmap showing the co-authorship relations of the articles presented in 20 years of Graph drawing conferences (courtesy of Gansner, Hu and Kobourov [48]).

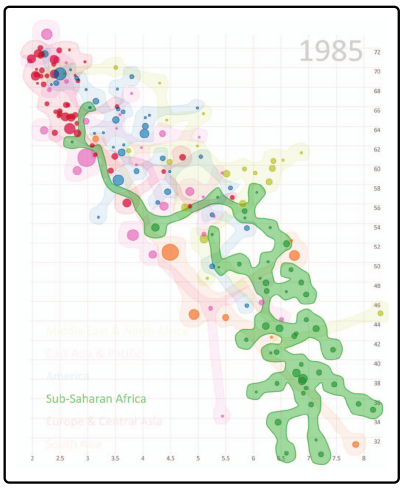


Figure 4.11: Bubble sets in a scatter plot showing the fertility rate by life expectancy by country (courtesy of Collins, Penn and Carpendale [15]).

The cluster region computed in this way is connected, and since it is dynamically generated, it would adapt to an eventual node movement and respond to the interaction of the user.

Remarks. The authors provide several case studies and applications that all prove the very high aesthetics and usefulness of the method. The computation of bubble sets is fast enough to be used in real-time applications, and the cluster regions readily respond the user interaction.

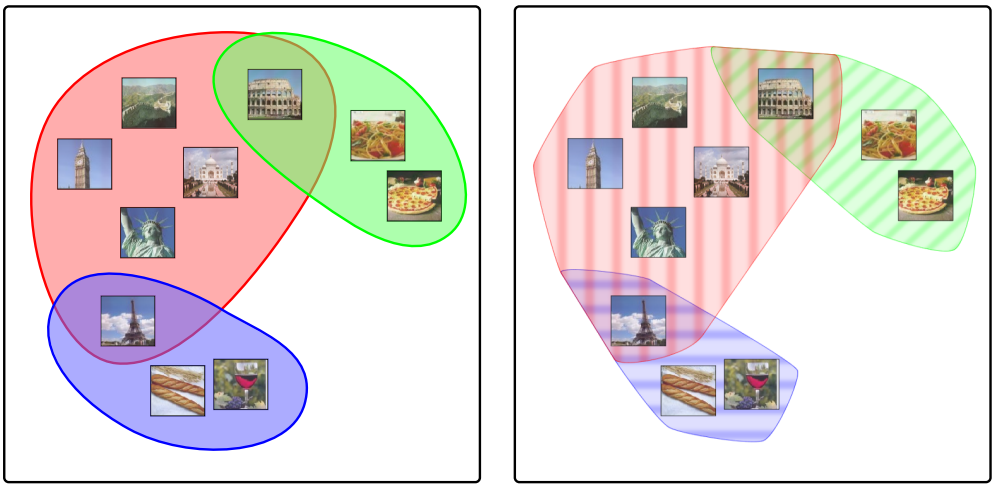
However, the aim of bubble sets and Euler diagrams only partially correspond. Bubble sets are characterised by connected cluster regions, but also for highly disconnected zone regions. Moreover, bubble sets might generate a high number of null zones, that is generally forbidden in Euler diagrams but that is not problematic since the depiction of the original elements removes the ambiguities (see section 2.2.2).

4.2 Euler Representations

As explained in chapter 1, the aim of our work is to develop a visual method for the analysis of overlapping clusters based on Euler diagrams. This application field adds a few constraints to the generation of Euler diagrams as considered by other authors. First, a diagram must always be produced. Second, the method must be reasonably fast even for large input instances. Third, the diagram must include the elements of the clustered graph.

While with the standard Euler diagram generation we can only rely on the visual encoding (the drawing style and conventions), a method inserted in a visualisation framework can also make use of interaction techniques to mitigate problems that the visual encoding cannot easily address.

For the reasons explained we devised a visual encoding, called *Euler representation*, with the following characteristics:



(a) (b)

Figure 4.12: Comparison between Euler diagrams and Euler representations. The diagrams are composed of three sets: the set of the French things (blue), the set of Italian things (green), and the set of monuments (red). The sets overlap on the Colosseum and on the Eiffel Tower. **(a)** A classic Euler diagram. **(b)** An Euler representation. Compared to the previous diagram, we can note the transparent textures and the introduction of concurrency on portions of the cluster curves.

- *Based on relaxed Euler diagrams.* Since we cannot have undrawable instances, it is necessary to work with the less restrictive class of Euler diagrams. Clearly, the generation process should avoid as much as possible violating the Euler diagram properties, in particular with respect to the connectivity of the cluster regions.
- *Large level of cluster curve concurrency.* As we might be required to draw diagrams having hundreds of sets, we chose to insert a large amount of concurrency between the cluster curves. This will help reduce the running times both in the generation phase and in the cluster curve optimisation phase, since the concurrency reduces the number of lines in the drawing.
- *Textures, colours and transparency.* Since the cluster curve concurrency obstructs the identification of the cluster regions, we insert some graphical techniques to help discriminate the clusters that form the zones of the diagrams. We fill the cluster regions with coloured, semi-transparent textures that permit the identification of the patterns even in the presence of multiple overlaps.

A comparison between a standard diagram and the Euler representation of a sample clustering is shown in figure 4.12.

4.2.1 The Generation Process

The generation process is composed of the following sequence of steps:

1. *Zone graph construction.* In the first step, a backbone graph, called *zone graph*, is constructed. The nodes of the graph correspond to the expressed zones of the diagram, that are identified by studying the inclusion relationships of the cluster

members. The edges of the graph are instead iteratively inserted in the graph, following metrics that highlight the most important ones at the current stage of the iteration. As in previous methods, the zone graph must be planar.

2. *Zone graph drawing.* Once the zone graph elements have been decided, the graph is embedded on the plane using a planar graph drawing algorithm. Since the layout is generally not very aesthetically pleasing, the graph layout is improved with a different force-directed algorithm that preserves graph planarity.

3. *Grid graph construction.* Instead of routing the cluster curves in the zone graph, as done in previous work, we obtain them by transforming the zone graph elements into regions. To do so, we construct a second graph, called *grid graph*, that wraps the nodes and edges of the zone graph.

4. *Grid graph drawing.* Once the grid graph has been computed, the members of each zone are inserted in the corresponding region. This is possible since the zone graph has a node for each expressed zone, and since the grid graph transforms the zone graph nodes into regions. The layout of the grid graph is again optimised through a force-directed method.

5. *Depiction of the cluster regions.* Finally, we identify the polygonal contours of the cluster regions and we smooth them by using Bézier curves. Also, we apply coloured textures to help discern clusters that overlap in a certain region.

The result of the operations performed at each stage of the computation is shown in figures 4.13 and 4.14.

4.2.2 Comparison with Methods in the Literature

The generation and drawing of the zone graph does not present substantial differences with that of other methods. However, a precise description of how to choose the edges to insert or delete, and of how to embed the graph on the plane is often missing, limiting the possibility of comparing our method with that of others.

The generation and drawing of the grid graph substitutes the routing of the curves in other methods. Our approach tends to be simpler and faster, but to penalise the quality of the drawing with less regular shapes and high levels of concurrency. Since these aesthetics issues are less evident with the increase of the diagram dimension, and since the computation time is a serious problem only for large input instances, this approach better suits the generation of large and complex diagrams than that of small ones.

The graphical additions we propose to assist the user on the identification of the cluster regions are typically not considered by the methods in the literature. These features are particularly important for Euler representations due to their purposes and characteristics, but could be applied to all existing methods. The cluster curves, which are the classical way of detecting the cluster regions, are smoothed to make them easier to follow. Coloured, semi-transparent patterns, that are easily identifiable even when several clusters intersect, are then inserted to provide a new method to detect the cluster regions.

In terms of the diagram generated, our method presents another major difference with respect to all other methods in the literature: the elements contained in clusters and zones are included the diagram. As the regions are now required to nicely fit their content, the improvement of the cluster shape is no more an optional step, but an integral part or the generation procedure.

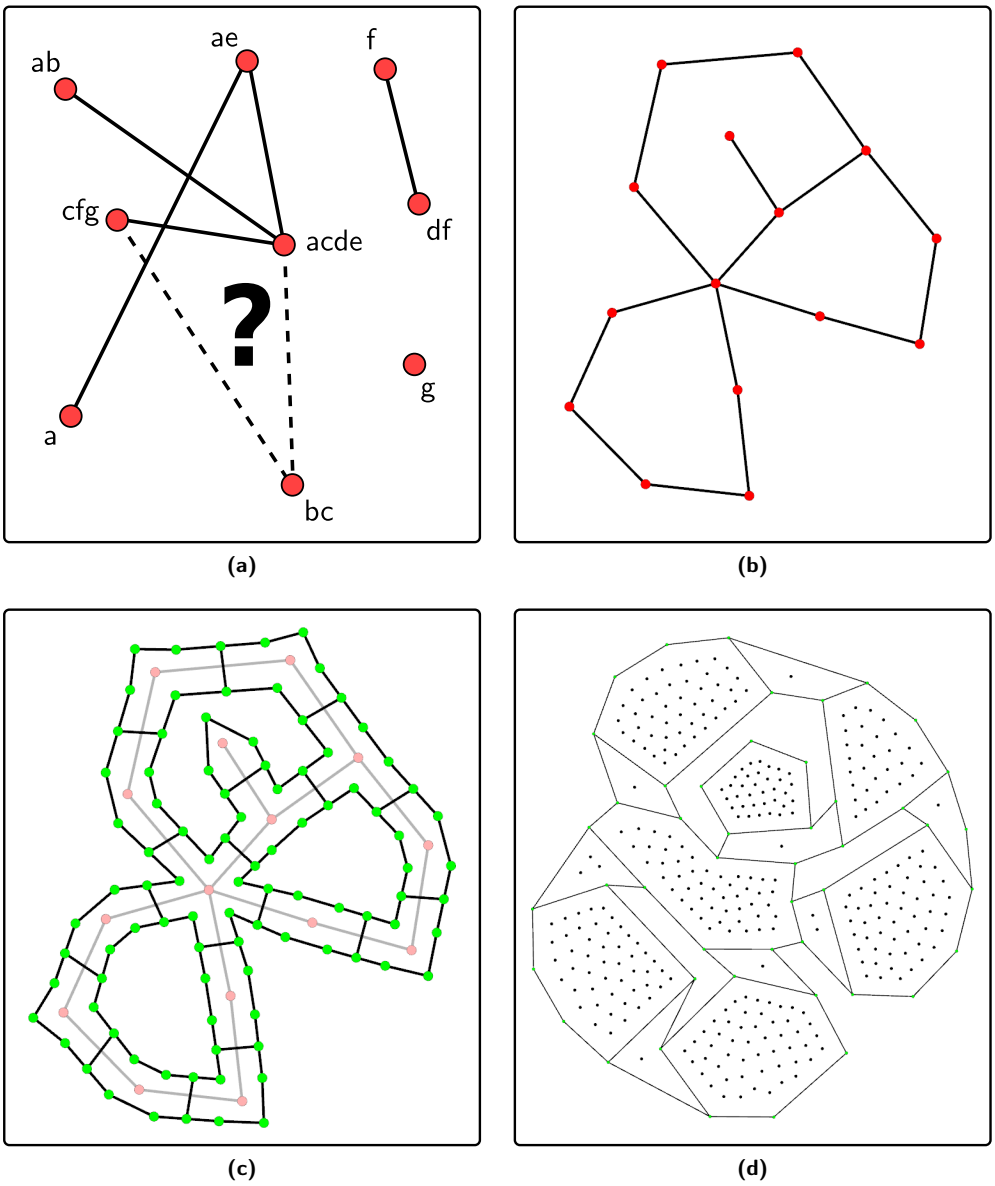


Figure 4.13: Procedure for the generation of Euler representations, up to the grid graph drawing. **(a)** Zone graph construction. The nodes and edges of the zone graph are identified at this stage. **(b)** Zone graph drawing. We identify a planar drawing for the zone graph. **(c)** Grid graph construction. The graph elements are placed on the plane to wrap the zone graph. **(b)** Grid graph drawing. The original elements are inserted in the relative zone regions, and the layout is optimised.

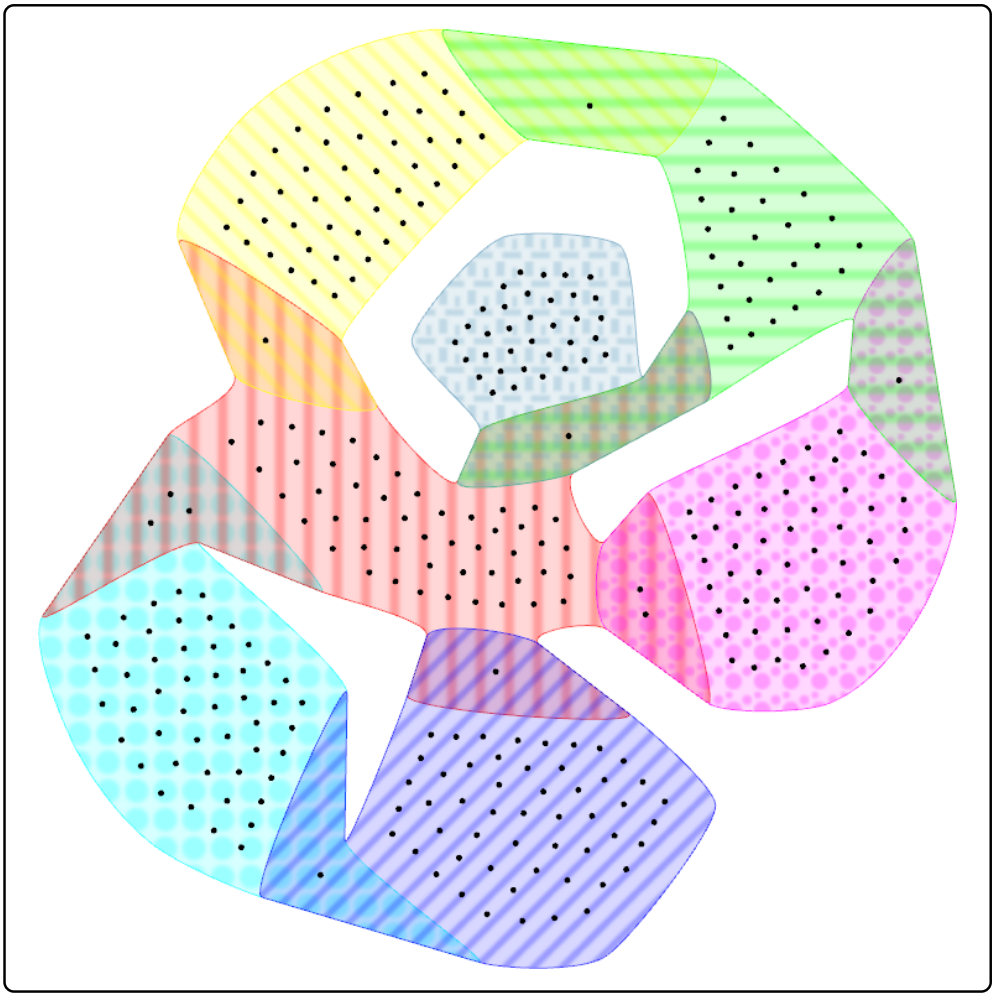


Figure 4.14: Last step of the generation procedure and results. From the drawing in figure 4.13d, we identify and depict the cluster regions using smooth curves, colours and textures.

Chapter 5

Automatic Generation of Euler Representations

In this chapter, we detail the generation process of Euler representations. As anticipated at the end of the previous section, there are five main steps:

1. *Zone graph construction:* we generate the backbone graph, called *zone graph*, by identifying its nodes and edges.
2. *Zone graph drawing:* we identify a planar layout for the zone graph that respects the graph drawing aesthetics criteria.
3. *Grid graph construction:* we generate a second graph, called *grid graph*, that wraps the elements of the zone graph.
4. *Grid graph drawing:* we place the original graph elements and we optimise the grid graph layout to obtain a first approximation of the final diagram.
5. *Depiction of the cluster regions:* we depict the cluster regions using several graphical techniques that improve the readability and the visual appeal of the drawing.

The generation procedure therefore involves three main graphs:

- *The original clustered graph.* The clustered graph that is provided as the only input of the procedure. The graph will be indicated with $G^o = (V^o, E^o, S)$, where S is the clustering (see section 3.1.11), and we will colour its nodes black.
- *The zone graph.* Shows the zones expressed by the diagram and their adjacency relations. We will indicate the graph with $G^z = (V^z, E^z)$ and we will colour its nodes red.
- *The grid graph.* Provides a first approximation of the final cluster regions. We will indicate the graph with $G^g = (V^g, E^g)$ and we will colour its nodes green.

In the following sections, we explain each step of the generation procedure. The first section details the construction and drawing of the zone graph. The second section explains the generation and drawing of the grid graph. Finally, the last section details the depiction of the cluster regions.

5.1 Generation and Embedding of the Zone Graph

In the first two stages of the algorithm we generate and draw a backbone graph that, in our method, we call zone graph.

To generate the zone graph we need to determine the nodes V^z and edges E^z that compose it. The determination of the nodes is simple since there is a one-to-one correspondence between them and the expressed zones of the diagram. The determination of the zone graph edges, however, is much more complex since we need to consider planarity, connectivity and aesthetics issues.

In the following sections, we will first explain how to efficiently detect the expressed zones of the diagram, and with that the set of nodes V^z. Then, we present the heuristic we developed to iteratively insert the zone graph edges, obtaining the final set of edges E^z. Finally, we explain how to obtain a planar drawing that respects the aesthetics criteria discussed in section 3.2.2.

5.1.1 Indentification of the Expressed Zones

The identification of the expressed zones can be performed by studying the inclusion relationships of the nodes of the original graph.

Let $G^o = (V^o, E^o, S)$ be the input clustered graph. We associate a list of clusters to each node $u^o \in V^o$, initially empty, that will contain all and only the clusters that contain u^o. To fill the lists, for each cluster $s \in S$, we scan the nodes contained in the cluster $u^o \in s$ and insert s on their cluster list. Once generated, the lists provide both the set of the expressed zones Z of the diagram, and the partitioning of the original nodes in zones (see algorithm 5.1 on page 81).

The process is explained for an example input instance having five original graph nodes and three clusters:

$$\begin{array}{lcll} & & u_1^o \in \mathsf{A} & \\ \mathsf{A} = \{u_1^o, u_2^o, u_3^o\} & & u_2^o \in \mathsf{A}, \mathsf{B}, \mathsf{C} & \\ \mathsf{B} = \{u_2^o, u_4^o, u_5^o\} & \rightarrow & u_3^o \in \mathsf{A} & \rightarrow \quad z_2 = \mathsf{b} = \{u_4^o, u_5^o\} \\ \mathsf{C} = \{u_2^o\} & & u_4^o \in \mathsf{B} & \quad z_3 = \mathsf{abc} = \{u_2^o\} \\ & & u_5^o \in \mathsf{B} & \end{array}$$

Since we obtained three different cluster lists, there are three expressed zones: a, b and abc. The lists also provide the content of each zone, since an original graph node u^o is contained in the zone $\mathcal{Z}(q)$, where q is the list of clusters of u^o. For example, the list of u_2^o is $\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, and therefore u_2^o is contained by $\mathcal{Z}(\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}) = \mathsf{abc}$.

Notation. Due to the one-to-one correspondence between zones and zone graph nodes, in the following sections we might abuse mathematical terminology and notation. For instance, by saying that we will insert edges between the zones z_1 and z_2, we mean that we will insert an edge between the zone graph nodes that correspond to those zones. Also, by indicating $\mathcal{A}(u)$, $u \in V^z$, we refer to the associated clusters of the zone that corresponds to the zone graph node u.

5.1.2 Insertion of the Zone Graph Edges

Once the set of the zone graph nodes is detected we proceed with the insertion of the zone graph edges. When doing so, we consider the following conditions:

- *Planarity.* The zone graph must be planar, as edge crossings correspond to undesired overlaps between the cluster regions that we will generate.

- *Cluster connectivity.* The subgraph induced by the zones $\mathcal{A}(s)$, for every $s \in S$, should be connected. When this happens, we obtain connected cluster regions (see figure 5.1a).

- *Cluster intersection connectivity.* The subgraph induced by the zones $\alpha*$, for every possible α, should be connected. This generalises the previous point by enforcing the connectivity of the regions where two or more clusters overlap (see figure 5.1b).

The first condition is strictly necessary in the generation process, and must therefore always be respected. The second condition allows to control the Euler diagram property "no disconnected clusters", and we will try to respect it whenever possible. The third condition does not actually refer to any Euler diagram property, but it deeply influences the readability of large diagrams. In fact, a cluster overlap composed of multiple disconnected regions can be difficult to spot, and can lead to misleading conclusions on the elements shared by the clusters (see figure 5.2b).

No other Euler diagram properties are enforced at this point. The properties required by relaxed Euler diagrams are in fact enforced by the whole process, rather than only by the selection of the zone graph edges. Properties typical of well-formed diagrams could be achieved by carefully inserting the zone graph edges, but we will not take them into direct consideration. As explained at the end of the previous chapters, concurrency and multiple crossing points are distinctive traits of Euler representations.

The Algorithm

Given a set of nodes V^z, there are a very large number of sets of edges E^z that strictly respect the first condition, and that partially respect the second and the third. Unfortunately very few of them lead to aesthetically pleasant diagrams. Since the edges of the zone graph deeply influence the shape and readability of the resulting diagram (see figure 5.2), it is necessary to carefully select the edges that help us optimise the second and the third conditions.

In the next section, we will define a few metrics that measure the contribution of an edge in satisfying the properties. As these metrics are based on the connectivity of portions of the zone graph, they must consider the edges that are already part of E^z. For example, an edge that connects abc to a helps connecting the cluster A, but its contributions is needed only if the nodes are not already connected through another path. This also means that each time an edge is added to the graph, we need to recompute or update the metrics of the edges that might be inserted in the future.

For these reasons, the algorithm that we propose for the edge insertion constructs a pool of candidate edges, and at each iteration moves the edge with highest global value to E^z. Whenever the insertion of the best edge would result in a zone graph that is no longer planar, the edge is removed from the pool of candidate edges and the next best edge is considered for insertion in E^z. The procedure is stopped when the pool of candidate edges is depleted. The pseudo-code of the procedure is reported in algorithm 5.2 on page 81.

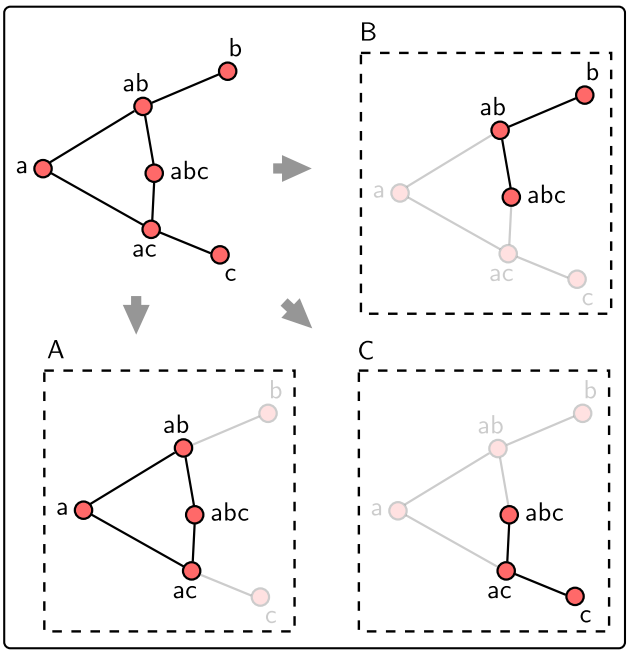


Figure 5.1: Enforcement of the zone graph conditions. We insert the edges in order to satisfy planarity, cluster connectivity and cluster intersection connectivity. **(a)** Control of the fulfilment of the cluster connectivity conditions. **(b)** Control of the fulfilment of the cluster intersection connectivity conditions.

Algorithm 5.1 Indentification of the expressed zones

▷ Cluster List Population

for all $u^o \in V^o$ **do**
  clusters(u^o) ← empty list of clusters
for all $s \in S$ **do**
  for all $u^0 \in s$ **do**
    insert s in clusters(u^o)

▷ Expressed Zones Identification

for all $u^o \in V^o$ **do**
  if expZones does not contain clusters(u^o) **then**
    expZones(clusters(u^o)) ← empty list of nodes
  insert u^o in expZones(clusters(u^o))

return expZones

Algorithm 5.2 Insertion of the zone graph edges

▷ Candidate Edges Initialisation

for all $u_1, u_2 \in V^z$ **do**
  if $u_1 \neq u_2 \ \wedge \ \mathcal{A}(u_1) \cap \mathcal{A}(u_2) \neq \emptyset$ **then**
    globalMetric($[u_1, u_2]$) ← global metric for the edge $[u_1, u_2]$
    add $[u_1, u_2]$ to candidateEdges

▷ Edge Insertion

while candidateEdges $\neq \emptyset$ **do**
  e ← best candidate

  if $G^z = (V^z, E^z \cup \{e\})$ is planar **then**
    insert e in E^z
    remove e from candidateEdges
    for all $e \in$ candidateEdges **do**
      globalMetric($[u_1, u_2]$) ← global metric for the edge $[u_1, u_2]$
  else
    remove e from candidateEdges

return E^v

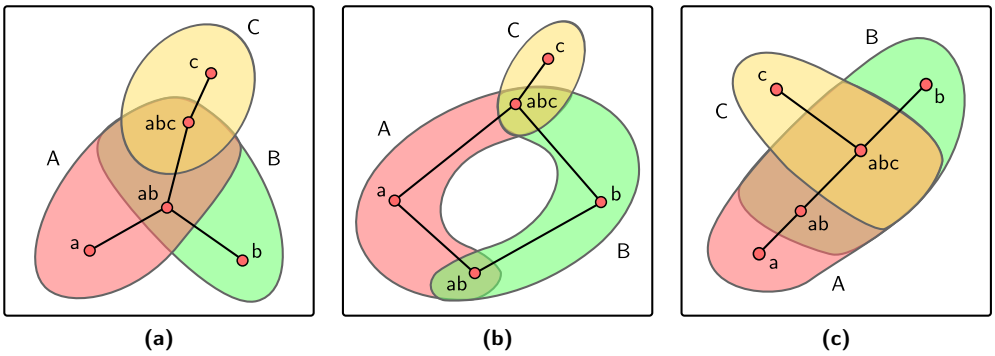


Figure 5.2: Results produced by different sets of zone graph edges. **(a)** A clear representation. **(b)** A less clear representation due to the disconnected overlap between cluster A and B. As we can see, the two clusters share the zone regions $\mathsf{ab}^{\mathcal{R}}$ and $\mathsf{abc}^{\mathcal{R}}$, but these regions are not adjacent to each other. This might cause the user to miss one of the regions, resulting in wrong conclusions being drawn on the overlap between A and B. **(c)** Another unclear configuration, since the placement of the zones makes the identification of the cluster regions harder.

The Metrics

We developed a number of metrics to evaluate the contribution of candidate zone graph edges. The metrics indicate how much the insertion of a candidate edge would assist in respecting the conditions above, or in other words, in generating a clear and readable Euler diagram.

The first metric promotes the connectivity of the cluster regions, and since this is the most important property to enforce it will have the highest importance in weighting the edges. Two other metrics are designed to penalise low aesthetic configurations, so that a more readable configuration, whenever it exists, is preferred. Finally, a global metric combines the contributions of the previous ones to provide the final edge weight.

Before describing the metrics in details, we define the concept of a cluster subgraph which corresponds to the subgraph of the current zone graph induced by the nodes that contain the same cluster letter.

Cluster Subgraph. We define the *cluster subgraph* G_s^z of a cluster $s \in S$ as the subgraph of G^z induced by the nodes $\mathcal{A}(s)$.

Promoting the Connectivity of the Cluster Regions. In order to avoid disconnected cluster regions, we need to insert edges between all nodes that belong to the same cluster subgraph. At the same time, reducing the number of edges is important since the more edges we need to insert, the more likely it is that the graph violates planarity. We will therefore favour edges that connect components in many cluster subgraphs at the same time, rather than edges that connect only a few of them.

For this reason we define the connection metric $m_c(e^z)$. The function counts the number of cluster subgraphs in which the insertion of e^z would connect disconnected components. It is worth noting that this function depends on the edges previously

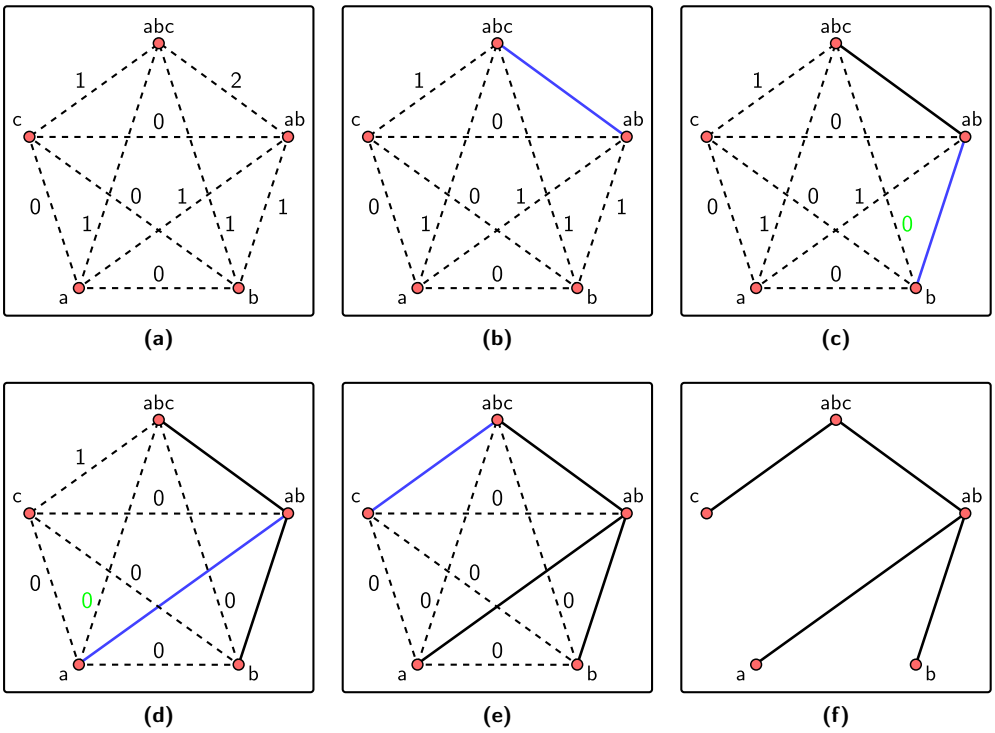


Figure 5.3: Functioning of the connection metric and its update. **(a)** The initial zone graph composed only by nodes and no edges. Each possible edge is weighted with the connection metric. **(b)** The edge with highest value is inserted in the graph. **(c-e)** Other edges are inserted, and the weights influenced by the insertion (coloured in green) are updated. **(d)** The resulting zone graph.

inserted. An example of what the metric m_c means and how it works when updating a graph, is shown in figure 5.3.

Connection Metric. Let G be the current zone graph, and G^e the current graph in which the edge e has been inserted. Let C_s be the set of connection components of the cluster subgraph G_s, and C_s^e be the analogous to G^e.

The *connection metric* $m_c(e)$ is computed as

$$m_c(e) = \sum_{s \in S} \left(|C_s| - |C_s^e| \right)$$

Demoting Low Aesthetic Configurations. In order to avoid unclear configurations (such as those in figures 5.2b and 5.2c), we penalise the insertion of edges that differ too much from each other. This is obtained by defining two other metrics.

The first function, the brushing metric, detects the number of unrelated clusters that are forced to be adjacent through this edge. For instance, the edge [ab, ac] forces clusters B and C to share a boundary, even when it is not necessary. This is not the

case for the edge [a, abc], since no unwanted boundaries would overlap.

The second function penalises edges between nodes that differ for more than one letter. For instance, the edge [a, abc] connects a zone to one contained in two more clusters, B and C, other than the first. When choosing between the edges [a, abc] and [ab, abc], the second is preferable, since the extremities differ for only one letter.

Brushing Metric. The *brushing metric* $m_b(e)$ for a zone graph edge $e = [u_1, u_2]$ is computed as

$$m_b(e) = \min\left(|\mathcal{A}(u_1)|,\ |\mathcal{A}(u_2)|\right) - |\mathcal{A}(u_1) \cap \mathcal{A}(u_2)|$$

Coincidence Metric. The *coincidence metric* $m_i(e)$ for a zone graph edge $e = [u_1, u_2]$ is computed as

$$m_i(e) = \max\left(|\mathcal{A}(u_1)|,\ |\mathcal{A}(u_2)|\right) - |\mathcal{A}(u_1) \cap \mathcal{A}(u_2)| - 1$$

The brushing metric returns one for the edge [ab, ac], and zero for the edge [a, abc]. The benefits of this penalisation are shown in an example in figure 5.4a.

The coincidence metric returns one for the edge [a, abc] and zero for the edge [ab, abc]. Figure 5.4b shows why this penalisation is helpful.

The Global Metric. The global metric combines the previous metrics into one value. Since the metrics have different importance, we weight their contribution on the global metric. The factors used in the global metric formula have been empirically determined.

Global Metric. The *global metric* $m_g(e)$ is computed as

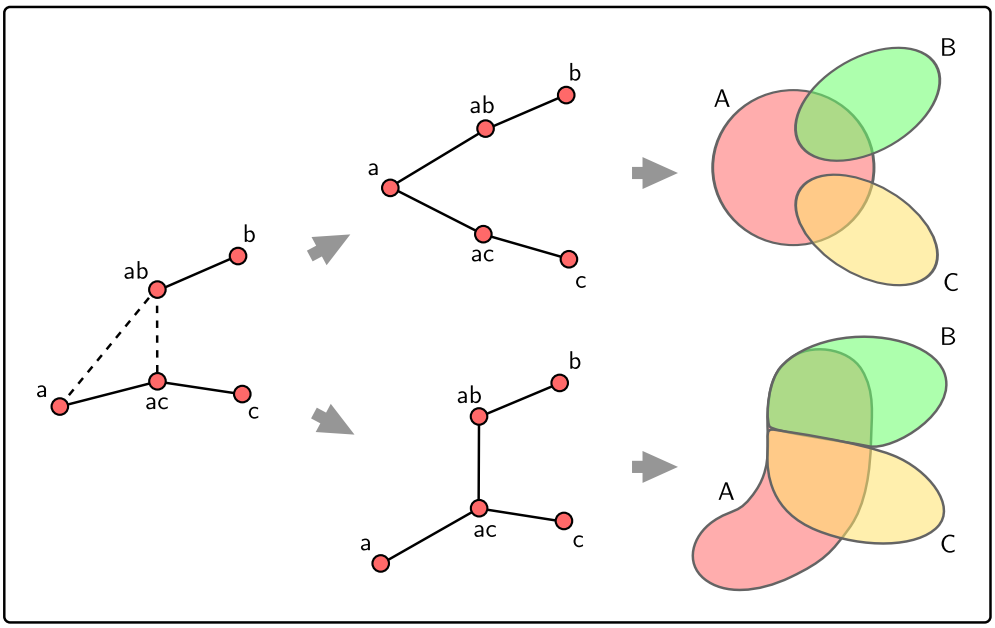
$$m_g(e) = m_c(e) - 0.15\, m_b(e) - 0.05\, m_i(e)$$

5.1.3 Embedding of the Zone Graph

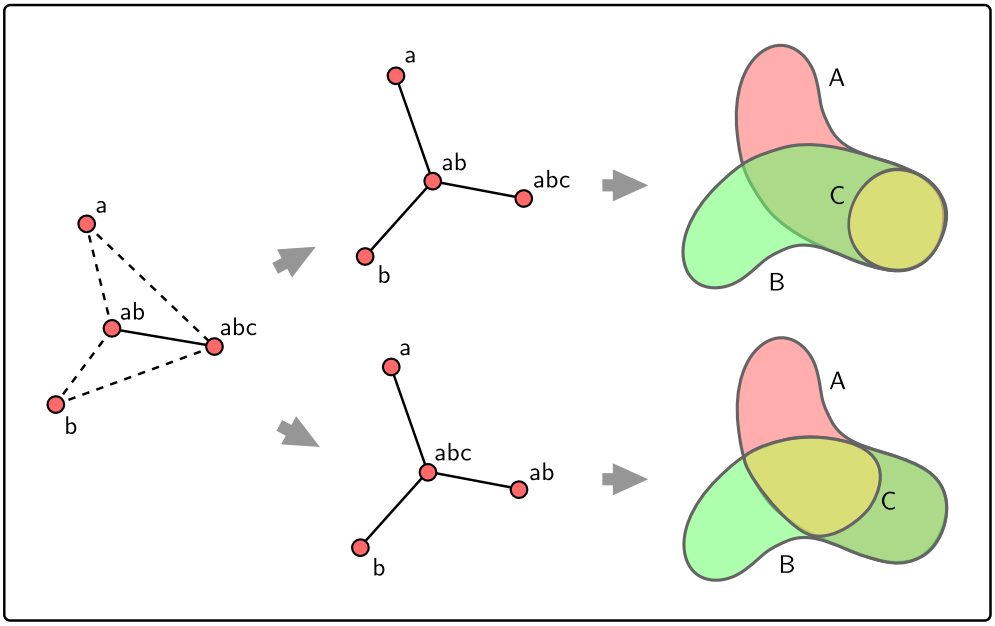
Once the elements of the zone graph are decided, we embed it on the plane. The drawing must be planar, and the aesthetics of the graph should be maximised to facilitate the optimisation of the diagram at the next steps of the generation algorithm. Particularly important are node spacing and angular resolution, as we will see in the next section.

In order to produce a drawing with these characteristics, it is possible to apply two algorithms presented in section 3.2.3. The first, `FPP` [18], allows us to identify a first planar drawing of the graph. The second, `PrEd` [6], optimises this drawing to increase its aesthetics.

Since the readability of the final diagram heavily depends on the optimisation produced by `PrEd`, and since we use the algorithm to improve both the zone graph and the grid graph, we spent a considerable effort on improving the method. The resulting algorithm, that we called `ImPrEd`, improves `PrEd` in terms of drawing quality, running time, and control over the optimisation process.



(a)



(b)

Figure 5.4: Effects of the metrics that demote low aesthetics configurations. **(a)** As a consequence of the brushing metric, the first configuration is chosen. The second zone graph leads to a less clear diagram, as B and C have an unnecessary overlap of their boundaries. **(b)** As a consequence of the coincidence metric, the first configuration is chosen. The second zone graph leads to a less clear diagram, as the zone ab is more difficult to recognise.

`ImPrEd` is explained in greater detail in chapter 6. For now, it is sufficient to know that the general behaviour of `PrEd` is preserved: `ImPrEd` is still a force-directed approach that impedes nodes from crossing edges, preserving all and only the crossings present in the input graph.

Algorithm Parameters. The algorithm `FPP` requires no parameters. `ImPrEd` requires the same parameters as `PrEd`, which are the optimal edge length (or minimal desired node-node distance) δ, the minimal desired node-edge distance γ, and the number of iterations.

Since the first two parameters influence the positioning of the zone graph nodes that correspond to the spacing between the zones of the diagram, we decided to link them to the number of zones and to the number of cluster elements that will be inserted in the zones. The formulae used to compute the parameters passed to `ImPrEd` are

$$\delta = \frac{\max\left(10,\ \frac{|V^o|}{100}\right) + \max\left(10,\ \frac{|V^z|}{5}\right)}{2} \qquad \gamma = 0.8\ \delta \qquad \text{iter} = 200$$

Results of Different Embeddings of the Graph

As explained in section 3.1.9, there might exist many planar maps for the same graphs. In other words, we might be able to generate many non-equivalent drawings.

In our algorithm, the embedding of the zone graph is automatically decided by the implementation of `FPP` that we use. However, the choice of the embedding produced would be of crucial importance for the aesthetics of the diagram.

Let us consider a complex zone graph formed by a tree rooted on the vertex of a small polygon. There are two major kinds of planar drawings that we can generate: one in which the tree is contained in the polygon, and one in which it is not. The second situation is clearly preferable over the first since a very large tree would force the edges of the polygon to stretch considerably (see figure 5.5).

Although there are methods for identifying and generating all the possible planar maps of a planar graph [21–23], their number can be exponential with the size of the graph. Moreover, it seem that there are no trivial ways of identifying the best planar map. In fact, the problem exemplified in figure 5.5 can subsist at every level of the graph, and not only on the external face (see figure 5.6).

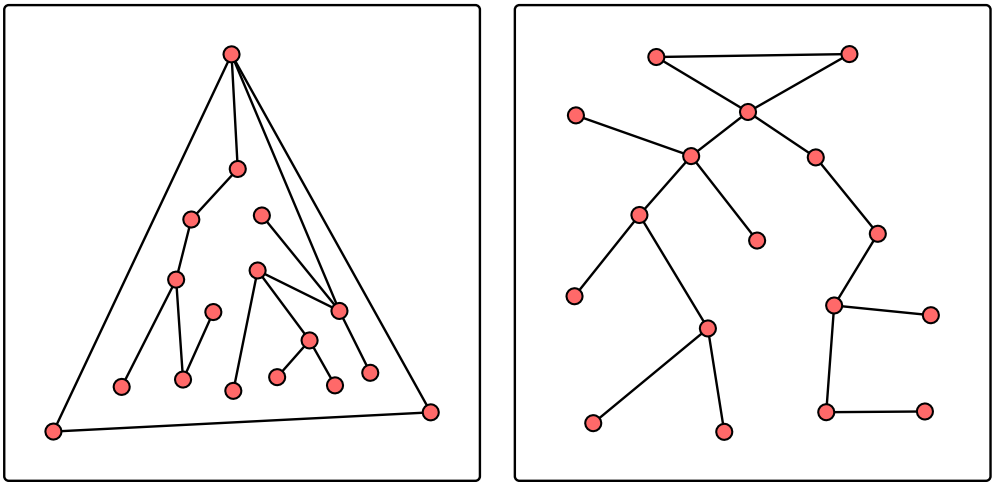
Due to the importance of the problem, which affects all the generation algorithms that use a backbone graph to generate an Euler diagram, it should be investigated further.

5.2 Generation and Improvement of the Grid Graph

In the third and fourth step of the algorithm we generate and improve a second graph, called the grid graph, that will provide the borders of the zone and cluster regions.

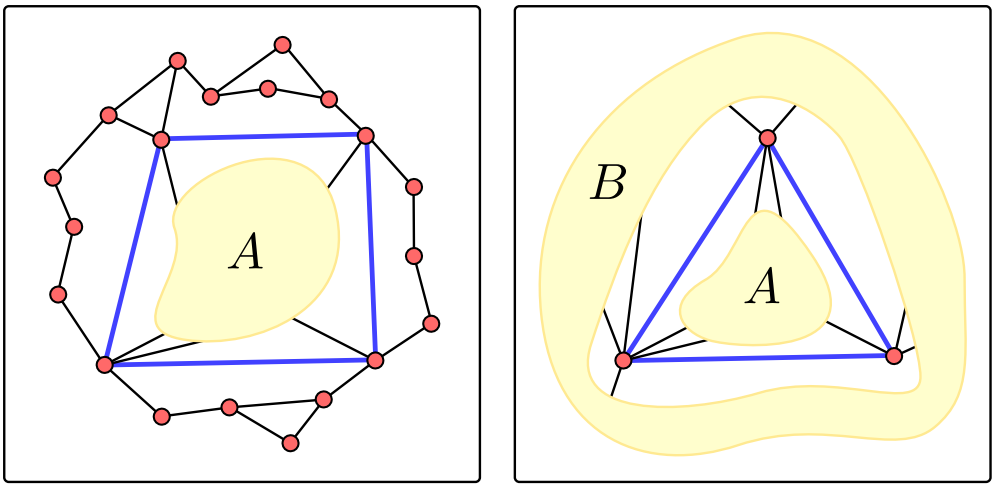
Once the backbone graph is drawn, most methods in the literature trace the cluster curves around the graph elements and eventually optimise them in a post-processing phase. In our method, the optimisation of the curves is an integral part of the drawing procedure, allowing to greatly simplify the curve tracing.

In fact, when generating the grid graph we will not make any effort to produce a decent approximation of the final diagram, but we will only convert the zone graph elements into regions of the plane. This is done for two reasons. First, a well devised



(a) (b)

Figure 5.5: Generation of non-equivalent embeddings for a zone graph. We consider a zone graph formed by a tree rooted on the vertex of a triangle. **(a)** The first kind of embedding encloses the tree in the triangle. In this case, the tree assumes the shape of the triangle and the triangle's edges are very stretched. **(b)** The second kind of embedding places the tree on the external face. As the previous constraints are absent, the spacing of the nodes and the angular resolution are much improved.



(a) (b)

Figure 5.6: Problems related to the choice of the embedding. **(a)** Although it is rather simple to force the face with higher number of edges to be on the outer face, this does not solve the problem. A small polygon including a large portion of the graph could be present just inside the outer face. **(b)** To choose the best planar graph, it would be necessary to investigate all possible cycles and the portions A and B in which the graph would be decomposed.

optimisation phase can produce layouts of high aesthetics even when starting from a grid graph of this kind. Second, a more complex routing system would probably present similar results in the case of complex and large diagrams.

In the following sections we will first detail the generation of the grid graph and of its initial drawing. Then, we will discuss the optimisation phase, that consists of the improvement of the initial grid graph layout.

5.2.1 Grid Graph Generation

The grid graph is built on the embedding of the zone graph, by placing nodes and edges on the plane. For this reason, once the generation procedure is completed, we obtain both the grid graph and its initial drawing.

There are two main aims in the construction of the gird graph:

- *Transform the zone graph nodes in actual regions of the plane.* Since we want to include the original elements in the final drawing, and since the zone regions should have a dimension sufficiently large to nicely fit their content, we insert the original graph nodes in the drawing before the optimisation phase. To do so, we need to reserve a portion of the plane for each zone, in correspondence to its current position in the zone graph.

- *Preserve the zone region adjacency.* Since the zone graph edges indicate adjacency relations between the zone graph regions, we need to enforce these relations in the portions of the plane reserved to each zone.

The easiest way to obtain these aims is to directly convert the nodes and edges of the zone graph into portions of the plane.

Enclosing the Zone Graph Nodes

The grid graph nodes are placed on the plane both around the zone graph nodes, and around the zone graph edges.

The grid graph nodes placed around the zone graph node u^z lie on a circle centred on the latter. We define a radius r that corresponds to a third of the minimal node-node and node-edge distance in the zone graph. Such a measure for r ensures that by placing a circle of radius r centred on each grid graph node, we do not have any intersection between the circles.

The circle centred in the zone graph node u^z is cut into circular arcs by the zone graph edges incident to u^z. We place the grid graph nodes on the arcs respecting the following conditions:

- each arc has at least one grid graph node,

- the central angles between consecutive nodes in the whole circle are never greater than $2\pi/3$,

- the central angles between consecutive nodes in the same arc have constant amplitude k, and the central angles between the external grid graph nodes and the relative arc extremities are equal to half this measure (see figure 5.7b).

The grid graph edges are instead inserted between consecutive nodes in the same arc, or in other words, we insert edges between consecutive nodes on the circle only when they do not cross zone graph edges (see figure 5.7c).

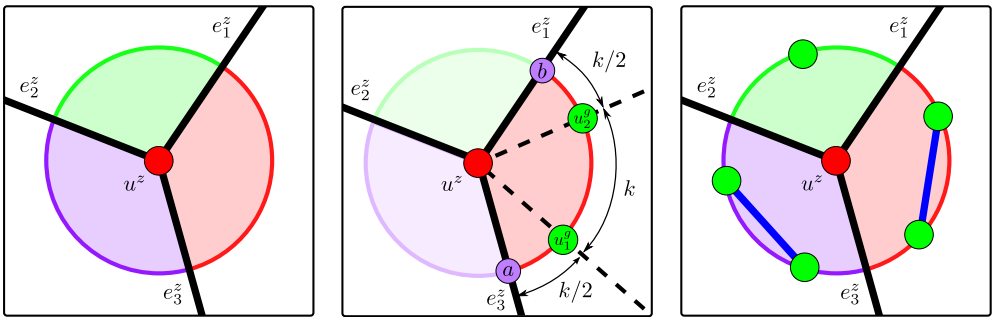


Figure 5.7: Construction of the grid graph around the zone graph nodes. **(a)** A circle of radius r, centred on a zone graph node u^z, is split into arcs by the edges incident to u^z. The arcs are identified by different colours. **(b)** Let us consider the red arc. The grid graph nodes are placed so that the central angles respect the given proportions. **(c)** The grid graph edges are added between the consecutive grid graph nodes on the same arc.

The pseudo-code of of this first step of the grid graph generation is reported in algorithm 5.3 on page 91.

Circle Radius. We compute the circle radius r as

$$r = \min \left(\min_{\substack{u,v \in V^z \\ u \neq v}} \|\boldsymbol{p}_u - \boldsymbol{p}_v\|, \min_{\substack{u \in V^z \\ e \in E^z \\ u \neq s(e), t(e)}} \|\boldsymbol{p}_u - \boldsymbol{p}_p\| \right) / 3$$

where p is the projection of u in the line of e, and where $\boldsymbol{p}_x$ is the position of the point x (node or projection) on the plane.

Positioning of the Grid Graph Nodes (Around Zone Graph Nodes). Let us consider a zone graph node $u^z \in V^z$ and the circle of radius r centred on it. We split the circle into arcs according to the incident edges of u^z (see figure 5.7a).

Let a and b be the extremities of one of these arcs. In this arc, we insert a number of grid graph nodes equal to

$$n = \left\lceil \frac{\angle ab}{2\pi/3} \right\rceil$$

where $\angle ab$ is the measure of the central angle $\widehat{au^zb}$.

The grid graph nodes $u_1^g \dots u_n^g$ are inserted on the arc so that the node u_i^g has a central angle from the beginning of the arc equal to

$$\angle a u_i^g \leftarrow \frac{\angle ab}{n} \cdot (i - 0.5)$$

This gives constant amplitude between the grid graph nodes, and half of this measure between a and the first node, and between the last node and b (see figure 5.7b).

The positioning of the grid graph nodes is computed for each arc of the circle individually. In the case of isolated zone graph nodes, since there are no incident zone graph edges to enforce the position of the grid graph nodes, we insert three grid graph nodes on the circle with the only condition of placing them evenly spaced.

Correctness. We verify that the formulae respect the previous conditions. We insert at least one node per arc, since $n \geq 1$ for every arc of positive amplitude. Let $k = \angle ab/n$. Since $\angle aa = 0$ and $\angle ab = nk$, we obtain that $\angle a u_1^g = \angle u_n^g b = k/2$ and that $\angle u_i^g u_{i+1}^g = k$. Also, since $k < 2\pi/3$, the central angle between consecutive nodes is smaller than $2\pi/3$ both when the nodes belong to the same arc and when they do not.

Insertion of the Grid Graph Edges (Around Zone Graph Nodes). We insert a grid graph edge between each consecutive pair of grid graph nodes $u_i^g u_{i+1}^g$ that belong to the same arc. No edges are inserted between nodes on different arcs (see figure 5.7c).

In the case of isolated zone graph nodes, the edges are inserted between each pair of grid graph nodes, creating a triangle.

Enclosing the Zone Graph Edges

Let us consider a zone graph edge $e^z = [u_1^z u_2^z]$. Since the edge e^z splits the circles of both the extremities into arcs, it is possible to detect the first clockwise and anticlockwise grid graph nodes from the edge in each circle. For node u_1^z, we denote the first clockwise one with u_{1c}^g and the first anticlockwise with u_{1a}^g. For u_2^z, the two are respectively u_{2c}^g and u_{2a}^g.

For each zone graph edge e^z we insert two grid graph nodes and five grid graph edges. Let us consider the segments $u_{1c}^g u_{2a}^g$ and $u_{1a}^g v_{2c}^g$. We insert the two grid graph nodes v_1^g and v_2^g in the middle points of the first and second segment. The five grid graph edges connect the extremities of the segments to their middle points, and the middle points between them (see figure 5.8).

The insertion of these elements completes the determination of the zone regions, since the boundary is well constructed and the zones do not overlap with each other. The pseudo-code of this last step of the grid graph generation is reported in algorithm 5.4.

Positioning of the Grid Graph Nodes (Around Zone Graph Edges). Let $e^z = [u_1^z u_2^z]$ be a zone graph edge. In the circle of u_1^z, we identify with u_{1c}^z the first grid graph node in clockwise order from the edge and with u_{1a}^g the first anticlockwise one. Similarly for u_2^z.

We place the grid graph node v_1^g in the middle point of the segment $u_{1c}^g u_{2a}^g$ and the grid graph node v_2^g in the middle point of the segment $u_{1a}^g u_{2c}^g$.

Insertion of the Grid Graph Edges (Around Zone Graph Edges). For each zone graph edge e^z, we insert the grid graph edges $[u_{1c}^g, v_1^g]$, $[v_1^g, u_{2a}^g]$, $[u_{1a}^g, v_2^g]$, $[v_2^g, u_{2c}^g]$ and $[v_1^g, v_2^g]$.

Algorithm 5.3 Grid graph construction around zone graph nodes

▷ Radius Computation

$r \leftarrow +\infty$
for all $u_1, u_2 \in V^z,\ u_1 \neq u_2$ **do**
  $r \leftarrow \min(r,\ \|\boldsymbol{p}_{u_1} - \boldsymbol{p}_{u_2}\|/3)$
for all $u \in V^z,\ e \in E^z,\ u \neq s(e),\ u \neq t(e)$ **do**
  $p \leftarrow$ projection of u in e
  if $p \in e$ **then**
    $r \leftarrow \min(r,\ \|\boldsymbol{p}_u - \boldsymbol{p}_p\|/3)$

for all $u^z \in V^z$ **do**

▷ Edge List Creation

  $[e_0 \dots e_a] \leftarrow$ incident edges of u^z in order
  if $[e_0 \dots e_a] = \emptyset$ **then**
    $e_0 \leftarrow$ dummy edge
  $e_{a+1} \leftarrow e_0$

▷ Element Insertion

  for $i \leftarrow 0$ **to** a **do**
    arcStart $\leftarrow$ angle of e_i
    arcAmpl $\leftarrow \angle e_i e_{i+1}$
    $n \leftarrow \lceil 3 \cdot \text{arcAmpl}/2\pi \rceil$
    for $j \leftarrow 1$ **to** n **do**
      ang $\leftarrow$ arcStart $+$ (arcAmpl$/n$) $\cdot\ (i - 0.5)$
      insert the node u^g_j at distance r and the angle ang from u^z
      if $j > 2$ **then**
        insert an edge between u^g_{j-1} and u^g_j

Algorithm 5.4 Grid graph construction around zone graph edges

for all $e^z \in E^z$ **do**

▷ Node Insertion

  $u^z_1 \leftarrow s(e^z)$ $u^z_2 \leftarrow t(e^z)$
  $u^g_{1c} \leftarrow$ firstClockNode(u^z_1)
  $u^g_{2c} \leftarrow$ firstClockNode(u^z_2)
  $u^g_{1a} \leftarrow$ firstAnticlockNode(u^z_1)
  $u^g_{2a} \leftarrow$ firstAnticlockNode(u^z_2)
  Insert node v^g_1 as midpoint of u^g_{1c} and u^g_{2a}
  Insert node v^g_2 as midpoint of u^g_{1a} and u^g_{2c}

▷ Edge Insertion

  Insert edge between v^g_1 and v^g_2
  Insert edges between u^g_{1c}, u^g_{1a}, u^g_{1c}, u^g_{1a} and the relative midpoint

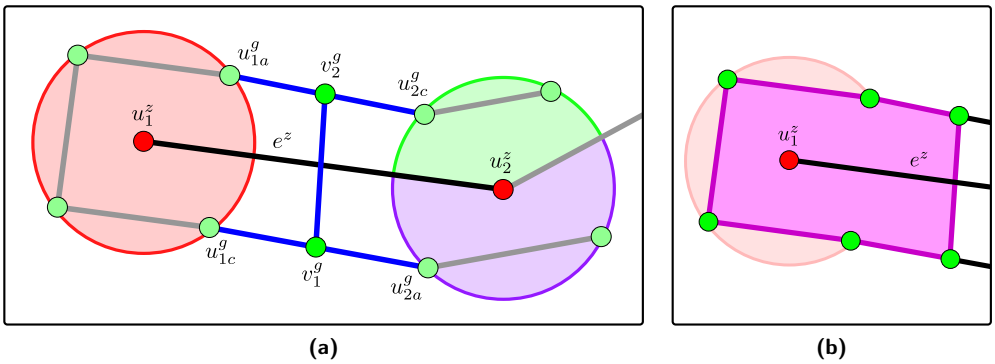


Figure 5.8: Construction of the grid graph around the zone graph edges. **(a)** For each zone graph edge, we identify the first clockwise and anticlockwise nodes on the two circles. We place two new grid graph nodes on the middle points of the segments $u^g_{1c}u^g_{2a}$ and $u^g_{1a}v^g_{2c}$, and we insert five new grid graph edges. **(b)** At the end of this procedure, we obtain a face that encloses the zone graph node. This is the initial drawing of the zone region associated with the node.

Correctness. The area dedicated to a zone graph node u^z corresponds to a face bounded by the grid graph elements inserted around u^z and around its incident edges. Since this area is at a maximum distance of r from the zone graph elements, and since such distance is determined as a third of the node-node and node-edge distances in G^z, there are no overlaps between the zone regions in the drawing.

5.2.2 Grid Graph Improvement

In this stage, we perform a few modifications of the current grid graph and we optimise its initial layout. The graph that results from this phase contains all the information about the positioning of both the cluster elements and the cluster curves.

Grid Graph Modifications

As a first modification, we substitute all the chains (see section 3.1.5) of the grid graph with polyline edges, so that the edge has the same extremities as the chain, and bends corresponding to the positions of the internal nodes of the chain.

Then, we insert the original graph elements in the relative zone regions. To ensure that the original graph nodes are placed inside the zone region boundaries detected at the previous stage, we position them in a limited area around the position of the zone graph node.

Since we place the grid graph nodes so that the maximal central angle between consecutive ones is $2\pi/3$, the minimal distance between the centre of the circle and a grid graph edge is equal to the apothem of an equilateral triangle. Since smaller central angles induce larger distances, and since the ratio between the circumradius and the apothem of an equilateral triangle is $\cos \pi/3 = 2$, any points in a circle with radius $r/2$, with r defined above, is surely inside the zone region (see figure 5.9).

Finally, before proceeding with the optimisation, we remove the zone graph from the drawing.

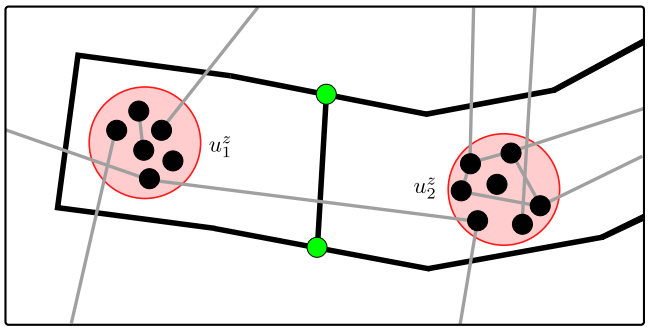


Figure 5.9: Modifications of the grid graph before the layout optimisation. First, the chains are replaced by polyline edges having bends in correspondence to the chain nodes. Then, the original graph nodes are placed in the respective zone regions by inserting them in random positions inside a circle with radius $r/2$. Finally, the original graph edges are also inserted.

Conversion of the Grid Graph Chains into Polyline Edges. Each chain $u_0 e_1 u_1 \dots e_l u_l$ in the grid graph is substituted by a polyline edge $[u_0, u_l]$ that has a bend in correspondence with each u_i, with $0 < i < l$.

Insertion of the Original Graph Elements. Let u^z be a node of the zone graph. We insert the original graph nodes of the zone u^z by placing them at random positions in the circle centred in u^z and having radius $r/2$ (with r defined in section 5.2.1).

Once the nodes are all inserted we also add the original graph edges.

Layout Improvement

The current drawing, formed by both the grid and the original graph, is improved by applying `ImPrEd` a second time. This time, we use some additional features that we introduced in `ImPrEd` to ensure a greater control over the generation of Euler diagrams.

The set of crossable edges Cr can be used to remove the non-crossing conditions to some of the graph edges. The set of flexible edges Fl can be used to enable some edges to be bent. The weight of the elements increases or decreases the importance of the nodes or edges in the determination of the final drawing.

Algorithm Parameters. Since we want the grid graph elements to have higher influence on the optimisation of the graph layout, we assign a higher weight to them. We set the original graph edges as crossable, since we do not want them to obstruct the movement of the nodes inside the zone regions, and we lower their weight. The grid graph edges are instead marked as flexible, since we want them to assume shapes that better fits the drawing.

The parameters passed to `ImPrEd` are

$$\delta = 5 \qquad \gamma = 4 \qquad \text{iter} = 300 \qquad \text{Cr} = E^o \qquad \text{Fl} = E^g$$

$$\mathcal{W}(V^g) = \mathcal{W}(E^g) = 2 \qquad \mathcal{W}(E^o) = 0.1$$

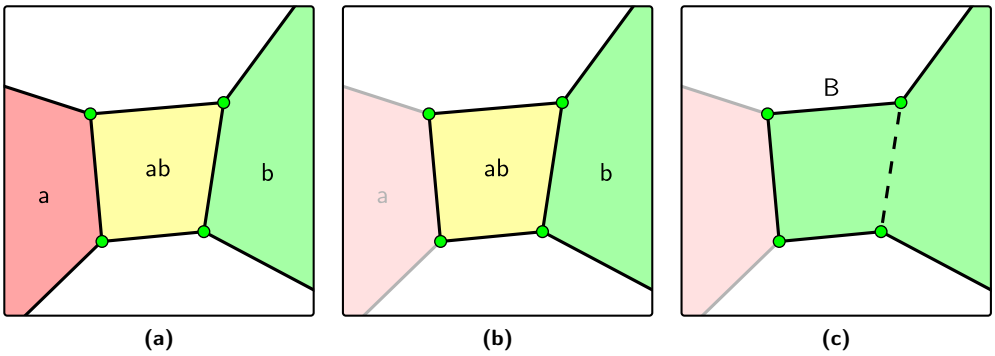


Figure 5.10: Identification of the cluster curves. **(a)** The drawing obtained with the grid graph optimisation. The cells of the grid graph are associated with their zone labels. **(b)** To identify the cluster curves of cluster B, we take into consideration the boundaries of the grid graph cells associated with the zones $\mathcal{A}(\mathsf{B})$. **(c)** The edges dividing two of such cells (such as the dashed edge) are removed. The remaining edges compose the cluster curves of B.

5.3 Depiction of the Cluster Regions

At this point, the cluster curves can easily be identified by merging the boundaries of the zone regions associated with the same cluster. In the case of adjacent zones belonging to the same cluster, the shared boundaries are not considered, so that the resulting cluster curve contains the zone regions without dividing them (see figure 5.10).

In order to improve the aesthetics and the readability of the final diagrams, we apply several graphical techniques to the depiction of the cluster regions.

First, we use smooth cluster curves. The current grid graph uses polyline edges to bound the cluster and the zone regions. We need to convert these polyline boundaries into smooth curves, while ensuring that this transformation does not generate unwanted overlaps.

Second, we compute a map of colours to assign to the clusters. This map aims to minimise the colours used in the drawing, while not assigning the same colour to overlapping sets.

Finally, we apply semi-transparent patterns to the cluster regions. The patterns help distinguish the clusters involved in an intersection, since the blending of the colour is often not sufficient to discern which and how many clusters overlap in the same region.

5.3.1 Smooth Cluster Curves

To convert the current polyline edges into smooth boundaries, we transform each straight-line edge, or each segment of a polyline edge, into a cubic Bézier curve.

In doing so, we need to consider a couple of important issues. First, since each edge will be individually converted, we need to ensure that the curves merge smoothly. Second, since the transformation modifies the shape of the current boundary, we need to ensure that the diagram is not altered by this operation.

We can ensure that these conditions are respected by using the properties of cubic Bézier curves.

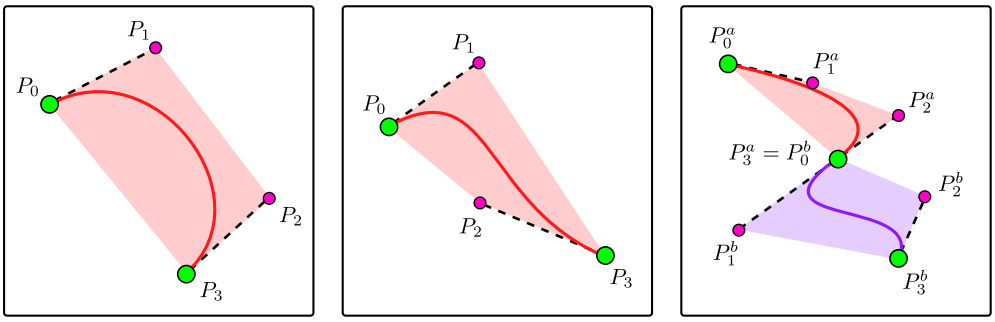


Figure 5.11: Examples of Bézier curves. The control points are marked with circles. In particular, P_0 and P_3, the extremities of the curves, are depicted by green circles, while P_1 and P_2, that are not generally crossed by the curves, are depicted by purple circles. The shaded area is the convex hull of the control points, that fully contains the curve. **(a)** A first example, in which the control polygon corresponds to the convex hull. **(b)** A second example, in which there is no correspondence between the control polygon and the hull. **(c)** A smooth connection between different curves can be obtained by overlapping P_3^a and P_0^b, and by making them collinear to the control points P_2^a and P_1^b.

Cubic Bézier Curves. A Bézier curve $\mathcal{C}$ is a parametric curve described by a tuple of points, called control points. Cubic Bézier curves have four control points, P_0, P_1, P_2 and P_3.

Properties of Cubic Bézier Curves. Cubic Bézier curves have the following properties:

1. The polygon formed by the edges P_0P_1, P_1P_2, P_2P_3, P_3P_0, is called a *control polygon*. Its convex hull completely contains the curve.
2. The curve starts in P_0 and ends in P_3.
3. The beginning of the curve is tangent to P_0P_1. The end of the curve is tangent to P_2P_3.

In particular, we will use the first property to ensure that the drawing will not be altered, and the last two to obtain a smooth connection of different curves. Examples of such properties are shown in figure 5.11.

Smooth Connection of Different Curves

For each curve, the start and the end points P_0 and P_3 are provided by the grid graph, as they correspond to the extremities of the edge. Thus, we only need to compute the points P_1 and P_2.

Let $\mathcal{C}_a$ and $\mathcal{C}_b$ be two cubic Bézier curves, and let P_i^a and P_i^b be their control points. According to the third property, it is sufficient to enforce $P_3^a = P_0^b$ to make them join. However, to avoid sudden changes of direction in correspondence with the joint, we must also ensure that the end of the first curve and the beginning of the

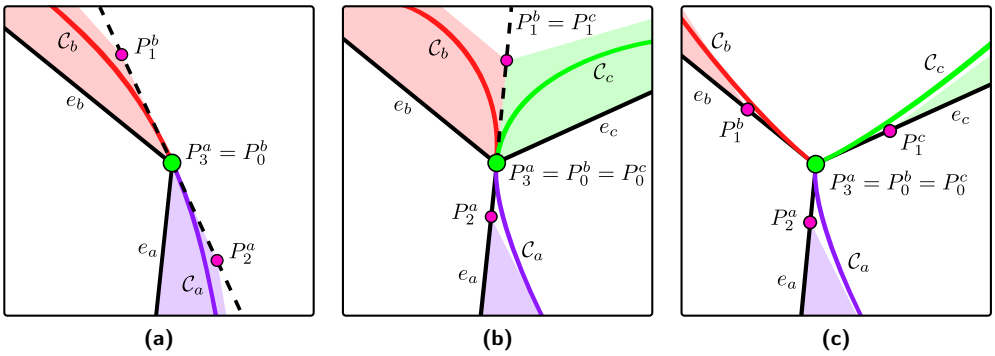


Figure 5.12: Depiction of the cluster curves, distinguished by the type of junction. At this stage of the computation, we only identify the line on which the control points are placed. The distance from the junction, and with that the actual position of the control points, is decided in the next step of the procedure. **(a)** Basic junction. The control points P_a^2 and P_b^1 are placed along the perpendicular of the bisector of the angle formed by e_a and e_b, passing through the junction. **(b)** Spring junction. If e_a is the edge shared by all the cluster boundaries passing through the junction, we place P_2^a along the edge. The remaining control points are placed along the line of e_a, on the opposite side of the junction. **(c)** Complex junction. Since we cannot obtain smooth and consistent boundaries for all the clusters in the junction, we place the control points along the respective edges.

second have the same tangent. Therefore, we need to place the control points P_2^a and P_1^b so that the segments $P_2^a P_3^a$ and $P_0^b P_1^b$ are collinear (see figure 5.11c).

The previous consideration implies that the position of the points P_1 and P_2 cannot be computed individually for each curve, since P_1 depends on the previous Bézier curve and P_2 depends on the next. For this reason, we will discuss how to compute P_2^a, P_1^b for adjacent curves, rather than P_1 and P_2 for the same one.

Since P_2^a, P_1^b and $P_3^a = P_0^b$ must be collinear, we first need to pick a line passing through the junction point $P_3^a = P_0^b$ to detect the remaining control points. We distinguish three cases, that differ in the way the cluster curves intersect in a junction point.

Basic Junction. If the junction point is a grid graph node with degree two, meaning that only two curves merge at this point, we have a basic junction. In this case, if e_a is the edge that corresponds to the first curve $\mathcal{C}_a$ and e_b is the edge corresponding to the second one $\mathcal{C}_b$, we pick the line that passes though the junction, and that is perpendicular to the bisector of the angle formed by e_a and e_b (see figure 5.12a).

Spring Junction. If the junction point is a grid graph node with degree greater than two, and if there is an edge used by all the cluster curves passing through the junction, we have a spring junction. In this case, for each cluster curve we pick the line that passes through the junction and is collinear to the edge shared by all cluster curves (see figure 5.12b).

Complex Junction. When the two previous cases do not apply, we have a complex junction. In this case, it is not possible to obtain smooth and consistent cluster curves, since we would break the concurrency of some cluster regions and generate undesired

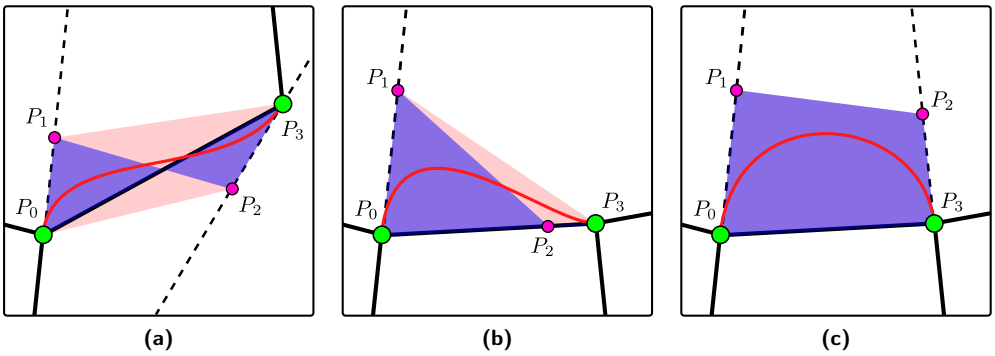


Figure 5.13: Different configurations of convex hulls and control polygons. The red shaded area is the convex hull, while the blue area is the control polygon. As the curve is fully contained in the convex hull, we need to ensure that the convex hull of different edges do not overlap. **(a)** Non simple control polygon. The convex hull is a quadrilateral that does not correspond to the control polygon. **(b)** Simple but not convex control polygon. The convex hull is a triangle that includes the edge. **(c)** Convex control polygon. The convex hull corresponds to the control polygon.

new zones. Therefore, we place the control point of each curve directly on its edge (see figure 5.12c).

Ensuring that the Diagram is Not Altered

At this point of the computation, we have only obtained the lines on which the control points lie. We now proceed with the computation of the distances of the control points from their junction, that allows us to fully identify their position on the plane.

The distance at which we place the control points is crucial for the correctness and the appeal of the diagram. If the distance from the junction is too low, the cluster curves will not appear smooth and regular, as we merely round the corners of the current polygons. If the distance from the junction is too high, we might generate overlaps between the cluster curves that were not originally in the drawing. In addition, too high distances might also produce unnatural, bulging cluster shapes.

Binding the Maximal Distance to Enforce Correctness. Since assigning too high distances risks altering the diagrams, we must compute the maximal values allowed for each junction and for the given directions. These values must ensure that the convex hull of the control points of each curve do not overlap with that of another curve.

We enforce a first condition to prevent the generation of bulging curves and to simplify the characterisation of the convex hulls. We bound the distance of P_1 from P_0, and of P_2 from P_3 not to be greater than a third of the edge length.

Under this condition, the convex hull of the control points of an edge can assume three configurations.

- If the control polygon is not simple, then its convex hull is a quadrilateral that does not correspond to the control polygon. For the given restrictions on the length of P_0P_1 and P_2P_3, it is necessarily formed by the edges P_0P_1, P_2P_3, P_0P_2 and P_1P_3 (see figure 5.13a).

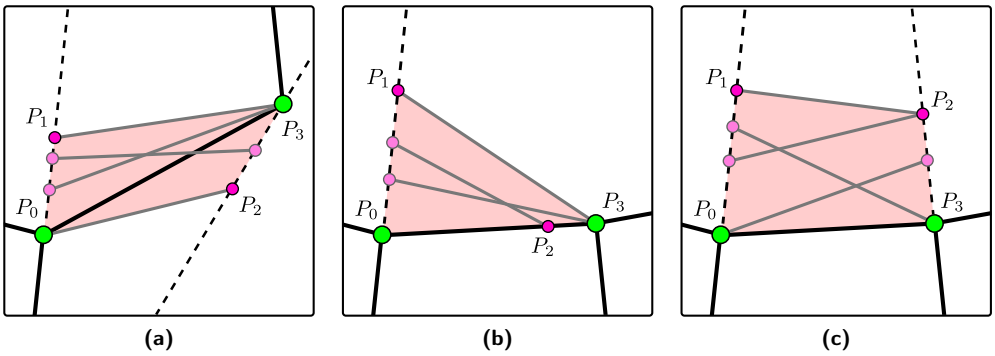


Figure 5.14: Analogies of the maximal node movement of `PrEd` and `ImPrEd` with the computation of the maximal distance of the control points from their junction. The grey lines are possible future positions for the edge considered. The region that contains any future position of the edge exactly corresponds to the convex hull of the control points of a Bézier curve (see figure 5.13). **(a)** First case. **(b)** Second case. **(c)** Third case.

- If the control polygon is simple but not convex, the convex hull will be a triangle. This triangle necessarily has P_0 and P_3 as vertices, since triangles that include such points and have the given restriction on the length of P_0P_1 and P_2P_3, would not satisfy the triangle inequality (see figure 5.13b).
- If the control polygon is convex, and therefore simple, the convex hull corresponds to it (see figure 5.13c).

Computation of the Maximal Distance. The problem of limiting the length of the edges P_0P_1 and P_2P_3 to ensure the absence of overlaps between the convex hulls of the Bézier curves is very similar to a problem addressed in `PrEd` and `ImPrEd`. In the algorithms, we compute the maximal displacement of the extremities of an edge to ensure that it will not cross another edge when moving.

This computation is performed for every direction in which the extremities of an edge can move. However, for the current purposes we can restrict the reasoning to only the movement of the edge extremities along a given direction. Let us indicate the current position of the edge extremities with P_0 and P_3, and a candidate future position for them with P_1 and P_2.

Since at the end of the computation each node is individually placed in a position between the current and the candidate future one, the algorithm must ensure that the region containing any possible future position of an edge does not overlap with that of another edge. As figure 5.14 shows, this region is determined by the points P_0, P_1, P_2 and P_3 and corresponds to the convex hulls of the Bézier curves described above.

For this reason, we can use `ImPrEd`'s computation of the maximal node movement $\mathcal{M}_{P_0}$ and $\mathcal{M}_{P_3}$ to compute the maximal distance allowed for the control points P_1 and P_2, along the direction computed at the previous step of the computation. This procedure will be explained in detail in the next chapter, and in particular in sections 6.1.4 and 6.2.6.

The final distances of P_1 and P_2 from their junction will be calculated as

$$\overline{P_0P_1} = \min\left(\frac{\overline{P_0P_3}}{3},\ \mathcal{M}_{P_0}\right) \qquad \overline{P_2P_3} = \min\left(\frac{\overline{P_0P_3}}{3},\ \mathcal{M}_{P_3}\right)$$

5.3.2 Assignment of the Cluster Colours

Humans have trouble distinguishing more than about six to eight colours [111] in a complex diagram. If we use transparency and allow colours to mix at each intersection, we can see why minimising the number of colours used becomes a priority. On the other hand, different colours should be used for different classes in the diagram.

To circumvent this problem, we generate another graph where each node is a cluster $s \in S$ and an edge indicates that two clusters overlap. We run the node colouring heuristic of Welsh and Powell [114] on this graph, to obtain a small number of colours to assign to the clusters. Assuming that the graph can be coloured with less than c colours and we have c colours at our disposal, it is guaranteed that overlapping cluster regions will have different colours assigned.

Currently, we use eight colours, $c = 8$. The colour chosen for a cluster s is $(\mathrm{idx}(s) \mod c)$, where $\mathrm{idx}(s)$ is the index provided by the colouring algorithm. Therefore, in very complex diagrams, the same colour could be associated with overlapping clusters. This should not be a problem, since we will also use textures to discriminate the cluster regions.

5.3.3 Application of Textures

To help the user discerning and identifying the cluster regions, we also use textures as previously done by Byelas and Telea [9]. This is especially helpful with complex intersections, since the resulting blended colour usually provides little information about the clusters that generate it.

Moreover, the use of textures also compensates the low number of colours we decided to use. Let t be the number of textures, currently fixed to seven. The texture assigned to the cluster s is $(\mathrm{idx}(s) \mod t)$, where $\mathrm{idx}(s)$ is again the index provided by the colouring algorithm. Since c and t are coprime, we can assign a different combination texture/colour up to an index of 56, that is reachable only by extremely complex clusterings.

Chapter 6

Improvement of a Graph Layout

This chapter introduces two algorithms that aim to improve the layout of an existing graph drawing while preserving some of its characteristics.

The first algorithm is called `PrEd`, and has been developed by Bertault [6] in 2000. The second algorithm, `ImPrEd`, is an improved version of the first. We developed it to increase `PrEd`'s performance and functionality, in particular with respect to the generation of Euler diagrams.

In the first section we will present the original algorithm, the details of its implementation and its advantages and disadvantages.

In the second section, we discuss `ImPrEd`. While presenting its new features, we put particular attention on highlighting the conceptual and technical changes they induce in `PrEd` and its behaviour.

Finally, in the closing section, we present the results of the comparative tests we ran to evaluate performance, drawing quality and reliability of `ImPrEd` with respect to `PrEd`.

6.1 `PrEd`

`PrEd` is a force-directed algorithm designed to improve an existing graph drawing while preserving its edge crossing properties. Preserving the edge crossing properties implies that two edges will cross in the final drawing if and only if they crossed in the original one, or in other words, that no new crossings will occur and no existing crossings will be undone.

To obtain this result, the algorithm verifies that the forces acting on a node do not displace it in such a way that the current edge crossings are altered. When this happens, the length of the node movement is restricted to a safe distance, eventually very close to zero.

The preservation of the edge crossing properties makes `PrEd` the ideal algorithm to optimise plane graphs, since we typically want to improve their layout without introducing edge crossings. However, the technique used to enforce this characteristic induces a slightly more restrictive behaviour: not only are edge crossings not added or removed, but also nodes cannot cross edges when moving.

This leads to a couple of interesting side effects. First, `PrEd` does not only grant the absence of crossings when improving a plane graph, but also preserves its embedding. Second, and more important, `PrEd` can be used to improve Euler diagrams, since the nodes enclosed by edges will not be able to escape that region.

6.1.1 Input Parameters

PrEd has one implicit parameter, the initial graph drawing, and three explicit parameters to control the quality and the characteristics of the desired output:

- *Initial graph drawing*: the graph drawing that we aim to optimise. The graph must be simple and the eventual direction of the edges is not taken into consideration. The drawing must follow the straight-line drawing conventions and must not contain any node-node or node-edge overlap. Although not explicitly stated by the author, the algorithm is designed to deal only with connected graphs.
- *Number of iterations*: the number of times the improvement procedure is repeated. Higher values produce better results, but induce higher running times. Typical values span from one to five hundred.
- *The optimal edge length*, δ: the distance at which repulsive and attractive forces acting on adjacent nodes balance (see figure 3.14). This can also be interpreted as the minimal desired distance between unrelated nodes.
- *The optimal node-edge distance*, γ: the minimal distance at which non-incident nodes and edges do not repel each other. This can also be interpreted as the minimal desired distance between nodes and unrelated edges.

6.1.2 The Algorithm

PrEd's algorithm retains many of the characteristics of standard force-directed algorithms (see section 3.2.3). The graph layout is iteratively optimised, meaning that the algorithm has a procedure that produces a slight improvement of the graph layout in a very short time, and that the final drawing is obtained by iterating this procedure a high number of times. Also, the procedure includes the two standard phases of a force-directed algorithm iteration: the computation of the forces acting on the nodes, and the node displacement.

The distinctive characteristics of PrEd reside in an additional phase in the procedure, the maximal movement computation, and on its effect in the node displacement. In this step, the algorithm computes a safe distance for each of the graph nodes, according to eight orientations of the maximal movement. In the node movement step, whenever the forces acting on a node would push it over the safe distance computed for a force with that orientation, the magnitude of the force is reduced to the value of the safe distance.

Algorithm 6.1 presents a pseudo-code of the PrEd algorithm, with highlights on the division into phases of the main cycle. The phases of the procedure, along with the PrEd parameters, are detailed in the following sections.

6.1.3 Force Computation

PrEd drives the node positioning using the three classical types of forces: edge attraction, node-node repulsion and node-edge repulsion (see figure 6.1).

As in previous work, the repulsive forces are quadratic with the distance, and the attractive forces are linear with the distance. This makes repulsive forces very influential at a local scale, to well separate the graph elements, and less influential at a larger scale, as the elements are already spaced enough. At the same time, attractive forces are designed to be less important than repulsive forces at a local scale, not to

Algorithm 6.1 PrEd. The algorithm consists of a cycle composed of three main phases: force computation, maximal movement computation, and node displacement.

for numberOfIterations **do**

▷ Force Computation

for all $u \in V$ **do**
  force$(u) \leftarrow 0$

for all $\{u, v\} \subseteq V$, $u \neq v$ **do**
  currentForce $\leftarrow$ node-node repulsion between u and v
  force$(u) \leftarrow$ force(u) + currentForce
  force$(v) \leftarrow$ force(v) − currentForce

for all $e \in E$ **do**
  currentForce $\leftarrow$ edge attraction between $s(e)$ and $t(e)$
  force$(s(e)) \leftarrow$ force$(s(e))$ + currentForce
  force$(t(e)) \leftarrow$ force$(t(e))$ − currentForce

for all $u \in V$, $e \in E$, $u \neq s(e) \wedge u \neq t(e)$ **do**
  currentForce $\leftarrow$ edge-node repulsion between u and e
  force$(u) \leftarrow$ force(u) + currentForce
  force$(s(e)) \leftarrow$ force$(s(e))$ − currentForce
  force$(t(e)) \leftarrow$ force$(t(e))$ − currentForce

▷ Max Movement Computation

for all $u \in V$, $i = 0 \dots 7$ **do**
  maxmove$(u, i) \leftarrow +\infty$

for all $u \in V$, $e \in E$, $u \neq s(e) \wedge u \neq t(e)$ **do**
  $p \leftarrow$ projection of u in e
  if $p \in e$ **then**
    restrict the movement of u, $s(e)$, $t(e)$, case "projection inside"
  else
    restrict the movement of u, $s(e)$, $t(e)$, case "projection outside"

▷ Node Displacement

for all $u \in V$ **do**
  $i \leftarrow$ sector where force(u) lies
  if $\|$force$(u)\| >$ maxmove(u, i) **then**
    force$(u) \leftarrow$ force(u) clamped to magnitude maxmove(u, i)
  position$(u) \leftarrow$ position(u) + force(u)

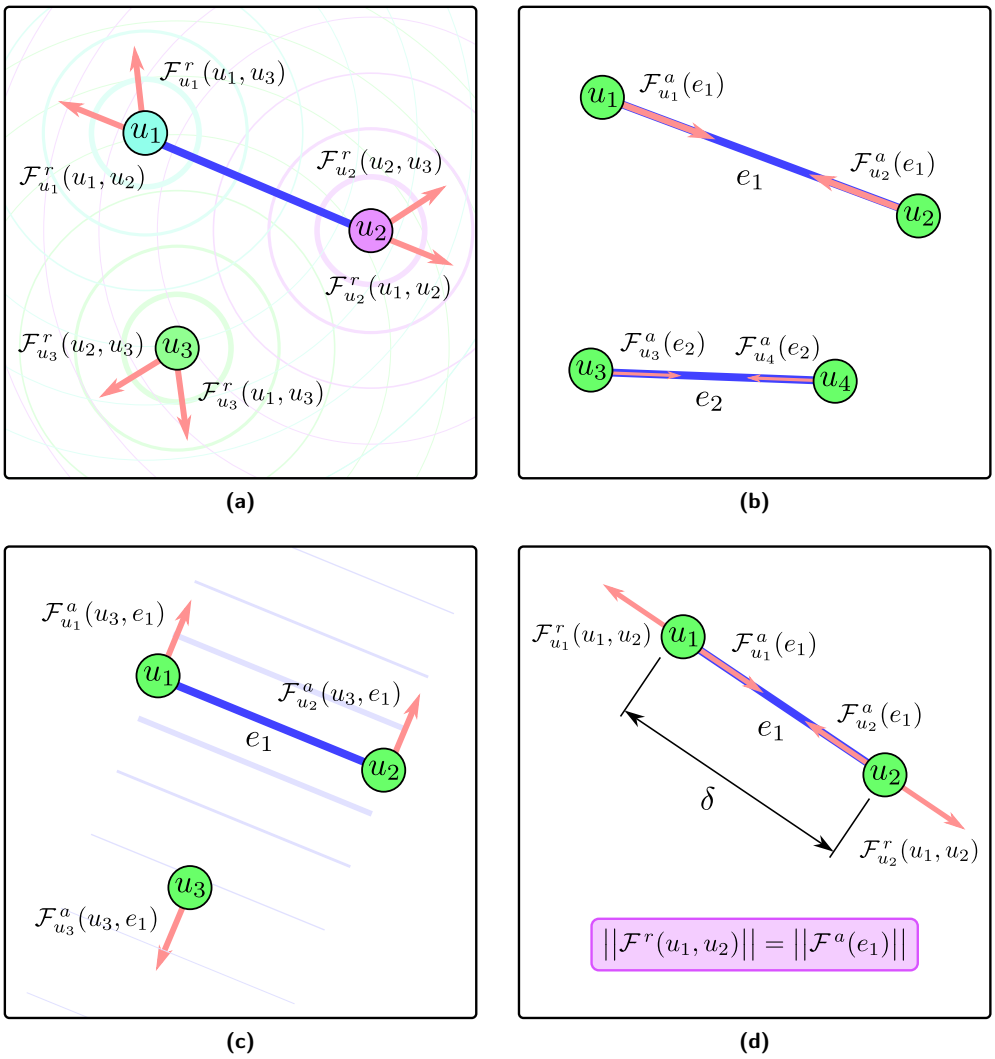


Figure 6.1: Forces considered in `PrEd`. **(a)** Node-node repulsion. **(b)** Edge attraction. **(c)** Node-edge repulsion. The force is null when the node projection is not on the edge or when the node is farther than γ form the edge. **(d)** Node-node repulsion and edge attraction balance each other at the optimal distance δ.

obstruct the element spacing, but still quite effective at high distances, making them able to pull together the extremities of edges longer than δ.

The edge attraction is computed between the extremities of each edge in the graph. The node-node repulsion is computed for every pair of different graph nodes, and the node-edge repulsion between every pair node-edge, where the node is not an extremity of the edge. In this last case, however, the force is null if the projection of the node is not on the edge, and if the distance between node and edge is greater than γ.

Position of Nodes and Edges. We indicate with $\boldsymbol{p}$ the position of the nodes and edges in the graph drawing. In particular, $\boldsymbol{p}_u$ is the vector containing the coordinates of the node u on the plane, and $\boldsymbol{p}_e$ is the segment, polyline or Jordan arch that represents the edge e in the drawing.

Node-Node Repulsion. In PrEd, the *node-node repulsion* $\mathcal{F}_u^r(u,v)$, acting on u for a pair of distinct nodes u and v is the vector

$$\mathcal{F}_u^r(u,v) = \left(\frac{\delta}{\|\boldsymbol{p}_u - \boldsymbol{p}_v\|}\right)^2 (\boldsymbol{p}_u - \boldsymbol{p}_v)$$

Since $\mathcal{F}_u^r(u,v) = -\mathcal{F}_v^r(u,v)$, the force can be computed only once for each pair of nodes and considered with opposite signs.

Edge Attraction. In PrEd, the *edge attraction* $\mathcal{F}_u^a(e)$, acting on u for an $e = [u,v]$ is the vector

$$\mathcal{F}_u^a(e) = \left(\frac{\|\boldsymbol{p}_u - \boldsymbol{p}_v\|}{\delta}\right) (\boldsymbol{p}_v - \boldsymbol{p}_u)$$

Since $\mathcal{F}_u^a(e) = -\mathcal{F}_v^a(e)$, this force also can be computed only once for each pair of nodes and considered with opposite signs.

Node-Edge Repulsion. In PrEd, the *node-edge repulsion* $\mathcal{F}_u^e(u,e)$, acting on a node u for a non-incident edge e, is the vector

$$\mathcal{F}_u^e(u,e) = \begin{cases} \dfrac{(\gamma - \|\boldsymbol{p}_u - \boldsymbol{p}_p\|)^2}{\|\boldsymbol{p}_u - \boldsymbol{p}_p\|}(\boldsymbol{p}_u - \boldsymbol{p}_p) & \text{if } p \in \boldsymbol{p}_e \text{ and } \|\boldsymbol{p}_u - \boldsymbol{p}_p\| < \gamma \\ \mathbf{0} & \text{otherwise} \end{cases}$$

where p is the projection of the node u onto the line defined by the edge e. For $e = [v_1, v_2]$, the forces acting on the edge extremities are

$$\mathcal{F}_{v_1}^e(u,e) = \mathcal{F}_{v_2}^e(u,e) = -\mathcal{F}_u^e(u,e)$$

Resulting Force. In PrEd, the resulting force $\mathcal{F}_u$ acting on node u is computed as the sum of all forces acting on the node:

$$\mathcal{F}_u = \sum_{\substack{v \in V \\ v \neq u}} \mathcal{F}_u^r(u,v) + \sum_{[u,v] \in E} \mathcal{F}_u^a([u,v]) + \sum_{\substack{e \in E \\ u \neq s(e) \\ u \neq t(e)}} \mathcal{F}_u^e(u,e) + \sum_{\substack{v_1 \in V \\ [u,v_2] \in E \\ v_1 \neq u \\ v_1 \neq v_2}} \mathcal{F}_u^e(v_1,[u,v_2])$$

6.1.4 Maximal Movement Computation

For each node, PrEd identifies a region where the node can move without crossing any edge. Avoiding crossings between nodes and edges in the movement phase is, in fact, a sufficient condition to preserve the edge crossing properties [6].

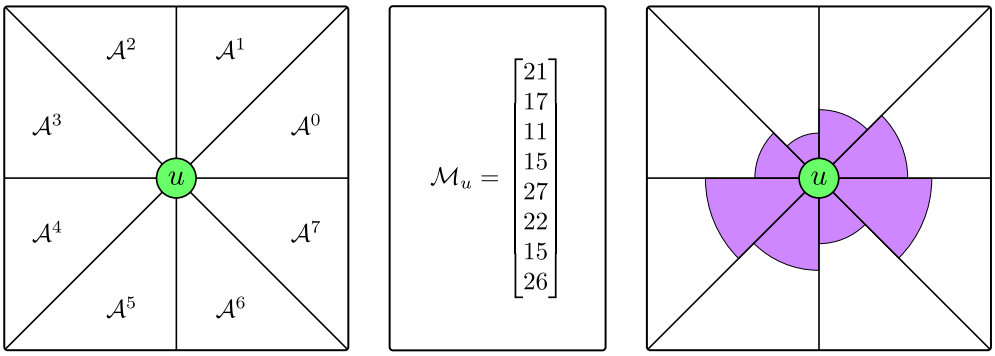


Figure 6.2: Maximal movement and movement region in `PrEd`. **(a)** Subdivision of the plane into angles, considering u as the angles' vertex. **(b)** An example of a maximal movement vector. **(c)** The movement region of u, according to the previous maximal movement vector.

The region of movement of a node u is computed considering eight orientations that the force $\mathcal{F}_u$ might assume. The plane is divided into eight angles $\mathcal{A}^i$ of equal amplitude having vertex in u. For each of these angles, `PrEd` identifies a radius $\mathcal{M}_u^i$ that represents the maximal distance that u can cover when moving inside $\mathcal{A}^i$. As a result, the movement region will be composed of eight sectors with central angle of $\pi/4$, typically with different radius, and with vertex in u (see figure 6.2).

To compute the movement region, `PrEd` takes into consideration every node-edge pair (u, e), where the node is not an extremity of the edge. Based on the reciprocal position, the sectors of both u and the extremities of e are reduced to a safe value. Since this value grants that u will not cross e, and since this control is applied to all possible node-edge pairs in the graph, the edge crossing properties are preserved in the whole graph.

The formulae for the computation of the sector radii consider two different cases: the case in which the projection of the node u on the line defined by the edge e is part of e, and the case in which the projection is external to e.

Maximal Movement and Movement Region. Let $\mathcal{A} = (\mathcal{A}^0 \dots \mathcal{A}^7)$ be a collection of angular intervals, where

$$\mathcal{A}^i = [\ i\pi/4,\ (i+1)\pi/4\ )$$

Let the vector of *maximal movement* $\mathcal{M}_u = (\mathcal{M}_u^0 \dots \mathcal{M}_u^7)$, associated with the node u, be a collection of eight real positive numbers. The *movement region* for node u is a region composed of eight sectors, having each one vertex in u, an angle that covers the interval $\mathcal{A}^i$ and radius $\mathcal{M}_u^i$, for $i = 0 \dots 7$.

Maximal Movement Computation. At the beginning of the maximal movement computation, $\mathcal{M}_u$ is set to ∞ for each node in the graph.

For each node-edge pair (u, e), where u is not an extremity of e, we perform the *movement restriction* operation. Let p be the projection of u onto the line described by e. We consider the case "projection inside" if p lies on the edge e, and the case "projection outside" otherwise (see figure 6.3).

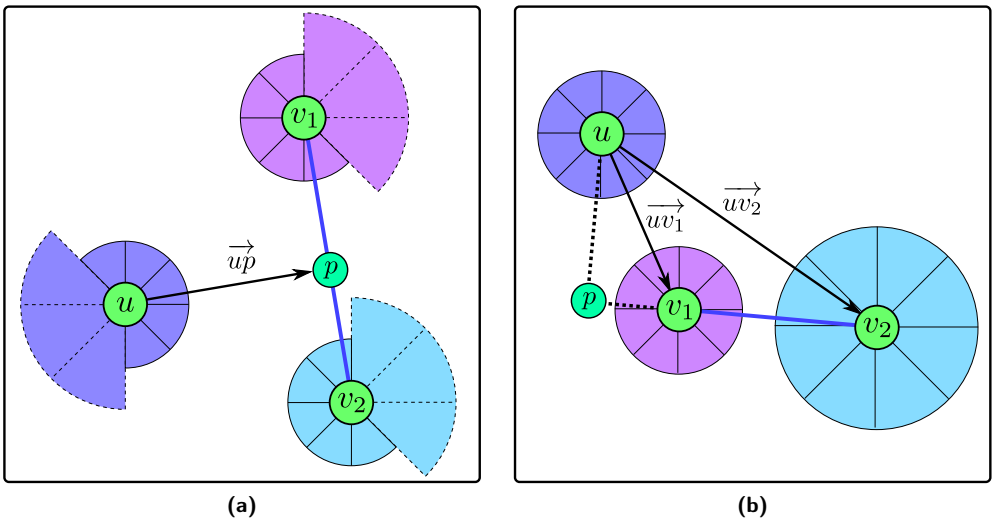


Figure 6.3: Restriction of the node movement in PrEd. **(a)** Case "projection inside", as p lies on the edge. The dashed sectors are not modified by the restriction operation. **(b)** Case "projection outside", as p lies outside the edge. In this case, all sectors are affected by the restriction operation. In both cases, the movement region is restricted more than what would be strictly necessary to preserve the edge crossing properties.

At the end of the maximal movement computation, the movement region of each node contains only points that u can reach without crossing any edge while moving.

Movement Restriction (Projection Inside). Let $e = [v_1, v_2]$ be an edge and u be a node that is not an extremity of e. Let p be the projection of u onto the line described by e, and let us assume $p \in \boldsymbol{p}_e$. We identify with $\overrightarrow{up}$ the vector connecting u to its projection p, and with i the index of the angular interval $\mathcal{A}^i$ that contains it. The maximal movement of the nodes u, v_1 and v_2 is restricted to

$$\mathcal{M}_u^j \leftarrow \min\left(\mathcal{M}_u^j, \frac{\|\overrightarrow{up}\|}{3}\right) \qquad j = (i-2 \mod 8)\ldots(i+2 \mod 8)$$

$$\mathcal{M}_{v_1}^j \leftarrow \min\left(\mathcal{M}_{v_1}^j, \frac{\|\overrightarrow{up}\|}{3}\right) \qquad j = (i+2 \mod 8)\ldots(i+6 \mod 8)$$

$$\mathcal{M}_{v_2}^j \leftarrow \min\left(\mathcal{M}_{v_2}^j, \frac{\|\overrightarrow{up}\|}{3}\right) \qquad j = (i+2 \mod 8)\ldots(i+6 \mod 8)$$

Movement Restriction (Projection Outside). Let $e = [v_1, v_2]$ be an edge and u be a node that is not an extremity of e. Let p be the projection of u onto the line described by e, and let us assume $p \notin \boldsymbol{p}_e$. We identify with $\overrightarrow{uv_1}$ the vector connecting u to v_1, and with $\overrightarrow{uv_2}$ the vector connecting u to v_2. The maximal

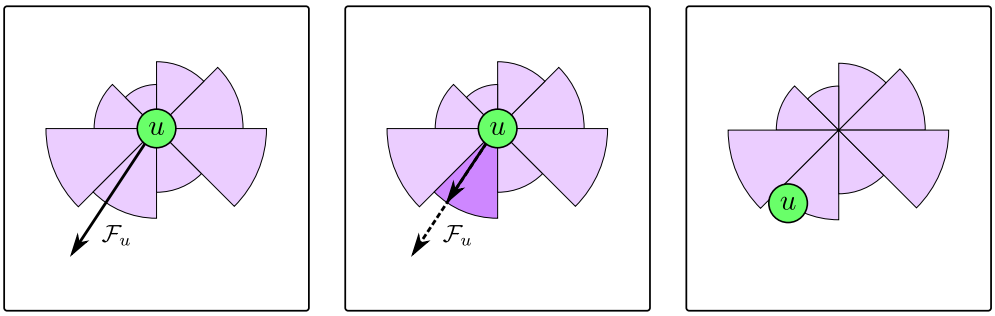


Figure 6.4: Node displacement in `PrEd`. **(a)** The resulting force $\mathcal{F}_u$ on node u and the computed movement region. **(b)** Since $\mathcal{F}_u$ has its angle on the interval $\mathcal{A}^5$, the length of the vector is clamped to $\mathcal{M}_u^5$. **(c)** Finally, the node is displaced by the current $\mathcal{F}_u$.

movement of the nodes u, v_1 and v_2 is restricted to

$$\mathcal{M}_u^j \leftarrow \min\left(\mathcal{M}_u^j,\ \min\left(\frac{\|\overrightarrow{uv_1}\|}{3}, \frac{\|\overrightarrow{uv_2}\|}{3}\right)\right) \qquad j = 0 \dots 7$$

$$\mathcal{M}_{v_1}^j \leftarrow \min\left(\mathcal{M}_{v_1}^j, \frac{\|\overrightarrow{uv_1}\|}{3}\right) \qquad j = 0 \dots 7$$

$$\mathcal{M}_{v_2}^j \leftarrow \min\left(\mathcal{M}_{v_2}^j, \frac{\|\overrightarrow{uv_2}\|}{3}\right) \qquad j = 0 \dots 7$$

6.1.5 Displacement of the Nodes

At the end of each iteration, `PrEd` moves the nodes according to the resulting force of each node. However, if the length of the resulting force vector exceeds the maximal movement of the sector in which it lies, the vector is clamped to the sector radius (see figure 6.4).

Node Displacement. Let $\mathcal{F}_u$ be the resulting force on the node u, and let i be the index of the angular interval $\mathcal{A}^i$ that contains $\mathcal{F}_u$. Before displacing each node, if $\|\mathcal{F}_u\| > \mathcal{M}_u^i$, the force vector is reduced to

$$\mathcal{F}_u \leftarrow \frac{\mathcal{F}_u}{\|\mathcal{F}_u\|} \cdot \mathcal{M}_u^i$$

Then, the new position of the node u is updated to

$$\boldsymbol{p}_u \leftarrow \boldsymbol{p}_u + \mathcal{F}_u$$

6.1.6 Advantages and Disadvantages

The advantages of PrEd reside on its main characteristic, the preservation of edge crossing properties, and on the typical advantages of force-directed algorithms:

- *Preservation of edge crossing properties.* Even though this characteristic has had a relatively small impact on the graph drawing field, the preservation of edge crossing properties opens a number of new applications for force-directed algorithms. Among those, the most important is the optimisation of planar straight-line drawings.

  Standard force-directed algorithms cannot guarantee that nodes will not cross an edge when moving: this is due to the discrete nature of the computation, and cannot be avoided by only relying on the force system (for instance by inserting strong node-edge repulsive forces). To our knowledge, the optimisation of the planar straight-line drawing of a graph with PrEd is the only way to obtain aesthetically pleasant graphs of this kind.

- *High aesthetics for certain kinds of graph.* As it typically happens with force-directed graphs, the output can present very high aesthetics, with good node distribution and angular resolution, good aspect ratio and uniform edge lengths.

  Unfortunately, these characteristics are generally only present in the layout of relatively small graphs. Also, due to the preservation of the edge crossing properties, the initial crossings must be compatible with those of a good final layout. This is particularly evident for the depiction of symmetries, since they will not appear if the initial crossings are different from those of the optimal drawing.

- *Intuitive and simple to implement.* As for most force-directed algorithm, the concepts behind PrEd are easily understandable and the algorithm is quite short and simple to implement.

However, PrEd also has a number of drawbacks:

- *High running time.* Each iteration of the algorithm has complexity $O(|V|^2 + |V||E|)$, and the number of iterations should be set to higher values when drawing larger graphs. As a result, the running time required to optimise the drawings of large graphs can be very large.

  High running times limit the use of the algorithm in applications that involve interactions with the user, since waiting times of more than a few seconds are reached already with graphs of a few hundreds nodes. Also, even when there is no real-time interaction with the user, PrEd is hardly usable with graphs of more than a few thousands nodes, since the computation for these graphs already reaches running times in the order of hours.

- *Over-restrictive node movement.* PrEd restricts the movement of the nodes more than what is strictly necessary to preserve the edge crossing properties. Figure 6.3 on page 107 shows how the restriction of the node movement is over-restrictive if the only aim is to avoid the node from crossing the edge.

  Even though the cautious approach of PrEd might increase the stability of the algorithm, a less restrictive one might accelerate the convergence of the drawing to its final configuration.

- *Low aesthetics of large and sparse graphs.* In the computation of the resulting force of a node, node-node repulsion forces typically have the greatest effect. Even though repulsive forces are less influential than attractive forces for far elements, their number is typically much higher. There are in fact $|V| - 1$ node-node repulsive forces acting on a node u, and only $\deg(u)$ attractive ones. When repulsive forces dominate the computation of the resulting force, the nodes are pushed towards the outer area of the drawing, adversely affecting the aesthetics criteria such as angular resolution, edge length uniformity and shape of the faces.

  If the graph is not dense, the imbalance between attractive and repulsive forces is worsened for larger graphs. If the graph is sparse, this problem is noticeable even for very small graphs. For example, in the case of a graph composed only of isolated nodes, $G = \{V, \emptyset\}$, there are no attractive forces and the nodes are pushed indefinitely far from each other, preventing the convergence to a stable configuration of the drawing.

- *Low control of the desired output.* The output obtained by the execution of `PrEd` can be influenced by choosing the parameters described in section 6.1.1. Higher values of δ will generate a more spaced layout, since it increases the magnitude of repulsive forces and the balancing distance between the edge extremities. Higher values of γ will instead induce higher distances between nodes and edges.

  Due to their role in the force formulae, δ and γ assume a natural interpretation that is fundamental for giving the user a way to select the parameter values. The parameter δ can be thought of as the optimal distance between adjacent nodes and as the minimal desired distance between unrelated nodes. The parameter γ can instead be thought of as the minimal desired distance between nodes and edges. Unfortunately, the results of `PrEd` are typically sub-optimal with respect to these interpretations, as the node spacing disrespects the value of the parameters even when this is not strictly required by the topology of the graph.

Finally, there is an aspect that cannot be classified as a limitation of the method, but that still reduces `PrEd`'s applicability:

- *Necessity of a meaningful initial drawing.* Most force-directed approaches can either start from an initial graph drawing, or generate an initial drawing where the nodes have random positions. In `PrEd` it is not possible to assign random initial positions to the nodes: the edge crossings obtained would not be meaningful, and there would be no reason to preserve them.

  Therefore, the algorithm can only be applied to graph drawings that have a meaningful set of initial crossings and a sub-optimal node distribution. This situation is not very common, as shown by the relatively low impact that `PrEd` has had in the graph drawing community.

6.2 ImPrEd

We developed `ImPrEd`, an improved version of `PrEd`, to mitigate or overcome some of the limitations of the original algorithm.

The main aims of `ImPrEd` are to significantly reduce the running time of `PrEd`, achieve high aesthetics even for large and sparse graphs, and make the algorithm more stable and reliable with respect to the input parameters. To achieve these results,

we inserted two new features to reduce the amount of computation required by the algorithm: one to improve the spacing of the graph elements and one to accelerate the convergence of the drawing to its final configuration.

In addition, we added two other features that are more oriented to the generation of Euler diagrams, but that still proved to be useful for the optimisation of general graphs. The first feature gives the possibility to define each individual edge as uncrossable or crossable, and as rigid or flexible. Thanks to this feature, the user can decide to have edges acting as in PrEd or as in a standard force-directed algorithm, and to decide whether an edge can insert or remove bends in order to respond to the stress of the current drawing. The second feature lets the user assign a weight to each graph element in order to increase or decrease their importance in the determination of the final drawing.

As with PrEd, we will start by presenting the input and the parameters of the algorithm. Then, we will present the modifications to the original algorithm, first by providing an overview of the code, and then by detailing each of the six newly introduced features.

6.2.1 Input Parameters

With the exception of the initial graph, the parameters of PrEd are substantially unchanged in ImPrEd:

- *Initial graph drawing*: the class of input graph is extended to multigraphs, and as in PrEd, the eventual direction of the edges is ignored. The drawing must follow the polyline drawing conventions and there must not be any overlap between nodes, bends and edges.
- *Number of iterations*: the number of times the improvement procedure is repeated. Higher values produce better results, but induce higher running times. Typical values span from one to five hundred. Unchanged with respect to PrEd.
- *The optimal edge length*, δ: the distance at which repulsive and attractive forces acting on adjacent nodes balance. Can also be interpreted as the minimal desired distance between unrelated nodes. Unchanged with respect to PrEd.
- *The optimal node-edge distance*, γ: the minimal distance at which non-incident nodes and edges do not repel each other. Can also be interpreted as the minimal desired distance between nodes and unrelated edges. Unchanged with respect to PrEd.

However, we added three optional parameters to steer the new features of ImPrEd that provide greater control on the desired output:

- *Set of crossable edges*, Cr (optional): the set of crossable edges in the graph. By default, according to PrEd behaviour, every edge is considered uncrossable. Using this optional parameter, the user can provide the set of edges to be marked as crossable, to have them acting as edges in a standard force-directed algorithm rather than as standard PrEd edges.
- *Set of flexible edges*, Fl (optional): the set of flexible edges in the graph. By default, all edges are considered rigid. Using this optional parameter, the user can provide the set of edges to be marked as flexible to make them adapt their number of bends according to the stress they are subjected to.

- *Element weight*, $\mathcal{W}$ (optional): the weight of nodes and edges of the graph. By default, each element of the graph has unit weight. Using this optional parameter, the user can set different weights for some or all the nodes and edges of the graph.

6.2.2 The Algorithm

`ImPrEd` differs from `PrEd` for the introduction of several new features that introduce a few steps in the algorithm or slightly modify the existing ones. However, the structure and the main phases of `PrEd` are mostly preserved.

The pseudo-code of `ImPrEd` is presented in algorithm 6.2. In the code we have used colours to identify the new commands and how they relate to the features introduced:

- *Force and movement cooling* (●). We define a new variable, called temperature, which gradually decreases with the progress of the computation. The node movement and the node ability to influence distant nodes are linked to the current temperature, obtaining a cooling effect that progressively enforces smaller movements and forces with local influence.

  This feature aims to strongly increase the aesthetics of the drawings and the ability of the user to control the characteristics of the output.

- *Surrounding edges computation* (●). In the maximal movement computation phase, `PrEd` restricts the nodes' movement by considering each node-edge pair. Since this is typically highly redundant we compute, for each node, a subset of edges, called the surrounding edges, that need to be considered at that stage.

  This feature aims to considerably reduce the amount of computation required, leading to lower running times.

- *QuadTrees* (●). We use a data structure called a QuadTree, that allows to efficiently retrieve the nodes and edges in a certain area of the drawing. Since distant elements can be ignored without serious consequences in the force computation, we use QuadTrees to detect the nearby elements of a node and we compute the forces only for those elements.

  This feature aims to reduce the amount of computation performed at each iteration, decreasing the running time of the algorithm.

- *New maximal movement rules* (●). We devised new formulae for the maximal movement computation in order to maximise the movement regions of the nodes. Since nodes are able to move more freely, less iterations should be required to reach the same position than with more restrictive rules.

  This feature aims to accelerate the convergence of the drawing to a stable configuration. Therefore, the user can either decide to set a lower number of iterations, obtaining a lower running time for the same drawing quality, or keep the number of iterations unchanged, obtaining a higher drawing quality for the same running time.

- *Crossable and flexible edges* (●). We extend the input class of the algorithm to polyline multigraphs, and we allow the marking of each edge as crossable or uncrossable, and as flexible or rigid. Edges marked as crossable are considered as in standard force-directed algorithms, without the additional constraints of

Algorithm 6.2 ImPrEd. The major modifications with respect to PrEd's algorithm are coloured according to newly introduced feature they belong to: force and movement cooling, surrounding edges computation, QuadTrees, new maximal movement rules, crossable and flexible edges, weight of nodes and edges.

▷ Pre-processing

$g \leftarrow$ gravity centre
compute the surrounding edges $\mathcal{S}^e$ for each node, using edges $\notin$ Cr

for currentIteration $\leftarrow 0$ **to** numberOfIterations $- 1$ **do**
  temp $\leftarrow 1 -$ currentIteration/numberOfIterations
  compute the nearby nodes $\mathcal{N}^n$ and edges $\mathcal{N}^e$ for each node

▷ Force Computation

  for all $u \in V$ **do**
    force$(u) \leftarrow$ attraction to the gravity centre g
  for all $\{u, v\} \subseteq V$, $v \in \mathcal{N}^n(u)$ **do**
    currentForce $\leftarrow$ node-node repulsion according to temp and weights
    force$(u) \leftarrow$ force$(u) +$ currentForce
    force$(v) \leftarrow$ force$(v) -$ currentForce
  for all $e \in E$ **do**
    currentForce $\leftarrow$ edge attraction according to temp and weights
    force$(s(e)) \leftarrow$ force$(s(e)) +$ currentForce
    force$(t(e)) \leftarrow$ force$(t(e)) -$ currentForce
  for all $u \in V$, $e \in \mathcal{S}^e(u) \cap \mathcal{N}^e(u)$ **do**
    currentForce $\leftarrow$ edge-node repulsion according to temp and weights
    force$(u) \leftarrow$ force$(u) +$ currentForce
    force$(s(e)) \leftarrow$ force$(s(e)) -$ currentForce
    force$(t(e)) \leftarrow$ force$(t(e)) -$ currentForce

▷ Max Movement Computation

  for all $u \in V$, $i = 0 \dots 7$ **do**
    maxmove$(u, i) \leftarrow 5 \cdot \delta \cdot$ temp
  for all $u \in V$, $e \in \mathcal{S}^e(u) \cap \mathcal{N}^e(u)$ **do**
    restrict the movement of u, $s(e)$, $t(e)$

▷ Node Displacement

  for all $u \in V$ **do**
    $i \leftarrow$ sector where force(u) lies
    if $\|$force$(u)\| >$ maxmove(u, i) **then**
      force$(u) \leftarrow$ force(u) clamped to magnitude maxmove(u, i)
    position$(u) \leftarrow$ position$(u) +$ force(u) / weight(u)

▷ Refresh

  if currentIteration $\equiv 0 \mod 10$ **then**
    $g \leftarrow$ gravity centre
    for all $e \in E$ **do**
      expand stretched edge segments
      contract compressed edge segments

PrEd. Edges marked as flexible are allowed to change their number of bends according to the stress exerted on them. Edges marked as uncrossable and rigid do not benefit from these additional features, and behave as normal edges in PrEd.

Crossable and flexible edges are inserted to extend the range of possible applications and of possible outputs of the algorithm.

- *Weight of nodes and edges* (•). By default all nodes and edges have the same influence on the other elements of the graph. However, there are cases where some of the graph elements have different roles or different logical importance. For instance, in a graph depicting the transport network of a country, the nodes relative to bigger hubs have higher importance than locations carrying little traffic. Element weight is designed to make important elements exert forces of higher magnitude and have higher reluctance to move, so that elements with higher weight contribute more significantly to the final graph structure.

  The element weight increases the ability of the user to control the shape of the final drawing.

Each of these features will be explained in greater details in the following sections.

6.2.3 Force and Movement Cooling

The force system and the movement in PrEd present a couple of problems that obstruct the determination of drawings with high aesthetics:

- *Force magnitude.* In our opinion, both repulsive and attractive forces should have lower magnitude when acting on distant elements. Repulsive forces, as described in section 6.1.6, are in great number and dominate the computation of the resulting force, leading to low aesthetics.

  Attractive forces are instead designed to pull together the extremities of an edge in order to obtain an uniform edge length, and are therefore of high magnitude in the case of long edges. Although this behaviour might fit a standard force-directed algorithm, we believe it is contradictory with the aims of PrEd and ImPrEd. Since nodes cannot cross edges, the extremities of a long edge may be tightly constrained, the edge may not be able to assume the desired length resulting in large, irreducible forces that delay convergence.

  On the other hand, we desire higher repulsion forces for nodes at a closer distance than δ, so that we can obtain an adequate spacing between them.

- *Difficult convergence of the nodes to a stable position.* The discrete nature of the computation might be an obstacle to the convergence of the nodes to a stable position. Let us consider two adjacent nodes at a distance significantly different than δ. Ideally, the sequence of the distances at each iteration should converge to an interval reasonably close to δ before the end of the computation.

  However, it might happen that nodes in such a configuration keep jumping between being too close and being too far so that they cannot reach the optimal distance within a reasonable number of iterations. It also might happen that the force system overreacts, generating forces of increasingly higher magnitude that impede the reaching of a stable configuration.

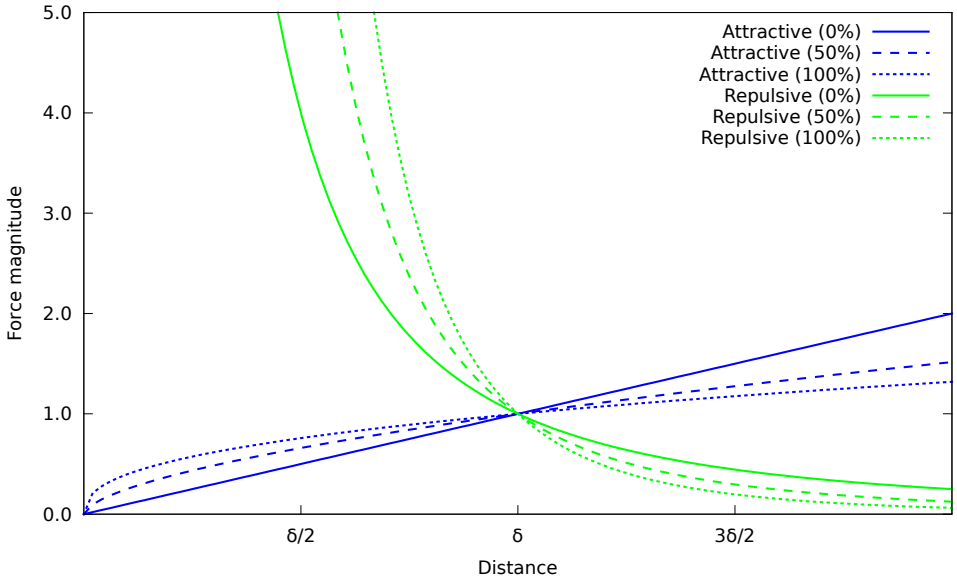


Figure 6.5: Magnitude of cooled forces in `ImPrEd`, with respect to the distance between the graph elements. In green, the node-node repulsive force. In blue, the edge attraction force. Lines with smaller dashes indicate later stages of the computation. The forces always balance at distance δ.

Ideally, the problems could be solved by lowering the magnitude of distant forces and by making nodes move a really small distance at each iteration. Unfortunately, this would considerably slow down the optimisation process, since forces acting on distant elements and high node mobility are essential at the early stages of the computation to quickly reach the rough shape of the final diagram. We therefore chose to lower these values along with the progress of the computation.

Temperature. The concept of temperature has been largely used in graph drawing [17, 42] to describe abstract behaviours that recall the physical phenomenon. Also in this case, we introduce the temperature as a measure of how much the graph nodes are able to displace and to influence other nodes, in analogy with the interpretation of temperature as a measure of the kinetic energy of atoms and molecules.

In `ImPrEd`, the temperature is a global variable that assumes rational values in the interval $(0, 1]$. The temperature starts at value one and gradually decreases towards zero with the progress of the computation. We control the mobility of the nodes by linearly decreasing their initial value with the cooling of the temperature.

The exponents of repulsive and attractive forces are also linearly increased and decreased according to the value of the temperature. As shown in figure 6.5, the forces always balance at the optimal distance δ, but the magnitude of the forces on distant elements decreases and the magnitude of the repulsive forces between close elements increases with the progress of the computation. This generates a shift between the influence of the forces from a global scale to a local scale.

Finally, in order to obtain a good aspect ratio and a compact drawing even in the case of disconnected graphs, we included a gravity force similar to that of Frick, Ludwig and Mehldau [42].

Temperature. We introduce a variable representing the global *temperature* of the graph, computed at each iteration as

$$\text{temp} = 1 - \frac{\text{currentIteration}}{\text{numberOfIterations}}$$

where currentIteration assumes the values from 0 to numberOfIterations − 1.

Force Exponents. In `ImPrEd`, we use variable exponents for the formulae of the forces:

$$x = x_f - (x_f - x_i) \cdot \text{temp}$$

where x_i is the initial exponent, and x_f is the final exponent. The exponent for the repulsive forces, x^r, is computed considering $x_i = 2$ and $x_f = 4$, while the exponent of the attraction forces, x^a, is computed considering $x_i = 1$ and $x_f = 0.4$.

In the following formulae, we will highlight in colour the differences with the relative formulae of `PrEd`:

$$\mathcal{F}_u^r(u,v) = \left(\frac{\delta}{\|\boldsymbol{p}_u - \boldsymbol{p}_v\|}\right)^{x^r} (\boldsymbol{p}_u - \boldsymbol{p}_v)$$

$$\mathcal{F}_u^a(e) = \left(\frac{\|\boldsymbol{p}_u - \boldsymbol{p}_v\|}{\delta}\right)^{x^a} (\boldsymbol{p}_v - \boldsymbol{p}_u)$$

$$\mathcal{F}_u^e(u,e) = \begin{cases} \left(\dfrac{\gamma - \|\boldsymbol{p}_u - \boldsymbol{p}_p\|}{\|\boldsymbol{p}_u - \boldsymbol{p}_p\|}\right)^{x^r} (\boldsymbol{p}_u - \boldsymbol{p}_p) & \text{if } p \in \boldsymbol{p}_e \text{ and } \|\boldsymbol{p}_u - \boldsymbol{p}_p\| < \gamma \\ \mathbf{0} & \text{otherwise} \end{cases}$$

Gravity Force. In `ImPrEd`, all nodes are attracted to a centre of gravity with position

$$\boldsymbol{g} = \frac{\sum_{u \in V} \boldsymbol{p}_u}{|V|}$$

The position of the gravity centre is updated only in the refresh phase, therefore once every ten iterations. The *gravity attraction* $\mathcal{F}^g(u)$ acting on a node u is the vector

$$\mathcal{F}^g(u) = \left(\frac{\delta}{\|\boldsymbol{p}_u - \boldsymbol{g}\|}\right) (\boldsymbol{p}_u - \boldsymbol{g})$$

Node Movement. In `ImPrEd`, at each iteration we initialise the maximal movement vector of each node u to

$$\mathcal{M}_u^i \leftarrow 6\delta \cdot \text{temp} \qquad \text{for } i = 0 \ldots 7$$

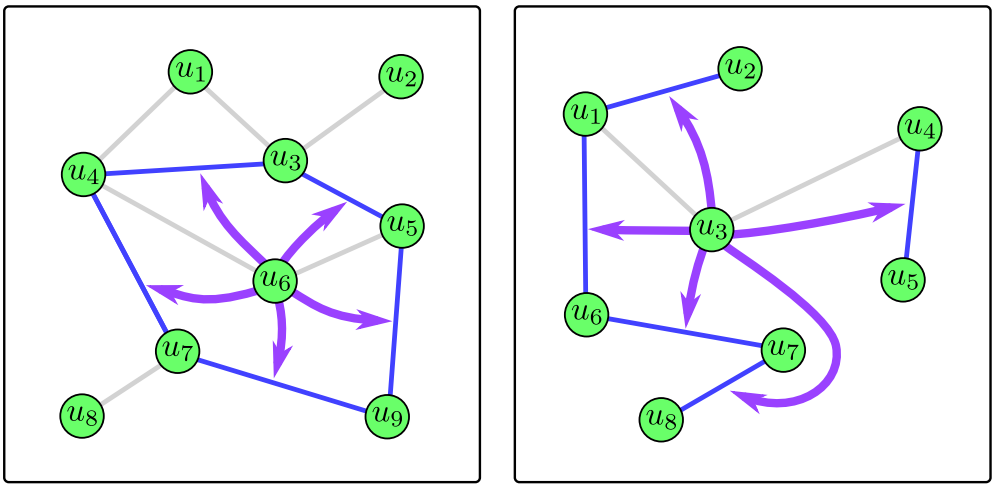


Figure 6.6: Examples of surrounding edges of a node. The surrounding edges are in blue. The grey edges are either incident to the node, and therefore ignored, or can be crossed only after crossing a surrounding edge. **(a)** Generally, the surrounding edges of a node are significantly fewer than $|E|$. **(b)** In a tree, all the nodes have every non-incident edge as a surrounding edge.

6.2.4 Surrounding Edges Computation

In `PrEd`, when restricting the maximal node movement, we check all the node-edge pairs to verify that a crossing will not occur. However, there are cases where many of these checks are redundant. Let us consider a large planar graph where an isolated node is contained in a small face. Since the node must cross the boundary of the face to cross any other edge in the graph, the checks performed for the non-boundary edges are unnecessary (see figure 6.6a).

We call the surrounding edges of a node the minimal set of edges that we need to check in order to enforce the non-crossing condition to all the edges of a graph. In the general case, the number of surrounding edges is much lower than the number of edges of a graph, providing a significant reduction in the necessary computation. Unfortunately, this highly depends on the kind of graph: in a tree, for example, all non-incident edges are surrounding edges of each node (see figure 6.6b).

In a graph drawing without crossings, the surrounding edges are identified by studying the boundary edges of the graph faces. In a drawing with crossings, the same method can be used, but only after extending the concept of a face, that is normally defined only for plane graphs. Finally, the newly introduced flexible edges can expand and contract by adding new bends. Since each bend is modelled as a node, it is necessary to provide the surrounding edges for these potential new nodes.

Surrounding Edges. The *surrounding edges* of a node u in a graph $G = (V, E)$ are a minimal subset of edges $\mathcal{S}^e(u) \subseteq E$ so that, for every edge $e \in E$,

- either u is an extremity of e,

- or $e \in \mathcal{S}^e(u)$,
- or e can be crossed by u only after crossing an edge in $\mathcal{S}^e(u)$.

Surrounding Edges in Plane Graphs. Let us consider a plane graph $G = (V, E)$ and the node $u \in V$. Let $G' = (V, E')$ be a subgraph of G in which the incident edges of u have been removed:

$$E' = \{e \in E : u \neq s(e) \wedge u \neq t(e)\}$$

In G', u is an isolated node inside a face f. The edges $\mathcal{B}^e(f)$ that bound f are the boundary edges of u, as they are not incident to it, and as all remaining edges can be crossed only if u moves out of the face. Also, the set is minimal, since every node in $\mathcal{B}^e(f)$ is directly reachable by u when moving.

The surrounding edges can also be identified directly in G as the union of the boundary edges $\mathcal{B}^e(f)$ for each face f that contains u, with the exclusion of its incident edges:

$$\mathcal{S}^e(u) = \left(\bigcup_{f \in F:\ u \in \mathcal{B}^n(f)} \mathcal{B}^e(f) \right) \setminus \{e \in E : e \text{ is incident to } u\}$$

Since flexible edges can introduce bends, that will be treated as new nodes, we compute the surrounding edges $\mathcal{S}^e(e)$ associated with a flexible edge e. The set $\mathcal{S}^e(e)$ represents the surrounding edges $\mathcal{S}^e(u)$ of each bend that might be introduced in e during the layout optimisation. The surrounding edges of a flexible edge are computed as

$$\mathcal{S}^e(e) = \left(\bigcup_{f \in F:\ e \in \mathcal{B}^e(f)} \mathcal{B}^e(f) \right) \setminus \{e\}$$

Figure 6.7a shows how to compute the surrounding edges of a node in a plane graph, while figure 6.7b shows the computation of the surrounding edges of a flexible edge.

Since in a plane graph the non crossing conditions produce drawings with equivalent embedding, the faces are not modified by the node movement and the surrounding edges do not change during the layout improvement.

Surrounding Edges in Non-Plane Graphs. Let us extend the concept of a face to non-planar drawings. As for plane graphs, a face is each of the connected regions in the plane. For the faces in non-plane graphs we have that

- only a portion of an edge might form the boundary of a face, while in plane graphs the whole edge contributes to the face,
- each edge can (partially) belong to more than two faces, while in plane graphs the edges belong only to one or two faces,
- it might not be possible to identify a closed walk on the boundary of a face, while in plane graphs the boundary is composed of the elements of one or more closed walks.

By extending the definition of a face, the formulae above hold in the case of non-plane graphs. Figure 6.7c shows the previous properties, as well as an example of surrounding edge computation for a node in a non-plane graph.

The determination of the surrounding edges requires that we identify the faces in the drawing. This task requires increasing effort with respect to the following conditions: plane connected graphs, plane disconnected graphs, and non-plane graphs. However, since the algorithm prevents edge crossings, the surrounding edges of a node do not change with the layout improvement performed at each iteration. Therefore, we can compute them before the main cycle, and we do not need to recompute or update them.

Face Identification in Plane Connected Graphs

If the graph is plane and connected, the boundaries of a face can be identified with a walk along the graph elements in the combinatorial embedding.

Since the combinatiorial embedding contains the ordering of the edges incident to a node, we can use this information to select the next edge in the boundary. If we start moving along one edge to one of its extremities, we can identify a walk on the boundary of a face by exiting the nodes using the first clockwise or the first anticlockwise edge from the one we used to enter.

By detecting a collection of different walks of this kind, so that each edge is used exactly twice, we obtain the boundaries of all the faces present in the drawing. An example of face identification in plane connected graph is shown in figure 6.8.

Face Identification in Plane Connected Graphs. In plane connected graphs, the following conditions hold:

- each edge belongs completely to the boundary of one or two faces,
- for each face, there is a closed walk that includes all the elements of the boundary of a face,
- if an edge belongs to a single face, the walk along the face boundary must include the edge twice.

Under these conditions, the boundary of the faces in the graph can be identified as a collection of walks computed as follows.

Let us select an edge, e_1, one of its extremities, u_0, and an orientation, either clockwise or anticlockwise. Using the combinatorial embedding of the graph, we perform the walk $u_0 e_1 u_1 \dots e_l u_l$ where the edge e_{i+1} is the following element of e_i in the edge ordering of the node u_i, according to the chosen orientation. This will grant that the walk only considers the elements in the boundary of a face.

When the walks reaches u_0 again, and becomes therefore closed, all the elements considered form the boundary of a face. To identify all the face boundaries in the graph, we perform another walk with different starting conditions until all edges have been considered twice.

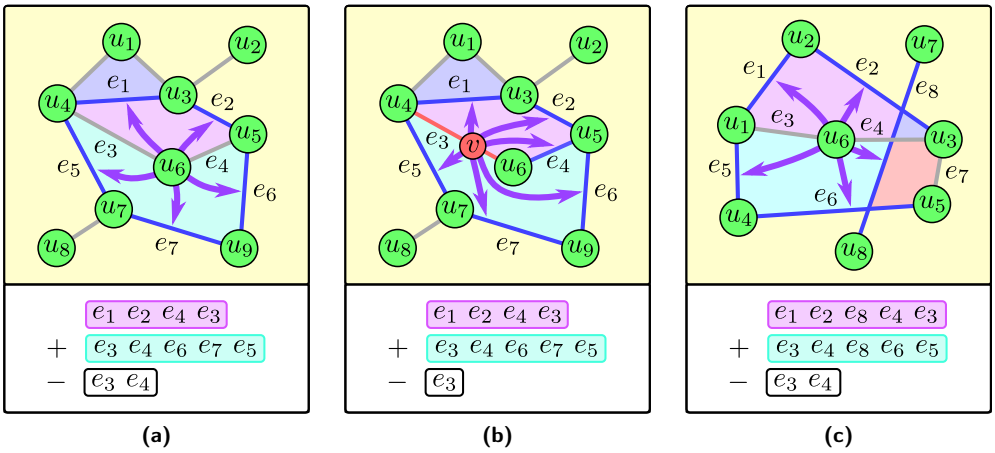


Figure 6.7: Computation of the surrounding edges. The surrounding edges are marked in blue. **(a)** The surrounding edges for u_6 are in the boundaries of the faces that contain the node (turquoise face and purple face), excluding the incident edges of u_6. **(b)** Let e_3 be a flexible edge. The surrounding edges for the eventual bends v of e_3 are in the boundaries of the faces that contain the edge (the turquoise face and purple face), excluding the edge itself. Note that this is consistent with the previous case. **(c)** In the case of crossing edges, the concept of a face needs to be extended. Note that edges such as e_4 and e_8 now only partially belong to the boundary of a face, and belong to more than two faces at the same time. Also, the elements on the boundary of the faces can no longer be visited in a single walk.

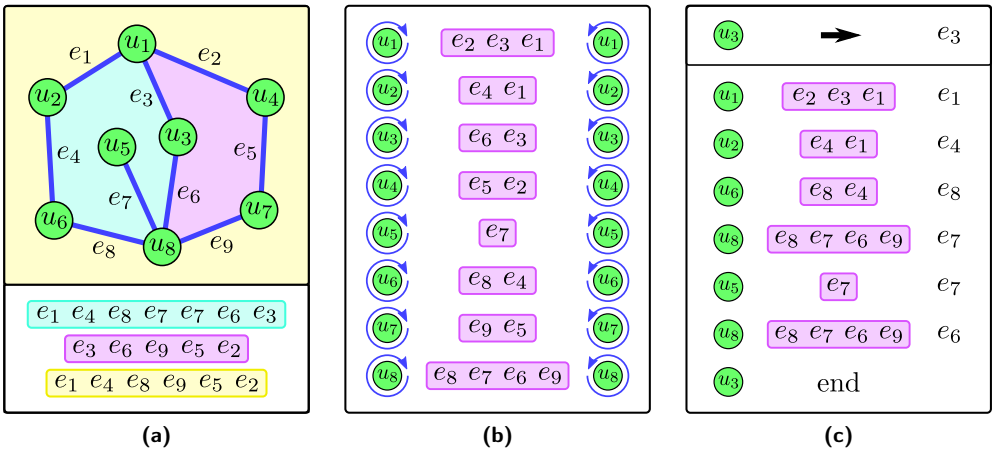


Figure 6.8: Identification of the face boundary in plane connected graphs. **(a)** A plane connected graph. **(b)** The combinatiorial embedding for the previous graph. **(c)** The identification of a face boundary starts with the choice of an edge, e_3, one of its extremities, u_3, and an orientation, clockwise, that corresponds to the right-hand direction in the lists provided. From u_3, the edge e_3 leads to the node u_1. Here, we select the element that is just right of the current edge, e_3, obtaining the edge e_1. The process is repeated until we reach the initial node, u_3. Note that the right column of the visited edges corresponds to the boundaries of the turquoise face in the initial graph.

Face Identification in Plane Disconnected Graphs

When the graph is plane but disconnected, the identification of the boundary of a face cannot be done only considering the combinatorial embedding, since disconnected face boundaries cannot be reached through a walk. We propose a method based on the determination of the inclusion relationships among the connected components of the graph.

Each connected component is analysed separately as described above. Then we study their inclusion relationships, since every connected component lies entirely in one face of any other component. Finally, each face boundary is computed as the inner face of one component, plus the external faces of all components lying in that face.

External and Internal Faces. A connected graph divides the space into one unbounded face and zero or more bounded faces. The unbounded face is also called *external face* and is indicated with f_0. The bounded faces are also called *internal faces* and are indicated with an index greater than zero or with f_+.

Relations Between the Components of a Disconnected Graph. Let C be the set of the connected components of the graph. Each component $c_i \in C$, considered individually, is a connected graph and divides the plane into faces f_j^i. Consistently with the above notation, we indicate with f_0^i the external face of the component c_i and with f_j^i, $j > 0$, or with f_+^i, an internal face of the component (see figures 6.9a and 6.9b).

In a plane disconnected graph, each connected component c_i lies entirely in one face f_k^j of a different component c_j. Also, we have that

$$c_i \text{ lies in an internal face of } c_j \quad \Rightarrow \quad c_j \text{ lies in the external face of } c_i$$

The converse implication does not hold, since two components c_i and c_j can both be on the external face of the other.

It is possible to establish a partial ordering between the components in C where $c_i < c_j$ if c_i lies in an internal face of c_j. It is also possible to describe the hierarchical organisation of the components into a bipartite forest H, where each component has its internal faces as children, and is child of the internal face that directly contains it according to the partial ordering (see figure 6.9c).

Face Identification in Plane Disconnected Graphs. Each internal face of a plane disconnected graph G is identified as the intersection between an internal face of a component and the external face of all its children in H:

$$f_+ = f_j^i \cap \bigcap_{k \in K} f_0^k \quad \text{where } j \neq 0 \text{ and } K = \{k \in \mathbb{N} : c_k \text{ is child of } f_j^i \text{ in } H\}$$

The external face of G is instead the intersection between the external faces of the roots of H:

$$f_0 = \bigcap_{k \in K} f_0^k \quad \text{where } K = \{k \in \mathbb{N} : c_k \text{ is a root in } H\}$$

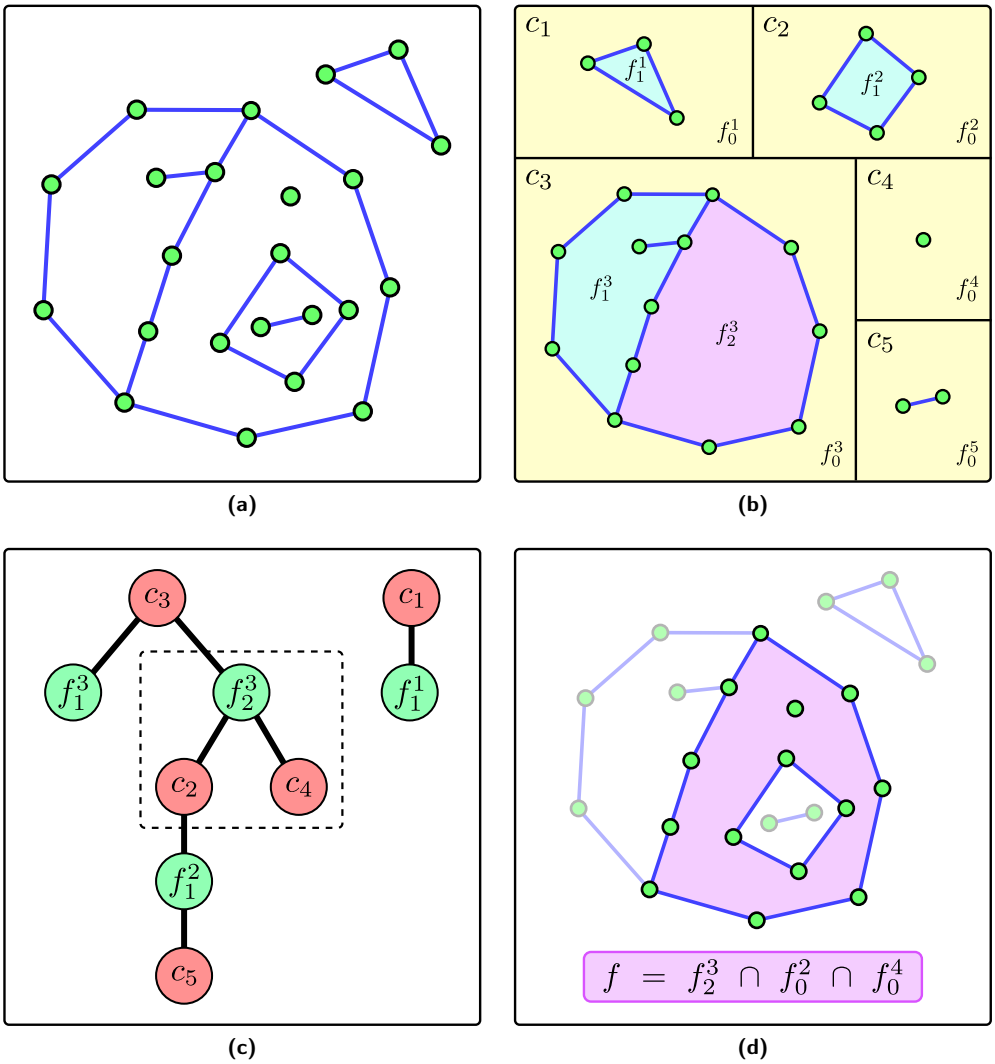


Figure 6.9: Determination of the faces in disconnected plane graphs. **(a)** An example of disconnected plane graphs. **(b)** Decomposition of the graph into its connected components and identification of their faces. **(c)** Organisation of connected components and internal faces into a hierarchy. A face of the original graph can be identified starting from an internal face and its children. **(d)** The face f is computed as the intersection of the internal face f_2^3 and the external faces f_0^2 and f_0^4. The boundary of f can be identified as the union of the boundaries of those faces, as shown by the non-shaded elements in the drawing.

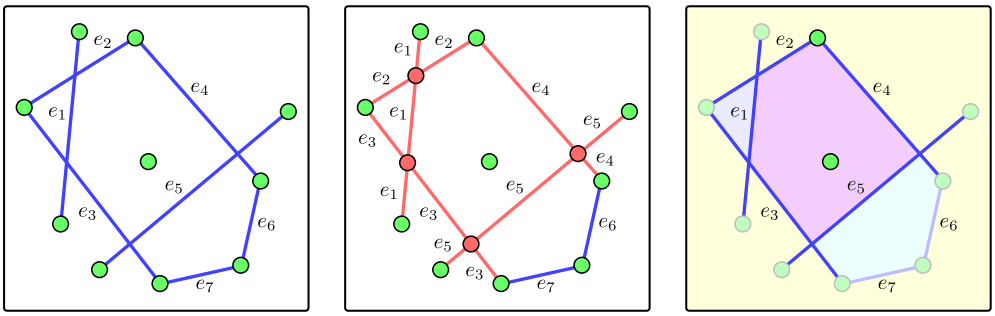


Figure 6.10: Determination of the faces in non-plane graphs. **(a)** An example of a non-plane graph. **(b)** Through planar augmentation, we obtain a plane graph. The newly inserted nodes and edges are coloured red, and the new segments are labelled with the identifiers of the original edges. **(c)** The faces of the original graph are identified on the previous one, ignoring the red nodes and using the identifiers of the original edges. The boundaries obtained for the purple face are in non-shaded colours. Unlike the case of plane graphs, the boundaries are not a collection of walks: often, the extremities of a boundary edge are not part of the boundary.

The boundaries of each of these faces can be computed as the union of the boundaries of the faces considered (see figure 6.9d), and therefore

$$\mathcal{B}(f_+) = \mathcal{B}\left(f_j^i\right) \cup \bigcup_{k \in K} \mathcal{B}\left(f_0^k\right)$$

$$\mathcal{B}(f_0) = \bigcup_{k \in K} \mathcal{B}\left(f_0^k\right)$$

where j and K have the same conditions as those in the above formulae.

Face Identification in Non-Plane Graphs

When the graph is not plane, we use a technique called planar augmentation to return to the previous case. Planar augmentation substitutes any crossing edges with their segments, connected to new nodes placed in correspondence to the crossing points. Thus, the resulting graph is plane even when the initial drawing contains crossings.

To compute the surrounding edges, we apply planar augmentation to the initial graph, assign the identifier of the initial edges to each of their segments, and compute the faces as described above. Figure 6.10 shows an example of the application of this procedure.

Planar Augmentation. Planar augmentation is a technique that transforms a graph $G = (V, E)$ with edge crossings into a plane graph $G' = (V', E')$. The set of nodes V' contains all the original nodes, but also a node for each edge crossing. The set of edges E' contains only the original edges without crossings, while the edges with crossings are substituted by an edge for each of their segments.

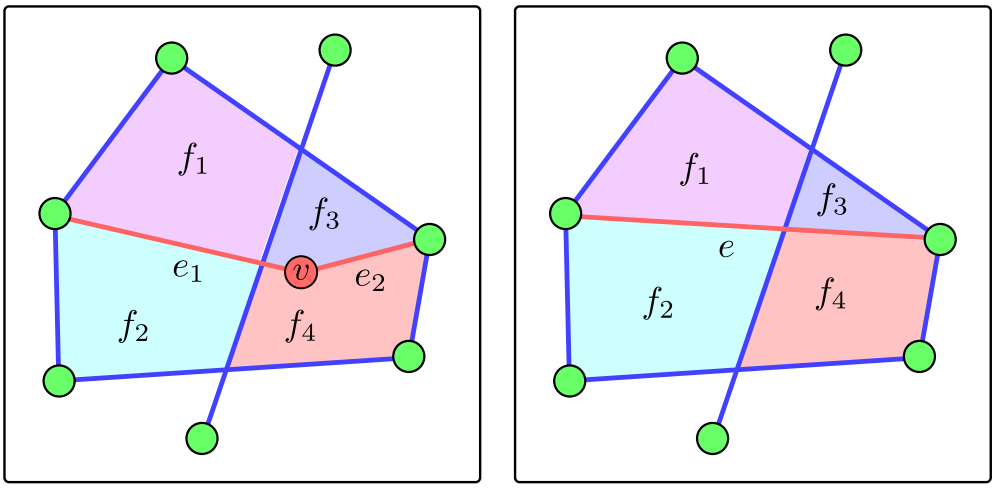


Figure 6.11: Surrounding edges in the presence of flexible edges. **(a)** When moving the bend v, it would not be necessary to control the edges in the boundary of f_1 and f_2, that are surrounding edges of v. **(b)** This redundancy eliminates the need of recomputing the surrounding edges if V is removed or if it is re-created along the borders of f_1 and f_2. Also, the surrounding edges are consistent with the set that would be computed if the edge started with no bends and if all the original bends were added during the computation.

Let $e = [u_1, u_2]$ be an original edge with n crossings. In G', each crossing is associated with a node $v \in V'$. Let $v_1 \dots v_n$ be these nodes in the order obtained when moving from u_1 to u_2 along the edge e. In G', the edge e is substituted by a sequence of edges $e_1 \dots e_{n+1}$ where $e_i = [v_i, v_{i+1}]$.

Surrounding Edges in Non-Plane Graphs. To specify the concept of a face, which is not defined for non-plane graphs, we transform the graph into a plane one through planar augmentation. We label each edge of G' with an identifier of the corresponding edge in G (see figures 6.10a and 6.10b). Since G' is a plane graph, we identify the surrounding edges as specified above, using the labels assigned (see figure 6.10c).

Let $\mathcal{S}^e_{G'}$ be the surrounding edges computed in G' as explained and $\mathcal{S}^e_G$ the surrounding edges to be computed with respect to G. For the nodes $u \in V$, we have that $\mathcal{S}^e_G(u) = \mathcal{S}^e_{G'}(u)$. The surrounding edges of a node $u \in V'$ that correspond to a crossing, $u \notin V$, are not used and their computation can be skipped. The surrounding edges $\mathcal{S}^e_G(e)$ of a flexible edge are instead computed as the union of the flexible edges $\mathcal{S}^e_{G'}(e_i)$ of each of its segments e_i.

Note that the mechanism of flexible edges can result in a non-minimal set of surrounding edges. However, by using the whole set of surrounding edges associated with a flexible edge, we do not need to identify the faces in which the new bend is created and to update the surrounding edges of each element according to the modifications of a flexible edge (see figure 6.11).

6.2.5 QuadTrees

We apply the technique of QuadTrees [45, 81] to efficiently retrieve nodes and edges that lie in a certain area of the graph. The method is well known in the literature and largely applied to reduce the computation time in force-directed algorithms.

QuadTrees progressively divide a given rectangle into four quadrants, starting from the whole plane and iterating the procedure on the quadrants. These subdivisions are then organised into a tree and associated with the set of elements they contain (see figures 6.12a and 6.12b).

In force-directed algorithms, the forces exerted by distant elements have low magnitude, especially when compared to that of nearby elements. Therefore, the effect of distant elements can be approximated or even ignored without significantly influencing the result of the computation. Using QuadTrees, we can obtain the elements at closer distance than d from a node u by analysing the cells in the surroundings of u, and not having to consider all the nodes and edges of a graph (see figures 6.12c and 6.12d).

The nearby elements detected with QuadTrees include all elements at a closer distance than d, but might also include some element at a farther distance. Also, it might include elements at a distance $d_1 > d$ and exclude elements at a distance $d_2 > d$, even when $d_1 > d_2 > d$. However, for the reasons explained above, we can ignore these issues once we choose a distance d high enough to include the elements that significantly influence the resulting force of a node.

QuadTrees in ImPrEd

In the algorithm, we compute the QuadTrees at the beginning of each iteration. For each node u, we compute the set of nearby nodes $\mathcal{N}^n(u)$ at distance 3δ from u and we use them in the computation of the node-node repulsive forces. A node at greater distance than 3δ generates a force with magnitude lower than a ninth of that generated by a node at distance δ, and can be ignored without great consequences on the resulting force of u.

Also, we compute two sets of nearby edges $\mathcal{N}^e(u)$ at distances γ and $12\delta \cdot \text{temp}$ from u. We use the first set for the computation of the node-edge repulsive force, since edges at a greater distance than γ from a node u do not contribute to the computation of the resulting force.

The second set is instead used in the maximal movement computation. Since we initialise the maximal movement to the value $6\delta \cdot \text{temp}$ (see section 6.2.3), and since this value can only be decreased during the computation, no node can move farther. For this reason, a node and an edge at a greater distance than $12\delta \cdot \text{temp}$ cannot cross at this iteration, even if they move towards each other. They can therefore be ignored in the computation of the maximal movement.

6.2.6 New Maximal Movement Rules

In ImPrEd, we decided to preserve the sector policy of PrEd. This is not strictly necessary, since it would be possible to compute first the resulting force, and then the maximal movement for the orientation of the force only. In this way we obtain both a reduction of the computation time and a more precise movement computation. However, the algorithm complexity would not change since the sectors only have a constant effect, and the identification of a full movement region facilitates the transformation of the grid graph edges into Bézier curves.

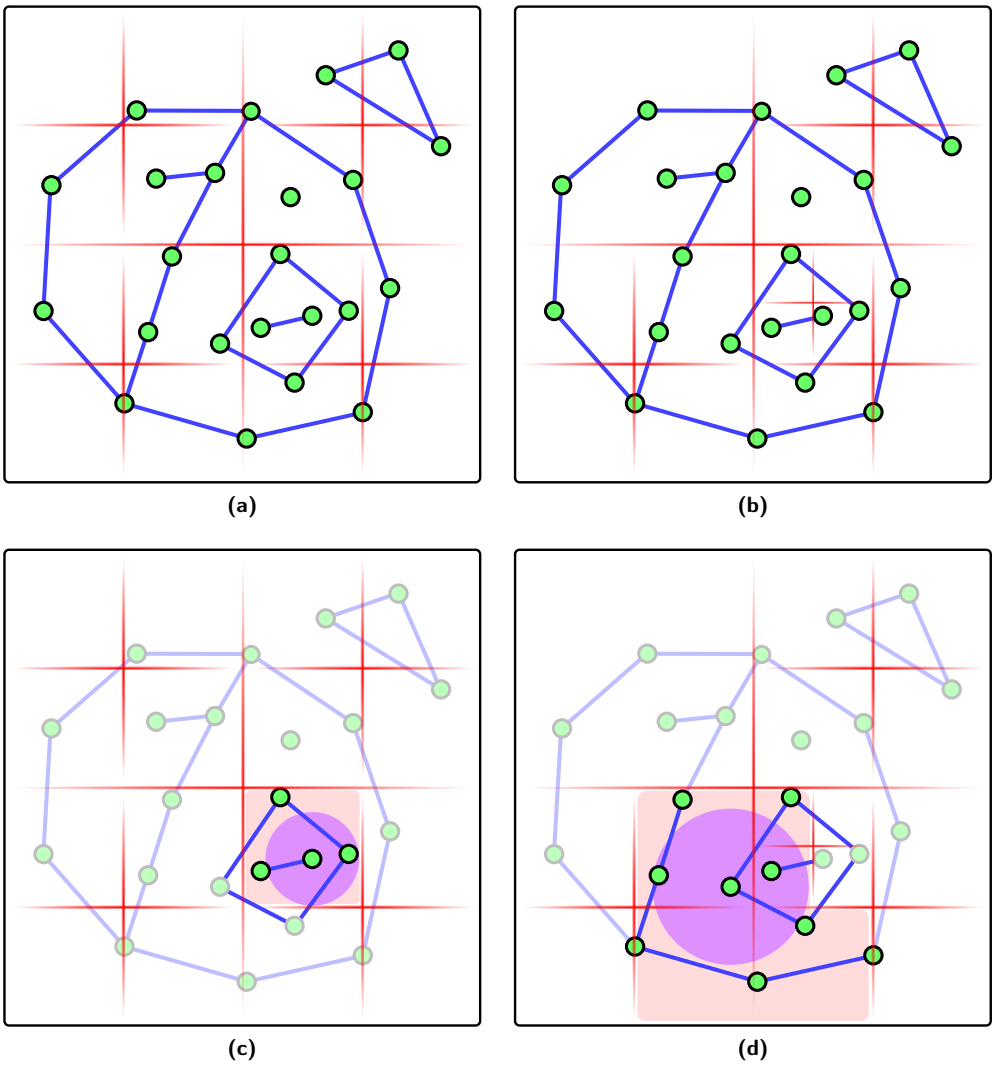


Figure 6.12: QuadTree illustration. **(a)** The area of the drawing is divided into quadrants and the graph elements are associated with the regions that contain them. The process is repeated in each individual quadrant. **(b)** The choice of further subdividing a cell is generally linked to the amount of graph elements contained in it. Therefore, denser regions might be divided in smaller cells, obtaining a non uniform grid. **(c)** When we are looking for the graph elements closer than a given distance d from a node u (the purple region in the figure), we only consider the cell or the cells that might contain such elements (the light red region in the figure). **(d)** Therefore, according to d and to the position of u in the cell, we might have to consider other cells that are nearby that of the node.

Nevertheless, the maximal movement computation in PrEd is improvable. In ImPrEd, we designed a restriction policy that maximises the movement region of the nodes to grant them higher mobility. This is particularly needed for the sectors that are not in the collision direction between node and edge.

Figure 6.13a shows how the movement of the nodes is heavily bounded even when moving along the perpendicular to $\overrightarrow{up}$, which is perfectly safe. Moreover, when a node is inside an acute angle or between two edges, its movement will be unnecessarily constrained in all directions. By allowing nodes to escape these high-stress configurations, we are likely to obtain a stable configuration quickly, with additional positive effects on drawing quality.

The main idea behind the new maximal movement rules is to divide the plane into two half-planes, so that node u and edge $e = [v_1, v_2]$ can move freely in the respective half-plane without occurring into crossings. Once this division of the plane is computed, it will be possible to extend the movement regions of u, v_1 and v_2, up to the limits of their half-planes (see figure 6.13).

Division of the Plane in Halves. Let $e = [v_1, v_2]$ be an edge and u be a node that is not extremity of e. Let p be the projection of u onto the line defined by e. We identify a line l as the line perpendicular to, and passing through the middle point of, the segment

$$\begin{aligned} &\text{Case } p \in e: \quad \overline{up} \\ &\text{Case } p \notin e: \quad \min(\overline{uv_1}, \overline{uv_2}) \end{aligned}$$

where the minimum is intended as the shorter of the two segments. In both cases, we defined the collision vectors $\boldsymbol{c}_u$, $\boldsymbol{c}_{v_1}$ and $\boldsymbol{c}_{v_2}$ as the vectors connecting the nodes u, v_1 and v_2 to their projection on l (see figure 6.14).

Movement Restriction (Both Cases). To permit maximal movement, each sector can be extended until it touches, but does not become collinear with, the line l. This result can be achieved by multiplying the length of each collision vector by σ, defined as the secant of the angle between $\boldsymbol{c}_x$ and closest radius of the sector:

$$\mathcal{M}_x^j \leftarrow 0.9 \cdot \|\boldsymbol{c}_x\| \cdot \sigma(\boldsymbol{c}_x, \mathcal{A}^j)$$

$$\sigma(\vec{c}_x, \mathcal{A}^j) = \begin{cases} 1 & i - j \equiv 0 \mod 8 \\ \sec\left(\angle\vec{c}_x - (j+1)\pi/4\right) & i - j \equiv 1, 2 \mod 8 \\ \sec\left(\angle\vec{c}_x - j\pi/4\right) & i - j \equiv 6, 7 \mod 8 \\ \infty & i - j \equiv 3, 4, 5 \mod 8 \end{cases}$$

where $\angle\boldsymbol{c}_x$ is the angle of the given vector and where i is the index of the angle $\mathcal{A}^i$ in which $\boldsymbol{c}_x$ lies.

In the first case, the vector lies on the sector and the closest radius coincides with $\boldsymbol{c}_x$, thus $\sec(0) = 1$. The second case considers the two sectors in clockwise order from $\mathcal{A}^i$, where the closest radius corresponds to the sector border of $\mathcal{A}^j$ that has angle $(j+1)\pi/4$. The third case handles the two sectors in anticlockwise order that have the closest border with angle $j\pi/4$. The forth case handles the sectors not facing l, that are unrestricted.

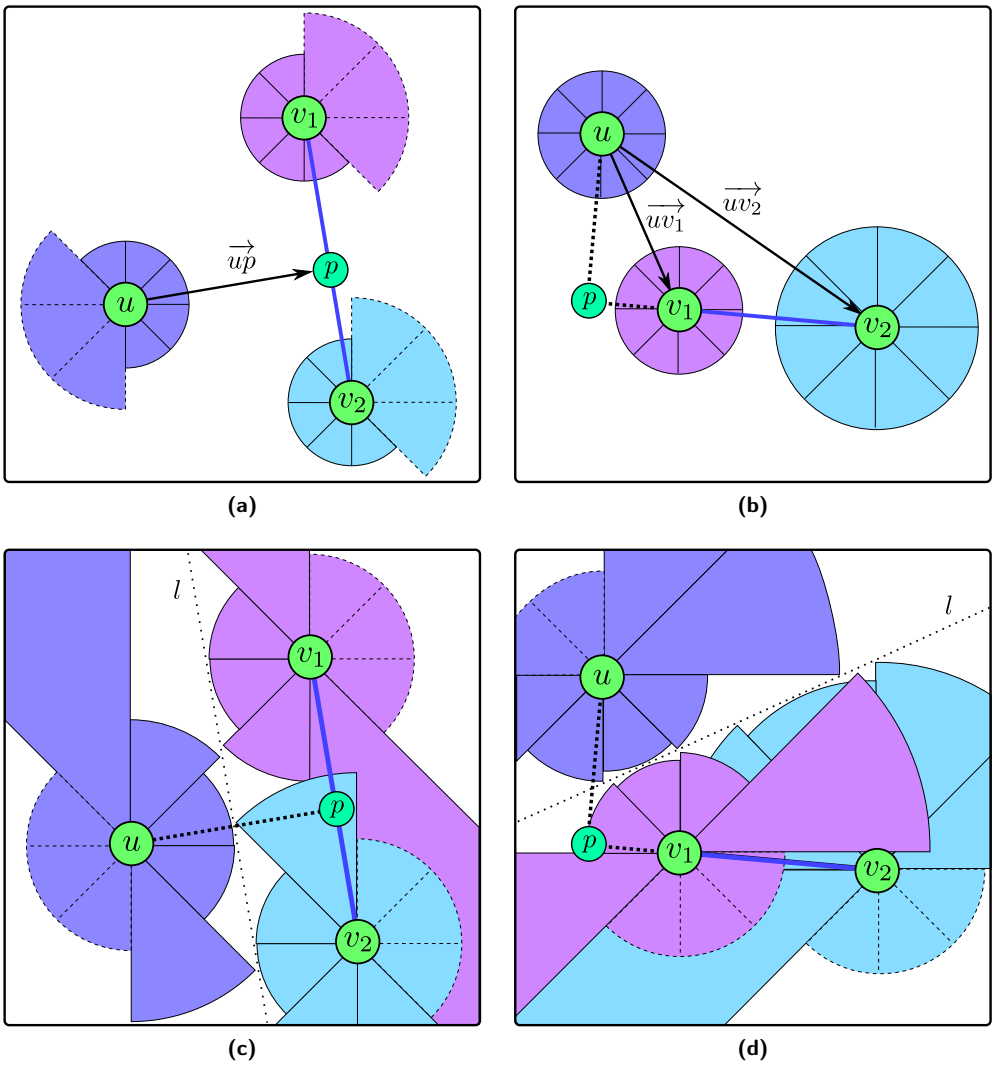


Figure 6.13: Restriction of the node movement in `ImPrEd` and comparison with `PrEd`. The dashed sectors are not reduced by the restriction operation. **(a-b)** Restriction in `PrEd` (see figure 6.3). **(c)** Case "projection inside" in `ImPrEd`. **(d)** Case "projection outside" in `ImPrEd`.

Correctness of the New Rules. To prove the correctness of the algorithm, we first prove that a node cannot cross an edge while moving. Given a node u and a non-incident edge e, the line l divides the plane into two halves: the first, $R_u(u, e)$, contains u and the second, $R_e(u, e)$, contains $e = [v_1, v_2]$ (see figure 6.14).

Let R_u be the intersection of all half-planes (polytope) computed for u and any non-adjacent edge $e \in E$:

$$R_u = \bigcap_{\substack{e \in E \\ u \neq s(e) \\ u \neq t(e)}} R_u(u, e)$$

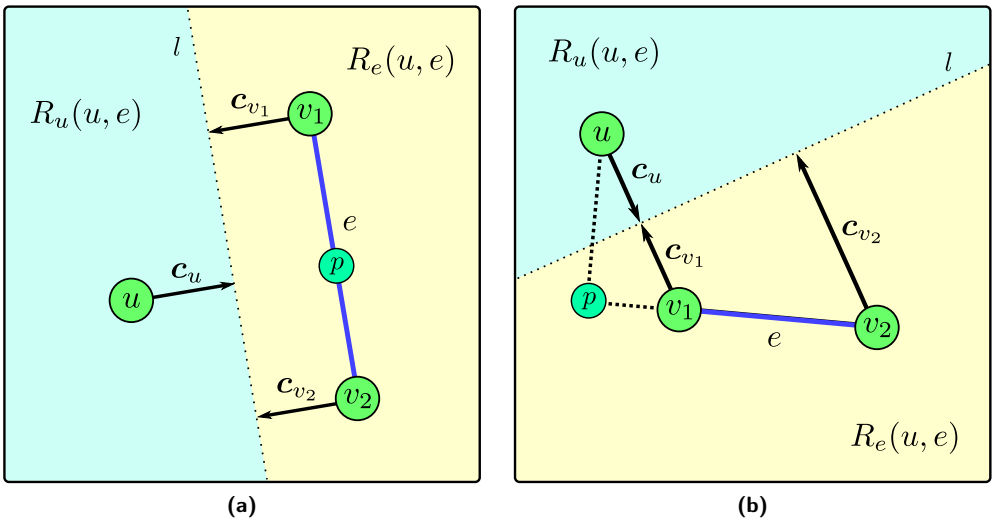


Figure 6.14: Plane division and collision vectors in the node movement restriction. **(a)** Case "projection inside". **(b)** Case "projection outside".

Similarly, let R_e be the intersection of all half-planes computed for e and any non-adjacent node $u \in V$:

$$R_e = \bigcap_{\substack{u \in V \\ u \neq s(e) \\ u \neq t(e)}} R_e(u,e)$$

The new formulae for the movement restriction bind the sectors around every node u to be in R_u. At the same time, they bind the movement of the extremities of an edge e to be in R_e. Therefore, R_u contains any future position of any node u and R_e, being convex (as the intersection of convex polytopes is convex), fully contains any final position of the edge e. Since for every node v and every non-incident edge e we have that $R_u \subseteq R_u(u,e)$, $R_e \subseteq R_e(u,e)$, and $R_u(u,e) \cap R_e(u,e) = \emptyset$, no node can cross an edge when moving (see figure 6.15).

The property also guarantees that, given two edges, no crossings can be incurred or undone since both cases require a node to cross an edge. Finally, $R_u(u,e)$ and $R_e(u,e)$ can always be computed, so long as u and e are not initially collinear, as we are working in a continuous space and nodes and edges cannot become collinear during the computation.

6.2.7 Crossable and Flexible Edges

In `ImPrEd`, we added the possibility of labelling each edge of the input graph as crossable or uncrossable, and as flexible or rigid. Furthermore, the class of input drawings was extended to polyline multigraphs, allowing edges to have bends.

An edge labelled as crossable will be treated by `ImPrEd` as it would be in a standard force-directed algorithm: it contributes to the computation of the resulting forces, but it does not impede the movement of the nodes. In particular, this means that a

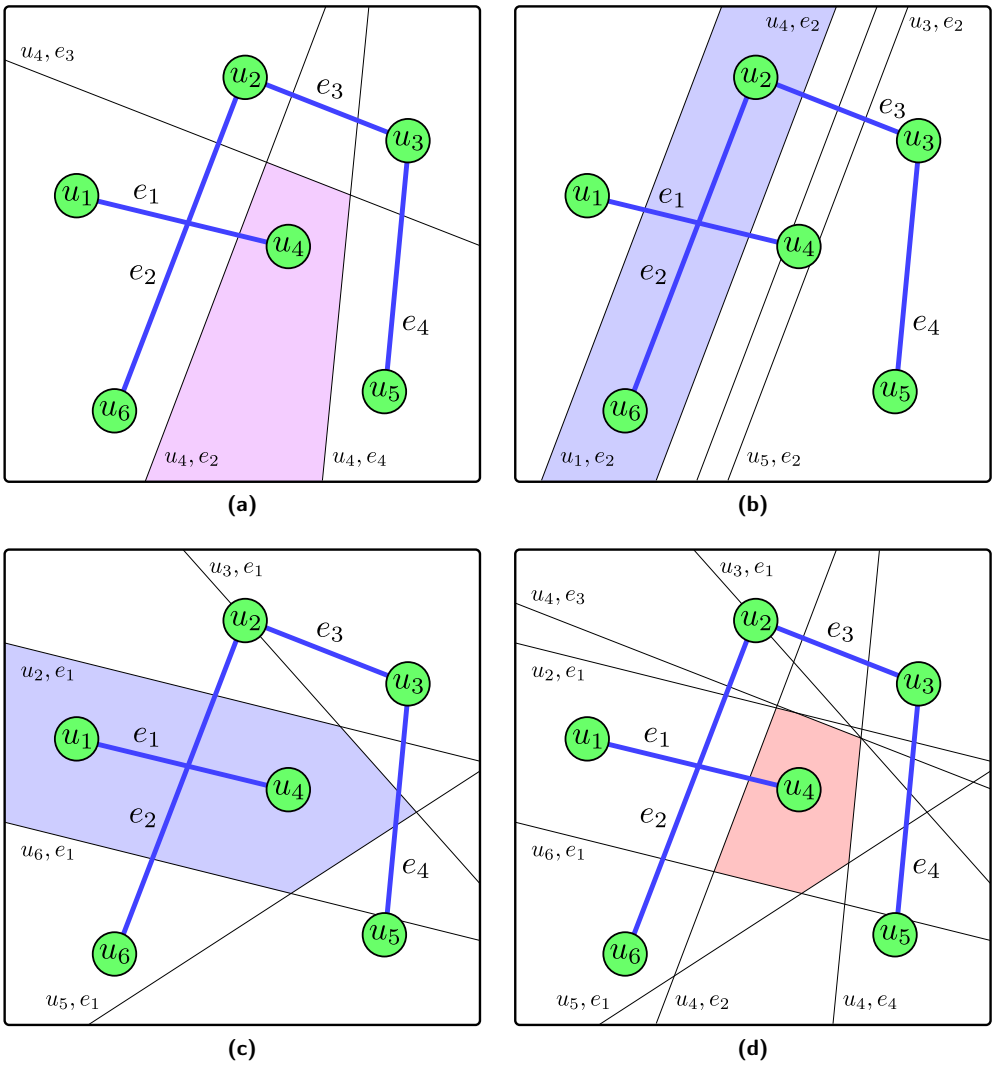


Figure 6.15: Node and edge polytopes in the node movement restriction. **(a)** Determination of a node polytope, R_{u_4}, in a sample graph. **(b)** Identification of an edge polytope, R_{e_2}, in the same graph. The absence of crossings between nodes and edges is guaranteed by the absence of overlaps between edge polytopes R_e and node polytopes R_u, where u is not an extremity of e. **(c)** Note that R_u does not necessarily correspond to the region where u can move: the node is also bounded as an extremity of its adjacent edges. The shaded region correspond to R_{e_1}, the edge polytope of the only adjacent edge of u_4. **(d)** The movement region of u_4 will be contained in the intersection between R_{u_4} and R_{e_1}.

crossable edge will not be considered in the computation of the maximal movement and in the computation of the node-edge repulsion forces, as this would also obstruct the node movement.

Uncrossable edges, instead, act like standard PrEd edges: they fully contribute to the resulting forces and they restrict the node movement to prevent crossings during the movement phase.

Edges labelled as flexible have the possibility of increasing or decreasing their original number of bends in order to reduce the stress exerted on them. An edge which is very stretched might decrease the magnitude of the forces acting on it by having more freedom in the shape it can assume, and might therefore benefit from the insertion of an additional bend. On the contrary, an edge with many bends that is very contracted generally benefits from the removal of some of its bends.

Rigid edges are instead polyline edges that are not allowed to change their number of bends during the computation. To replicate the behaviour of PrEd, it is therefore sufficient to mark all edges as rigid.

Extension of the Input Class. In order to benefit the most from flexible and rigid edges, the input class of ImPrEd is extended to multigraphs with polyline drawing. At the beginning of the computation, the polyline edges are converted into sequences of straight-line edges and nodes. The newly introduced nodes and edges have the same effect on the computation as the original elements of the graph.

At the end of the computation, the sequences of nodes and edges relative to a polyline edge are re-converted into a bent edge.

Crossable and Uncrossable Edges. Crossable edges are removed during the computation of the surrounding edges (see section 6.2.4). They will therefore not be considered in the computation of the maximal movement and in the computation of the node-edge repulsive forces. Uncrossable edges are instead treated normally.

Flexible and Rigid Edges. During the refresh phase, flexible edges are scanned to verify whether they should be contracted, expanded, or left unchanged.

Let $u_0, e_1, u_1 \dots u_n$ be a sequence of nodes and edges corresponding to a flexible edge. If two nodes u_i, u_{i+2} are at a smaller distance than 2δ, and if no other node is in the area enclosed in the triangle that has u_i, u_{i+1}, u_{i+2} as vertices, the flexible edge is contracted. When this happens, u_{i+1} and the adjacent edges are removed and the nodes u_i and u_{i+2} are connected with a new edge (see figures 6.16a–6.16c).

If two nodes u_i, u_{i+1} are at a distance greater than 3δ, the flexible edge is expanded. When this happens, a new bend u_{n+1} is inserted in the middle point of e_i, and the edge e_i is substituted by the path $u_i, e_{n+1}, u_{n+1}, e_{n+2}, u_{i+1}$ that includes two new edges (see figures 6.16d–6.16f).

If the previous conditions do not apply, the flexible edge is left unchanged. Rigid edges are not considered during this phase.

Notes on Flexible Edges. To obtain the best results with flexible edges, we remove the node repulsion force between consecutive nodes on a flexible edge path with one or more bends. As a consequence, the edge attractive force exerted by

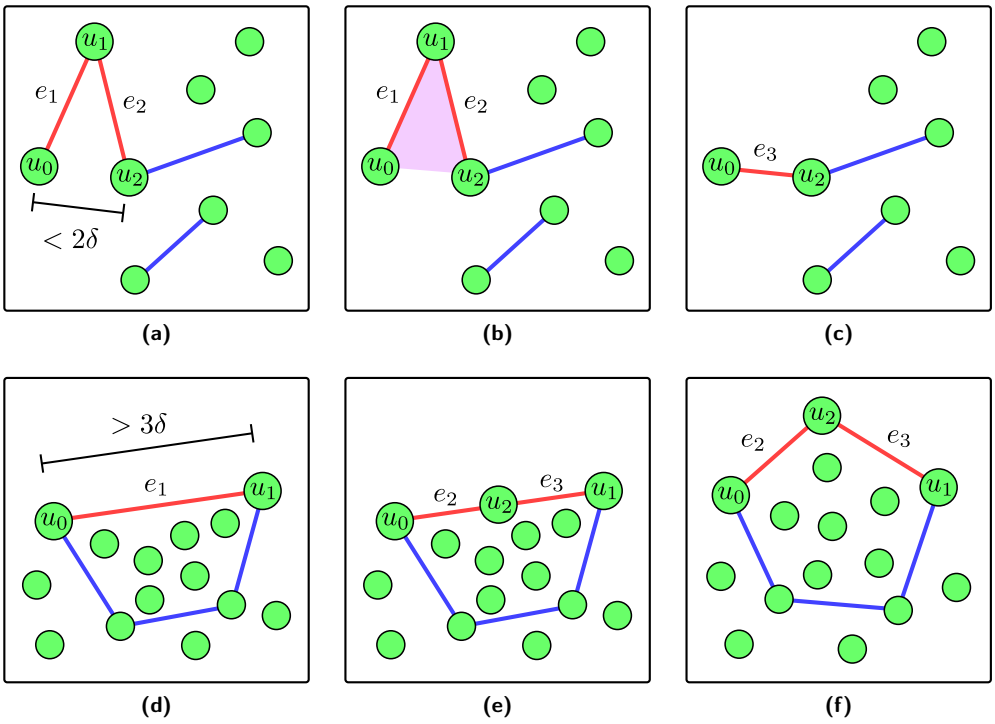


Figure 6.16: Contraction and expansion of flexible edges. The flexible edge is coloured red. **(a)** A case of a flexible edge with too many bends. Two nodes u_i, u_{i+2} at a geometrical distance lower than 2δ are potential extremities for the contraction of the edge. **(b)** The contraction is executed only if no other node is inside the triangle formed by u_i, u_{i+1}, u_{i+2}. **(c)** The elements e_1, e_2 and u_1 are replaced by the new edge e_3. Note that the operation cannot alter the edge crossing properties once the previous control is performed. **(d)** A case of a stretched flexible edge. The expansion of the edge is executed if two consecutive nodes are at a greater distance than 3δ. **(e)** The edge e_1 is replaced by the elements e_2, e_3 and u_2. Note that the operation cannot alter the edge crossing properties since the shape of the edge is not altered. **(f)** The new bend reduces the stress on the flexible edge increasing its optimal length and allowing it to assume more complex shapes.

a flexible edge segment is not balanced and acts like a tension force that helps remove unnecessary bends.

Also, because of the expansion mechanism, we advise not to use flexible edges that cross each other. A new bend might be placed in a very close position to the crossed edge, introducing great instability in the graph drawing and preventing the convergence to a stable configuration. Therefore, two crossing edges or two crossable edges that might end up overlapping *should not* be marked as flexible.

6.2.8 Weight of Nodes and Edges

In `ImPrEd`, it is possible to assign a weight to the nodes, so that the more important ones have higher influence in the resulting forces and lower tendency to change their

position. As a result, high weight nodes tend to determine the overall structure of the graph, and low weight nodes tend to assume a position that fits this configuration.

This effect is obtained by further modifying the formulae for the force computation and the node movement. The following box reports the new formulae with the modifications (with respect to those in section 6.2.3) highlighted in colour.

Modifications in the Force Formulae. The formulae that consider the weight of the elements are

$$\mathcal{F}_u^r(u,v) = \left(\frac{\delta \cdot \sqrt{\mathcal{W}(u)\mathcal{W}(v)}}{\|\boldsymbol{p}_u - \boldsymbol{p}_v\|}\right)^{x^r} (\boldsymbol{p}_u - \boldsymbol{p}_v)$$

$$\mathcal{F}_u^a(e) = \left(\frac{\|\boldsymbol{p}_u - \boldsymbol{p}_v\|}{\delta \cdot \mathcal{W}(e)}\right)^{x^a} (\boldsymbol{p}_v - \boldsymbol{p}_u)$$

$$\mathcal{F}_u^e(u,e) = \begin{cases} \left(\dfrac{\gamma - \|\boldsymbol{p}_u - \boldsymbol{p}_p\|}{\|\boldsymbol{p}_u - \boldsymbol{p}_p\|} \cdot \sqrt{\mathcal{W}(u)\mathcal{W}(e)}\right)^{x^r} (\boldsymbol{p}_u - \boldsymbol{p}_p) & \text{if } \|\boldsymbol{p}_u - \boldsymbol{p}_p\| < \gamma \\ \mathbf{0} & \text{otherwise} \end{cases}$$

In the optimisation of Euler diagrams, we use this feature to ensure that the elements of the grid graph have higher influence on the drawing than the elements of the original graph. In fact, the latter should adapt and fit the shape of the diagram more than radically change it.

Also, we might want to assign a weight to the grid graph edges that increases with the size of the diagram. When the diagram is complex, a larger distance of the elements from the borders of the zone regions might increase the aesthetics and the readability of the drawing.

6.3 Results

In this section, we conclude by evaluating the impact of the newly introduced feature on the aspects we aimed to improve. In particular, we compare `PrEd` and `ImPrEd` on the complexity, quality of the resulting drawing, running times, reliability of the input parameters and control over the output. Since the introduction of flexible edges significantly alters the behaviour of the algorithm, we run the benchmarks both considering all edges marked as rigid (standard behaviour of `PrEd`) and considering all edges marked as flexible. We refer to the first case with `ImPrEd` (R), and to the second case with `ImPrEd` (F).

The tests are executed on two data sets of different nature. The first data set is composed of a large number of randomly generated graphs. We generated 10 random plane graphs for each of the following dimensions: 50 nodes and 144 edges, 75 nodes and 204 edges, 100 nodes and 294 edges, 150 nodes and 444 edges, 200 nodes and 594 edges. Then, we generated 10 additional non-plane graphs by randomly adding 20 crossing edges to the previous graphs. In all, we tested 100 randomly generated graphs.

We also compared `PrEd` and `ImPrEd` on the generation of Euler representations, for both graphs optimised in the procedure:

- *zGraph*, the zone graph G^z, target of the first graph optimisation,
- *gGraph*, the graph formed by the grid graph G^g and the original graph G^o, target of the second graph optimisation.

The graphs are used to generate two diagrams that will be discussed in greater detail in section 7.2. In particular, *zGraphA* and *gGraphA* are the graphs optimised in the generation of the *IMDb 60* diagram, that shows the intersections between the casts of sixty films and features more than 2000 actors. The graphs *zGraphB* and *gGraphB* are built in the generation of the *gene interaction* diagram, that shows an interaction network of 170 genes grouped into ten clusters.

6.3.1 Complexity

The complexity of `ImPrEd` is mostly unchanged with respect to `PrEd`.

The complexity of an iteration of `PrEd` is dominated by the force computation, that requires $O(|V|^2 + |E||V|)$ operations. The insertion of surrounding edges and QuadTrees reduces the complexity of the general case, as the number of nearby elements and of surrounding edges is typically much smaller than $|V|$ and $|E|$. However, these features do not reduce the worst case complexity of an algorithm iteration.

The insertion of the pre-processing step might instead cause an increase of the total complexity. The complexity of the surrounding edges computation depends on the initial layout configuration. If G is a plane graph, each edge is considered twice. Since the number of edges and faces is linear with respect to the number of nodes, the overall complexity is $O(|V|)$.

If G is not a plane graph, in the worst case, we have $O(|E|^2)$ crossings. As each crossing produces at most one node and at most two segments and outputs a plane graph, the overall worst case complexity is $O(|V| + |E|^2)$. Here there appears a quadratic term in the number of edges that was not present before. It is important to note, however, that this is a pre-processing step and does not influence the main cycle of the algorithm.

6.3.2 Execution Time

To test the algorithm's performance, we executed `PrEd`, `ImPrEd` (R) and `ImPrEd` (F) on the graphs of the two data sets. For each graph, we applied the algorithms with 100, 250 and 500 iterations of the main cycle. Also, for each combination of graph, algorithm and number of iterations, we executed the algorithm ten times and computed the average. The average running times for the randomly generated graphs are collected in table 6.1, and those of the Euler diagram graphs are collected in table 6.2.

The data obtained for the randomly generated graphs show that `ImPrEd` (R) is consistently faster than `PrEd` and that the speed-up is related to the size of the graph. On the smallest graphs, 50 nodes, `ImPrEd` (R) is five times faster than `PrEd`. On the largest graphs, 200 nodes, `ImPrEd` (R) is more than twenty times faster. On the other hand, on the same graphs `ImPrEd` (F) is consistently twice as slow as `PrEd` for all graph sizes, with only small variations. Figure 6.17 shows these results as plots of the average running times per iteration.

Also on the Euler diagram graphs `ImPrEd` (R) performs up to twenty times faster than `PrEd`. However, this time `ImPrEd` (F) outperforms `ImPrEd` (R) and is up to thirty times faster than `PrEd` (see figure 6.18). This suggests that the performances of `ImPrEd` (F) heavily rely on the characteristics of the input graph and on its initial

Random Plane Graphs						
Nodes		50	70	100	150	200
Edges		144	204	294	444	594
PrEd	100	6.68	14.34	31.11	72.67	132.16
	250	16.71	35.85	77.59	181.91	330.40
	500	33.38	71.73	155.21	363.40	661.39
ImPrEd (R)	100	1.20	1.84	2.93	4.69	6.87
	250	2.97	4.50	7.07	11.07	16.14
	500	5.94	8.97	12.95	21.79	31.59
ImPrEd (F)	100	11.79	23.16	52.54	103.06	177.66
	250	34.61	73.29	172.59	350.47	631.05
	500	78.34	157.64	380.02	792.97	1439.32
ImPrEd (PP)	—	0.04	0.09	0.18	0.41	0.73
ImPrEd (R)	Gain	5.60	7.92	11.19	16.20	20.21
ImPrEd (F)	Gain	0.49	0.52	0.48	0.56	0.58

(a)

Random Non-Plane Graphs						
Nodes		50	70	100	150	200
Edges		164	224	314	464	614
PrEd	100	7.66	15.83	33.98	76.40	137.72
	250	19.13	39.57	84.85	191.08	344.90
	500	38.30	79.20	169.83	382.42	689.24
ImPrEd (R)	100	1.33	1.86	3.54	4.95	6.76
	250	3.21	4.47	8.49	11.65	15.59
	500	6.35	8.87	16.74	22.79	30.31
ImPrEd (F)	100	—	—	—	—	—
	250	—	—	—	—	—
	500	—	—	—	—	—
ImPrEd (PP)	—	0.09	0.14	0.26	0.52	0.87
ImPrEd (R)	Gain	5.92	8.76	9.91	16.20	21.75
ImPrEd (F)	Gain	—	—	—	—	—

(b)

Table 6.1: Average running times of PrEd and ImPrEd for 100, 250 and 500 iterations of the main algorithm cycle, over the randomly generated graphs. ImPrEd (PP) is the preprocessing required before ImPrEd can execute. Green numbers indicate improvements, red numbers indicate slower execution. **(a)** Random plane graphs. **(a)** Random non-plane graphs. Here, ImPrEd (F) has not be applied as flexible edges should not cross.

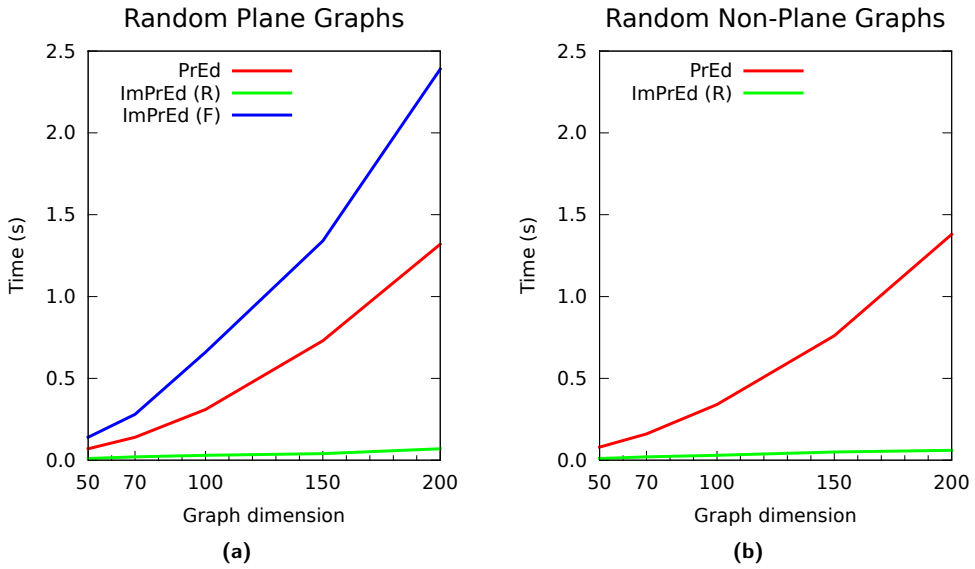


Figure 6.17: Plot of the average running time per iteration on random graphs. **(a)** Plot for random plane graphs. **(b)** Plot for random non-plane graphs. As explained above, `ImPrEd` (F) is not present as flexible edges should not cross.

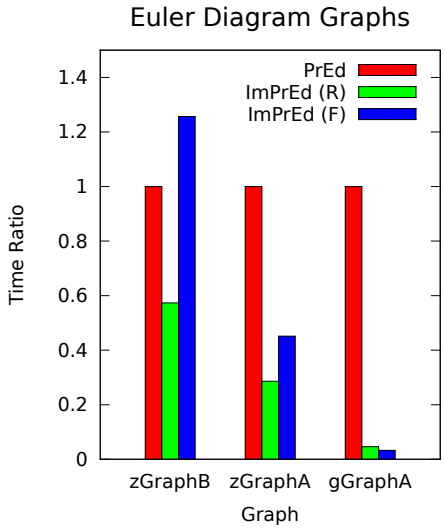


Figure 6.18: Graphical comparison of the execution times for the Euler diagram graphs. The average running time per iteration for these graphs has been normalised to the running time of `PrEd`, and the graphs have been ordered by dimension. Again, the performance of `ImPrEd` improves with the increase of the graph dimension.

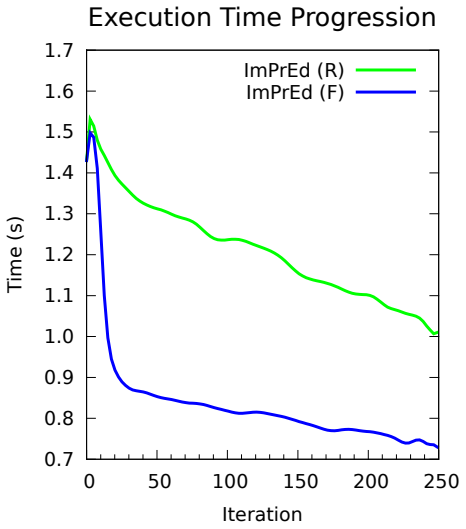


Figure 6.19: Plot of the running time as a function of the progress of the computation. The times are relative to the execution of `ImPrEd` on gGraphA with 250 iterations. The plot shows how flexible edges can drastically reduce the running time, starting from the very first iterations, when the graph has an oversized number of bends.

Euler Diagram Graphs					
Graph		zGraphA	gGraphA	zGraphB	gGraphB
Nodes		149	2601	23	220
Edges		169	507	22	362
PrEd	100	17.50	2721.79	0.51	—
	250	40.91	6792.53	1.28	—
	500	80.46	13 585.36	2.53	—
ImPrEd (R)	100	4.90	131.55	0.30	15.65
	250	11.81	308.83	0.73	33.49
	500	23.34	597.55	1.42	65.34
ImPrEd (F)	100	6.68	97.22	0.47	9.25
	250	22.02	215.62	1.73	18.07
	500	35.29	409.18	3.79	30.81
ImPrEd (PP)	—	0.13	5.64	0.02	0.14
ImPrEd (R)	Gain	3.49	21.81	1.75	—
ImPrEd (F)	Gain	2.25	30.90	0.73	—

Table 6.2: Average running times of PrEd and ImPrEd for 100, 250 and 500 iterations, over the graphs in the Euler diagram generation data set. ImPrEd (PP) is the pre-processing step. To gGraphB, we only applied ImPrEd as the input includes edges marked as crossable, which cannot be handled by PrEd. Green numbers indicate improvements, red numbers indicate slower execution.

configuration. Since the randomly generated graphs have a high node-to-edge ratio, ImPrEd (F) tends to insert a high number of bends, considerably slowing the execution. On the other hand, on the Euler diagram graphs, ImPrEd (F) is able to reduce the high number of initial bends, effectively reducing the size of the input (see figure 6.19).

On both the random and Euler diagram data sets, pre-processing time is, for the most part, negligible. On the smallest graphs, it accounts for only 5% to 10% of the total running time. On the larger graphs, it accounts for only 1% of the total running time.

6.3.3 Drawing Quality and Parameter Reliability

At first, we illustrate the behaviour of flexible edges in cases of clear oversized and undersized edges. According to the results shown in figure 6.20, the contraction and expansion policies seem to be correctly devised, since flexible edges nicely respond to the stress exerted by the other elements of the graph.

Figure 6.21 shows, in a second example, how ImPrEd generates a final diagram with a better spacing and with higher reliability of the input parameters δ and γ. The spacing is more uniform and corresponds better to the interpretation of δ as the minimal desired node-node distance, and of γ as the minimal desired node-edge distance. This is very important in the generation of Euler diagrams, since δ allows to control the spacing between the elements, and γ the distance between elements and cluster boundaries.

We now proceed to the comparison of the results obtained by PrEd and ImPrEd on several graphs. Figure 6.22 shows the results of PrEd and ImPrEd (R) on an example presented by Bertault [6, Figure 1]. ImPrEd seems to reach a more stable configuration

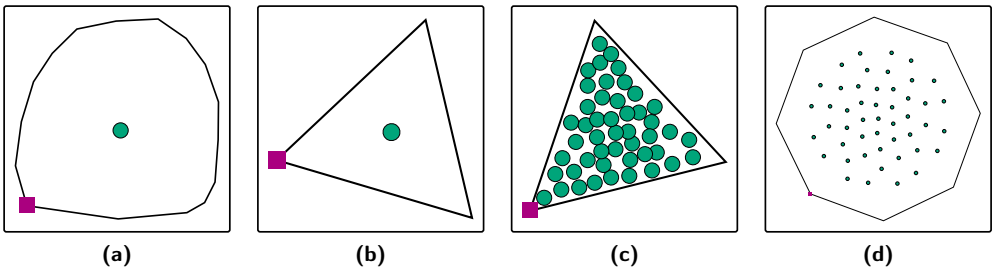


Figure 6.20: Example of contraction and expansion of flexible edges. The picture shows a flexible loop containing isolated nodes. **(a)** The loop having an initial configuration with an oversized number of bends. **(b)** The result of the application of `ImPrEd`. **(c)** The loop having an initial configuration with an undersized number of bends. **(d)** The result of the application of `ImPrEd`.

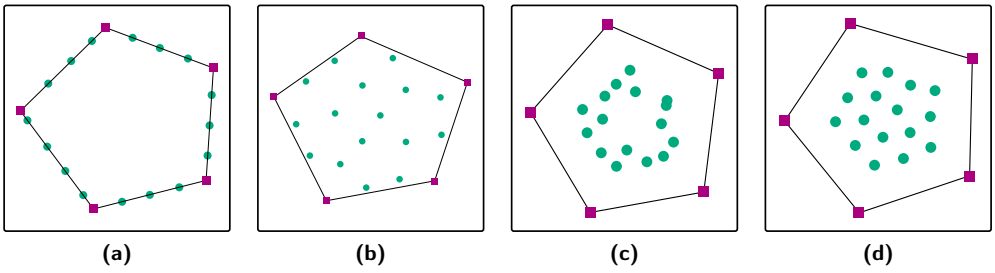


Figure 6.21: Comparison of the reliability of the parameters δ and γ. It is possible to see how `ImPrEd` generates results where the spacing better respects the interpretation of the mentioned parameters. **(a)** The results of `PrEd` with parameters $\delta = 5$, $\gamma = 2$. **(b)** The results of `ImPrEd` with the same parameters. **(c)** The results of `PrEd` with parameters $\delta = 2$, $\gamma = 5$. **(d)** The results of `ImPrEd` with the same parameters.

with the same number of iterations, possibly because of the new movement rules. If the convergence of the algorithm is accelerated as the image suggests, it is either possible to reduce the number of iterations and obtain lower execution times, or maintain the same number of iterations to obtain higher drawing quality.

Figure 6.23 shows the results produced by the algorithms on a planar graph of the random data set. `ImPrEd` (R) can produce a better spacing than `PrEd`, but the high density of the graph prevents it from obtaining a readable drawing. By marking all edges as flexible, and therefore by moving from a straight-line to a polyline drawing style, we can obtain a better spacing of the nodes and a better angular resolution.

Figure 6.24 shows the results of the algorithms on zGraphA and figure 6.25 on gGraphA. `PrEd` produces drawings where the nodes seem forced far from the centre. In particular, figure 6.25b illustrates that nodes are distributed along region boundaries, probably because the graph induces few attractive and many repulsive forces acting on disconnected nodes. This shows how the original forces of `PrEd` are hardly usable for graphs with a large number of disconnected nodes. In fact, to obtain appreciable results in the generation of Euler diagrams with `PrEd` [91], we had to carefully select the algorithm parameters and increase the repulsive force exponent. However, despite these efforts, the nodes tend to accumulate on the cluster borders.

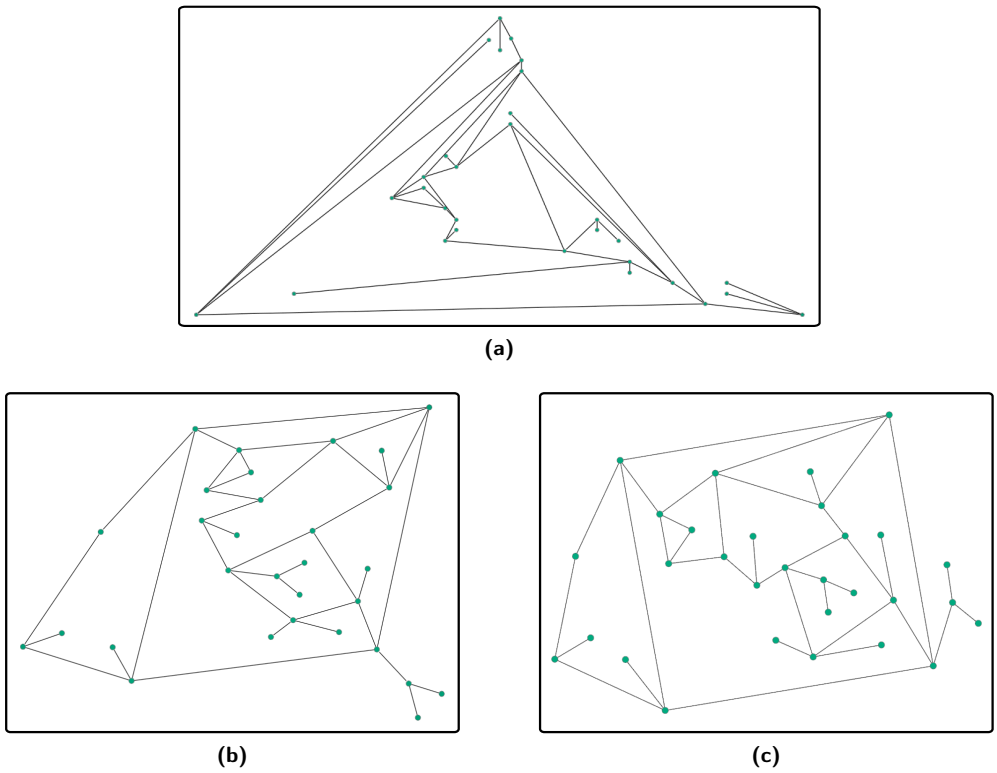


Figure 6.22: Comparison of the drawing quality on an example of `PrEd`'s original article [6, figure 1]. **(a)** The initial configuration of the graph. **(b)** The results of the application of 100 iterations of `PrEd`, as presented in the article. **(c)** The result or the application of 100 iterations of `ImPrEd` (R) on the initial configuration. Although both drawings have high aesthetics, the configuration reached with `ImPrEd` seems to be closer to a stable one.

Figures 6.24c and 6.25c show how the force and movement cooling of `ImPrEd` helps to obtain a more regular node distribution and an element spacing closer to the chosen parameters. Finally, figures 6.24d and 6.25d show how the introduction of flexible edges can help improve the aesthetics of the drawings. In the first figure, the angular resolution of the graph is increased thanks to the insertion of bends in the most stressed edges. In the second figure, the extension and contraction of the flexible edges further improve the node distribution and lead to smoother cluster boundaries.

Figure 6.26 shows the result of the application of `ImPrEd` (F) to gGraphB. Here, the edges of the original graph are defined as rigid and crossable, and the edges that form the cluster boundaries are defined as flexible and uncrossable. This allows us to take the original graph edges into consideration while keeping them straight-line and preventing them from obstructing the node movement. At the same time, it allows us to have uncrossable boundaries that can grow and shrink as needed.

Even though the drawing is mainly shaped by the boundary nodes and edges that had higher weight assigned there is a clear contribution of the original graph edges. First, the cluster elements with adjacent nodes in different regions are pulled towards their neighbours, assuming a position that reduces the edge length and cluttering. Second, the sets themselves seem to minimise the length of the edges connecting nodes in different regions.

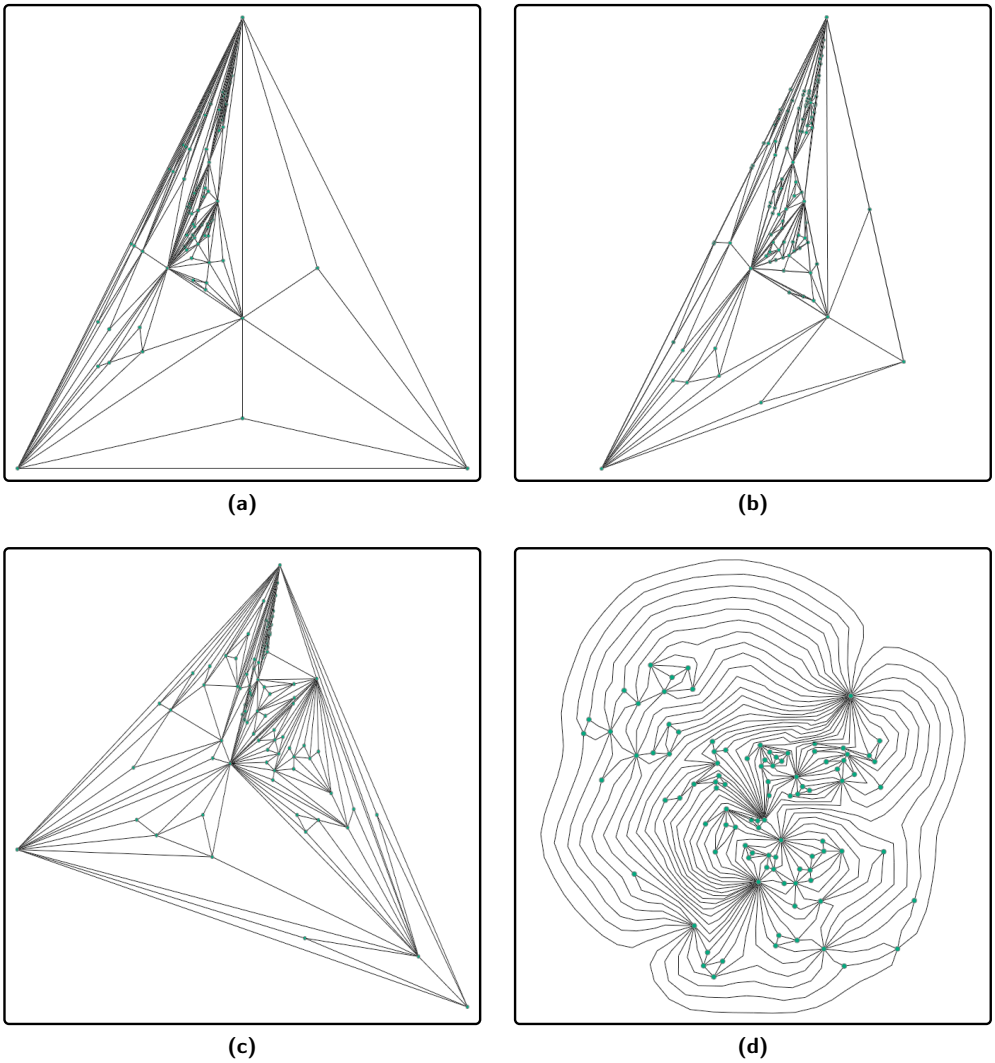


Figure 6.23: Comparison of the drawing quality on a random planar graph. All algorithms used the parameters $\delta = 10$, $\gamma = 10$, 250 iterations. **(a)** The initial configuration of the graph, provided by the generator of random graphs. **(b)** The results of the application of `PrEd`. **(c)** The result or the application of `ImPrEd` (R). **(d)** The result or the application of `ImPrEd` (F).

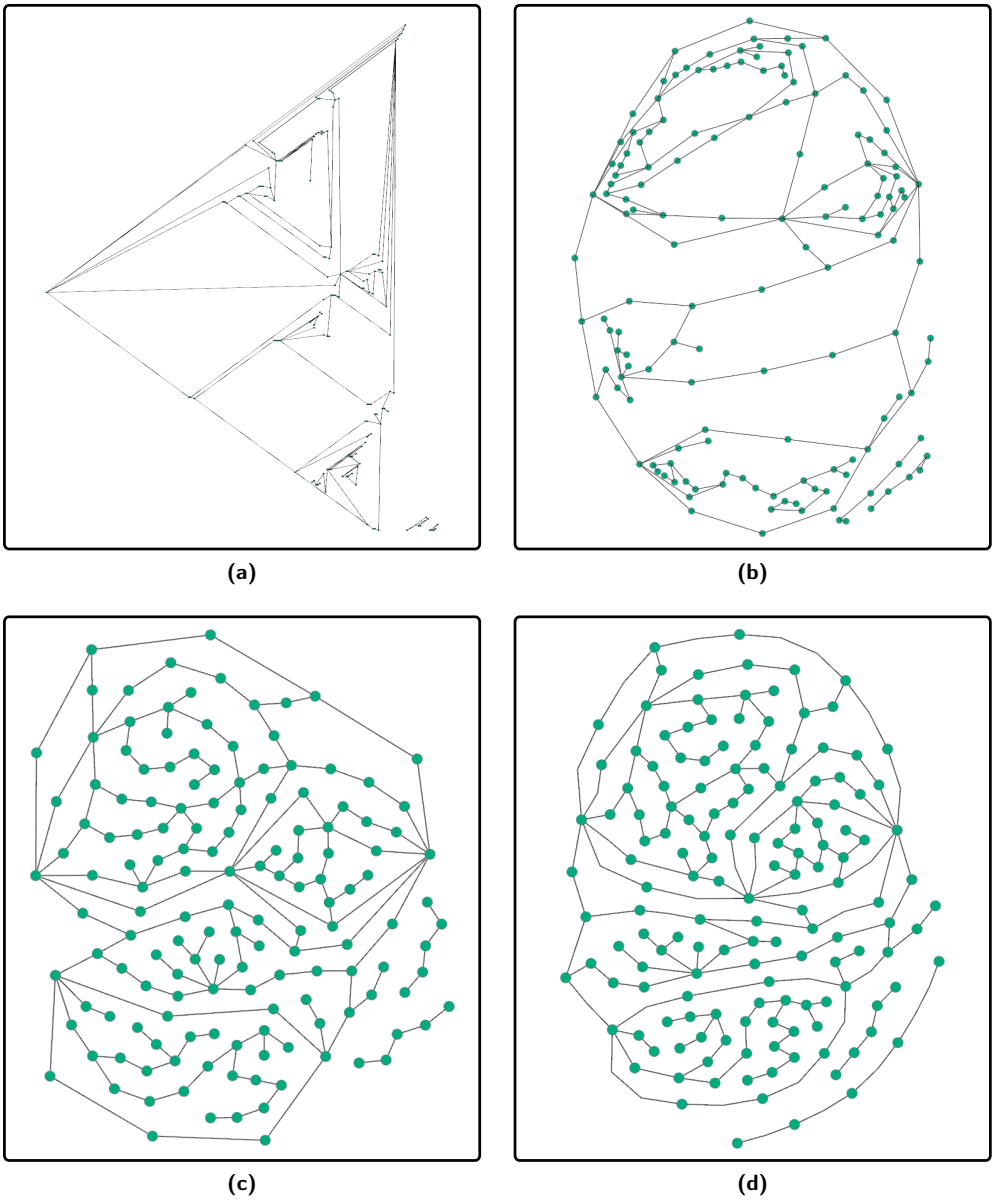


Figure 6.24: Comparison of the drawing quality on zGraphA. All algorithms used the parameters $\delta = 27$, $\gamma = 21$, 250 iterations. The node diameter is set to 10 to better show the graph proportions. All figures have been rotated 90° anticlockwise to better fit the page. **(a)** The initial configuration of the graph, provided by the Euler representation algorithm. **(b)** The results of the application `PrEd`. **(c)** The result or the application of `ImPrEd` (R). **(d)** The result or the application of `ImPrEd` (F).

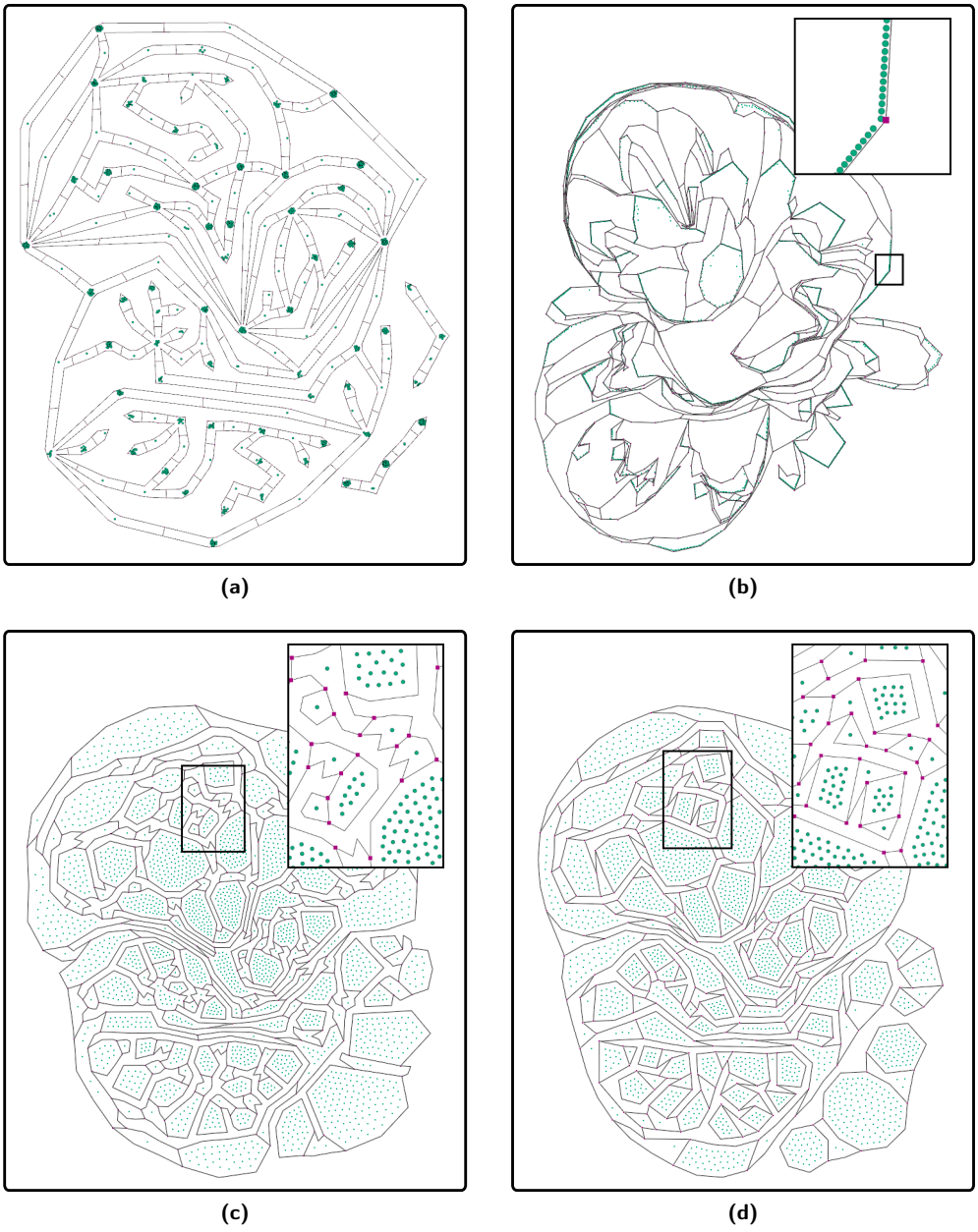


Figure 6.25: Comparison of the drawing quality on gGraphA. All algorithms used the parameters $\delta = 5$, $\gamma = 4$, 250 iterations. All figures have been rotated 90° anticlockwise to better fit the page. **(a)** The initial configuration of the graph, provided by the Euler representation algorithm. **(b)** The results of the application of `PrEd`. **(c)** The result of the application of `ImPrEd` (R). **(d)** The result of the application of `ImPrEd` (F).

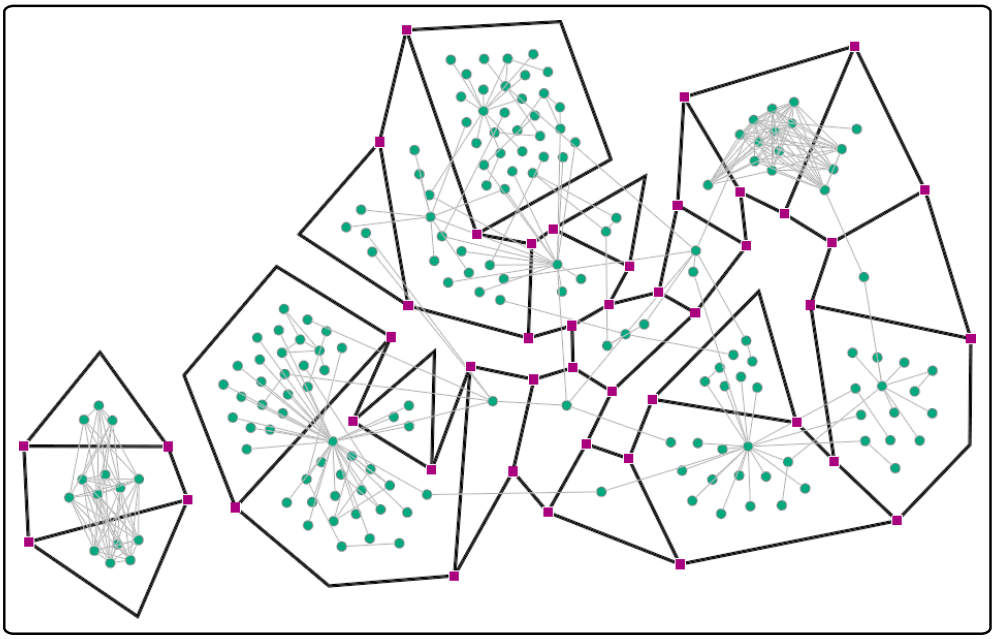


Figure 6.26: Application of `ImPrEd` (F) to gGraphB, that contains both flexible and crossable edges. The black edges are associated with the cluster boundaries, and were therefore marked as uncrossable and flexible. The grey edges are the edges of the clustered graph, and were marked as crossable and rigid.

Chapter 7

Software Implementation and Output Examples

In this chapter we present EulerView, a software tool that implements the generation method described in this thesis and that permits some basic interaction with the resulting diagram.

In the first section we introduce the software and the visualisation framework in which it is inserted. We will in particular illustrate some interaction features we decided to provide, as they allow to mitigate some of the limitations of Euler representations.

In the second section, we show some examples of Euler representations generated with this software. Each diagram is built from real world data, that will be presented along with the actual drawing.

7.1 EulerView

EulerView is a software tool that generates Euler representations according to the generation procedure described in the previous chapters. It is an extension of a graph drawing and data visualisation framework called Tulip [3, 4], which supports the creation of plug-ins to expand its initial set of functions.

EulerView is a *view plugin*, meaning it provides an alternative visualisation of the data under analysis. When the view is called on a clustered graph, a standard node-link diagram view is augmented with the graphical entities representing the cluster regions, and the graph layout is changed to reflect the position of the original elements in the diagram. This way, we obtain a drawing such as that shown in figure 7.1.

Interaction. Since we inherit the functions of the standard node-link diagram view of Tulip, the user will be able to perform all the classic operations available in a graph visualisation software, except those involving the movement of the nodes. These operations include, among others, the computation of metrics and characteristics of the graph or the enquiry or modification of the element label, shape and dimension.

Furthermore, we devised a few basic interaction features specific for Euler representations. Most features are oriented towards the shading of the portions of the diagram that are not of current interest, through the mechanism of the cluster and zone selection. We also insert tooltip panels that provide information on the clusters

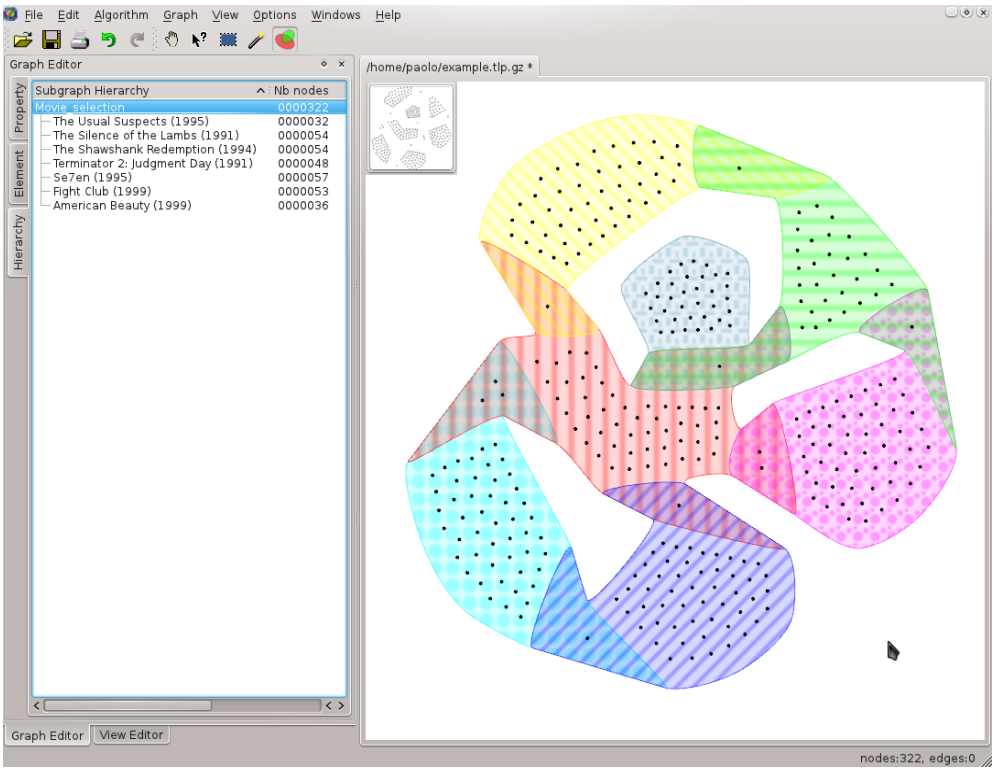


Figure 7.1: A screen shot of EulerView.

currently observed or selected. Finally, we provide the possibility of grouping the original elements contained in a zone into path-preserving meta-nodes.

7.1.1 Cluster and Zone Selection

The *current selection* is the selection executed with a single input action, such as a mouse click. The *performed selection* is instead the selection obtained as a result of a sequence of current selections. Once we have chosen to show only some of the diagram's clusters or zones, we just need to select them. The software shows the clusters or zones of the performed selection in full colours, and heavily shades the remaining ones. The latter remain still visible, but they do not produce significant distraction in the representation.

There are different ways of performing a current selection:

- *Cluster list selection*. The left panel of the software presents the list of the clusters in the diagram, along with information such as the number of elements contained and the number of intra-cluster edges. Once a cluster is selected, only the original graph nodes and edges contained in the cluster are shown, while the other elements are hidden (see figure 7.2).

  This selection is particularly useful to spot the position of a cluster in the diagram, and it is strongly recommended when the diagram contains more than about a dozen clusters.

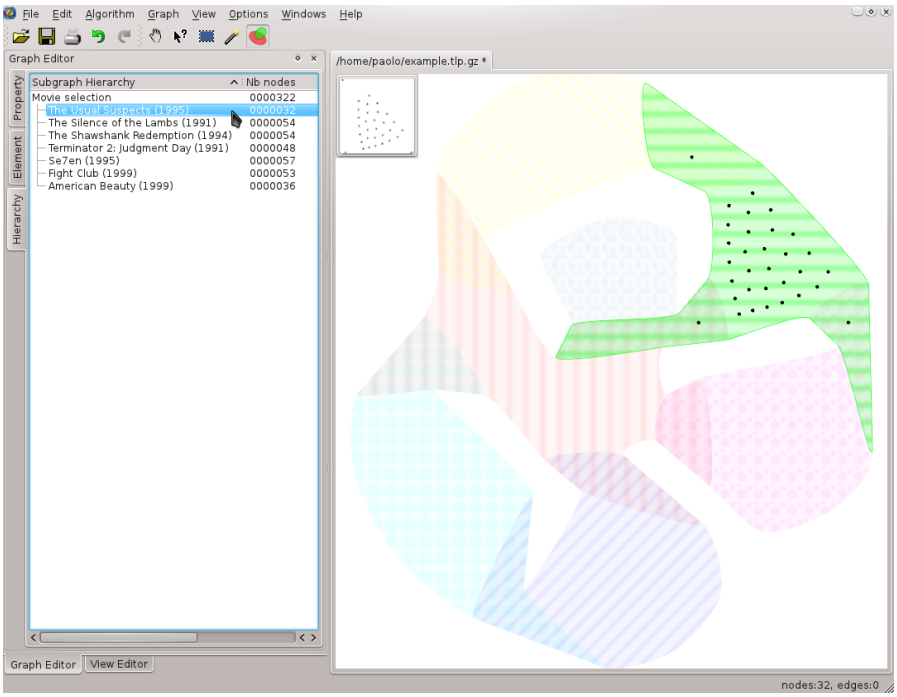


Figure 7.2: An example of list selection, performed by clicking on the cluster list on the left.

- *Cluster and zone simple selection*. By clicking on an area of the drawing containing a single cluster, we select that cluster (see figure 7.3). By double-clicking on any of the zones of the graph, we select that zone. Zones and clusters cannot be selected at the same time.

  This selection shows the actual shape of the cluster region of the diagram, that can be difficult to follow when the diagram presents complex cluster intersections.

- *Cluster multiple selection*. By clicking on an area of the diagram where multiple clusters overlap, we select all the clusters in the intersection (see figure 7.4).

  This selection is particularly useful when we want to identify the clusters involved in an intersection.

- *Spin cluster selection*. When a multiple selection has just been done, the cursor changes shape to indicate the possibility of performing a spin selection. By acting on the mouse wheel, it is possible to scroll through the clusters involved in the given intersection (see figure 7.5). The selection is confirmed by moving the cursor from the clicked point.

  This selection can be used to select a specific cluster in an overlapping region, or to show the individual clusters that intersect at a given point.

- *Cluster and zone global selection*. By clicking on an empty area of the diagram, we select all the clusters of the diagram. By double clicking on an empty area, we select all the zones in the diagram.

  This selection is particularly useful to reassign full colour to all clusters of the diagram, showing the Euler representation as it appears after the generation.

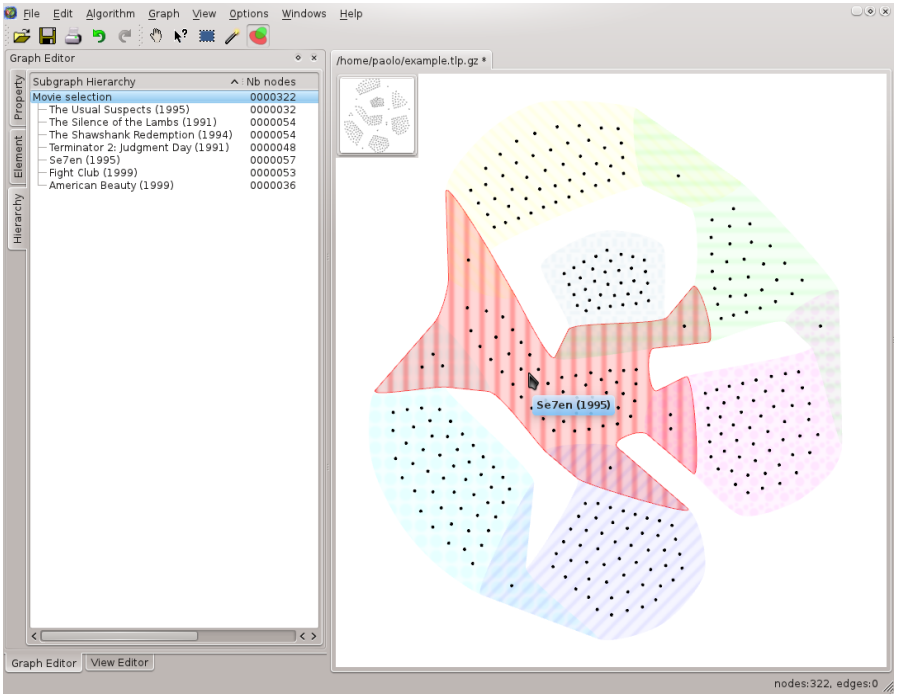


Figure 7.3: An example of simple selection, performed by clicking on a region that only contains the red cluster.

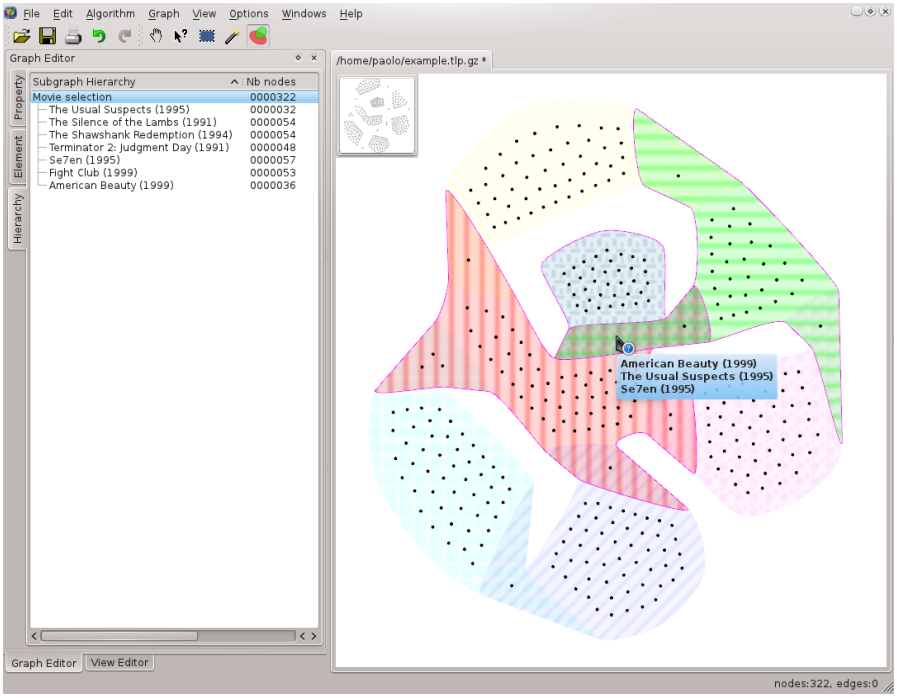
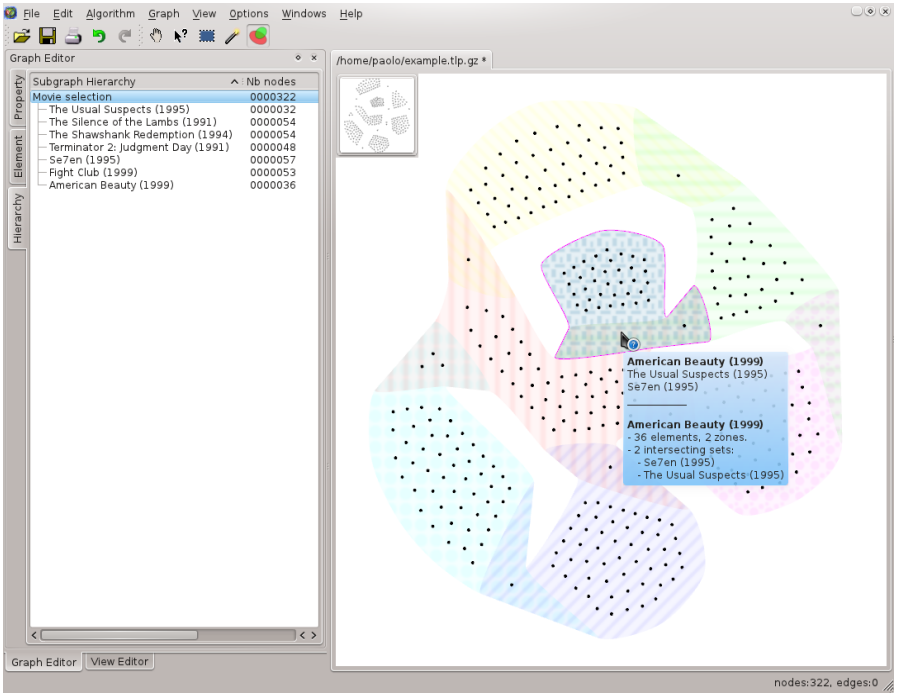
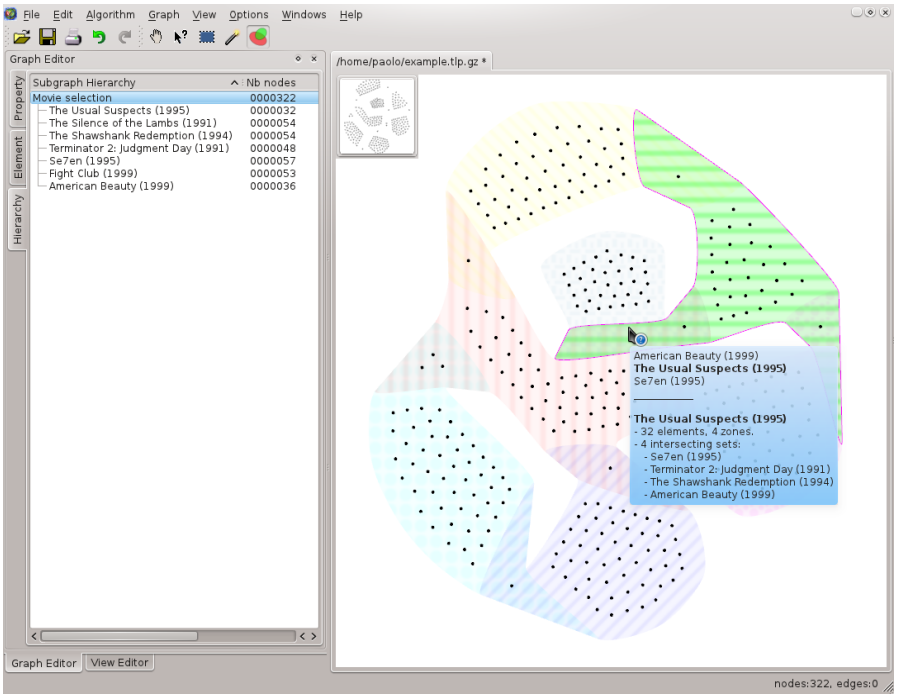


Figure 7.4: An example of simple selection, performed by clicking on the overlap between the highlighted clusters.



(a)



(b)

Figure 7.5: An example of spin selection. These selections are performed by using the mouse wheel on the multiple selection in figure 7.4. Each step of the mouse wheel moves the selection to the next (or previous) cluster in the list. **(a)** One step down. **(b)** Two steps down.

The way the current selection acts on the performed selection can be altered through the following modifier keys:

- *No modifier keys:* the current selection replaces the performed selection.
- *Shift key:* the current selection is added to the performed selection.
- *Control key:* the current selection is removed from the performed selection.

7.1.2 Tooltips

EulerView shows tooltips that are meant to facilitate the analysis of the diagram and of the information it contains. The tooltips contain information on the number of elements and zones contained in a given cluster, and information on the clusters that have inclusion or intersection relationships with it.

There are three types of tooltip:

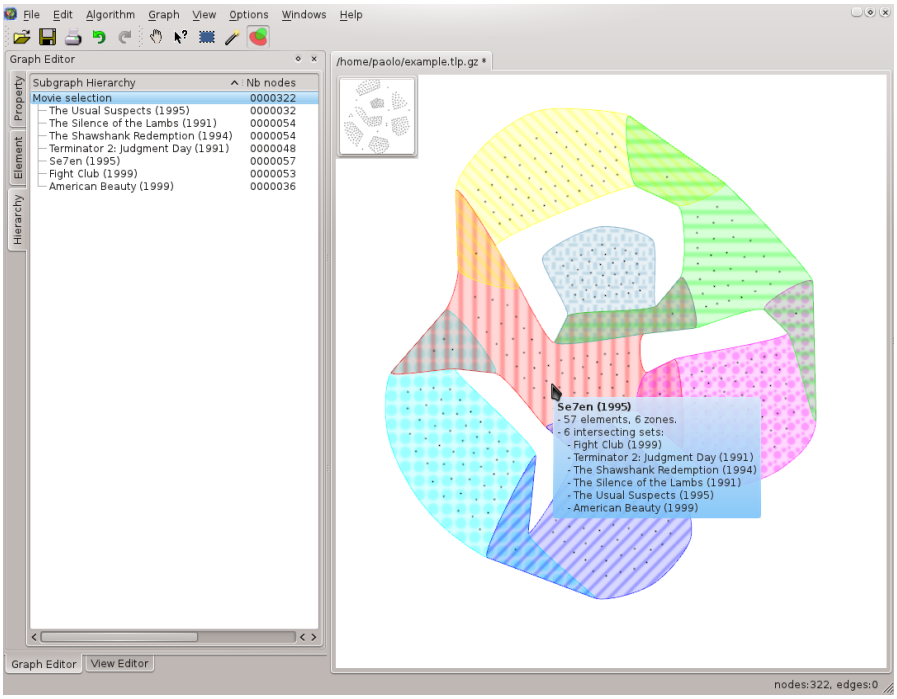
- *Selection tooltip*. This tooltip is shown when a simple or multiple selection is performed. The tooltip only contains the name of the selected clusters so as not to obstruct the view of the diagram (see figures 7.3 and 7.4).
- *Spin tooltip*. When a spin selection is performed, the previous tooltip is expanded with detailed information on the cluster currently selected. The list of clusters reports only the selected cluster in bold, and the remaining ones in normal font (see figure 7.5).
- *Position tooltip*. When the cursor is kept still over a single cluster, the tooltip shows detailed information on that cluster (see figure 7.6a). When it is placed over an intersection, the tooltip presents the information relative to each of the clusters in a more condensed way (see figure 7.6b).

7.1.3 Path-Preserving Meta-Nodes

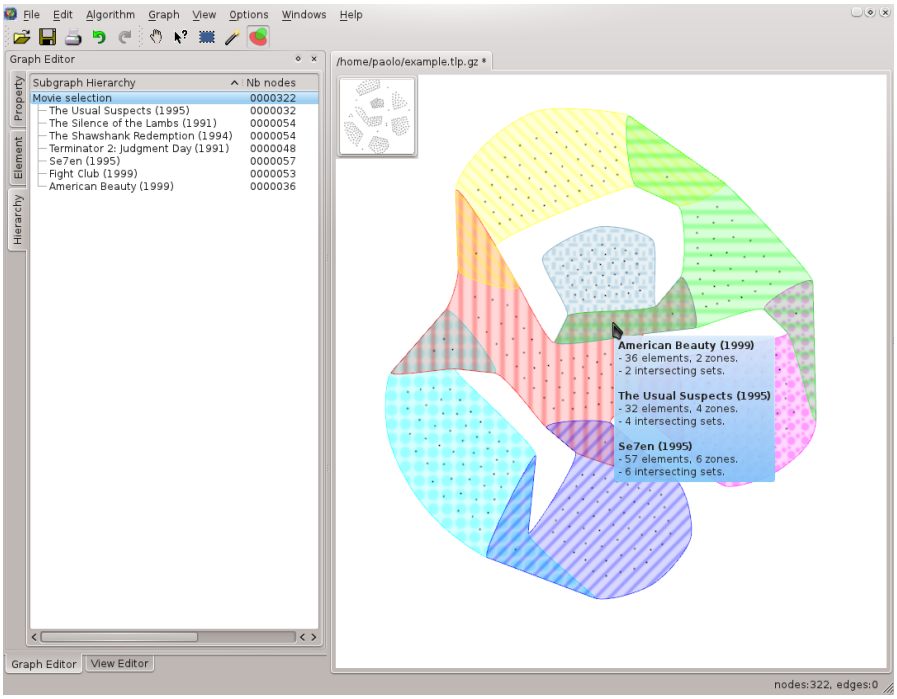
In order to reduce and condense the critical information of a complex graph, Archambault, Munzner and Auber [2] introduced the decomposition in path-preserving meta-nodes. A *meta-node* is a structure that encompasses a portion of the graph into a graph node. Typically, the meta-nodes can be dynamically opened and closed to show or hide their content.

When closed, a meta-node appears as a graph node, generally having different colour or shape with respect to the standard nodes. The content of the meta-node is collapsed in the meta-node position, so that the edges exiting the meta-node appear incident to it. When the meta-node is opened, the content is re-positioned in the graph and the meta-node glyph is partially hidden.

Path-preserving meta-nodes are devised to avoid misleading the user on the connectivity of the meta-node content. For instance, a meta-node could contain two or more components that are not connected. If two external nodes are linked to disconnected components, they will appear connected through it when the meta-node is closed. Path-preserving meta-nodes avoid this behaviour by forbidding the creation of meta-nodes that might induce fictional connections when they are collapsed (see figure 7.7).



(a)



(b)

Figure 7.6: Examples of position tooltips. **(a)** Single cluster position tooltip. **(b)** Multiple cluster position tooltip.

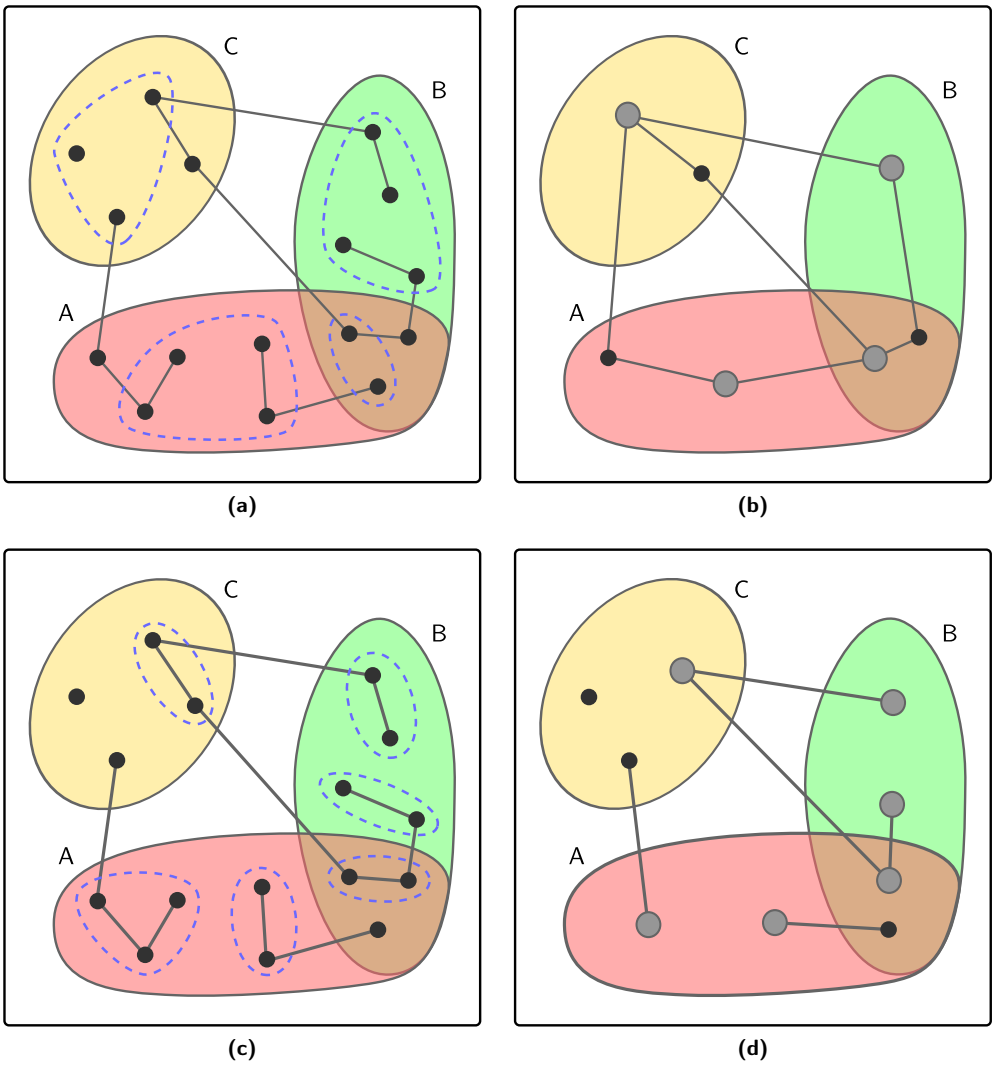


Figure 7.7: Generic meta-nodes and path-preserving meta-nodes. **(a)** Let us suppose that we choose to collapse the nodes contained in the dashed lines into meta-nodes. **(b)** The resulting diagram, with the meta-nodes closed, might induce the user to think that the nodes contained in zone a and in zone b are connected, which is not correct. **(c)** The meta-nodes we generate in Euler representations. **(d)** These meta-nodes do not mislead the user on the connectivity between the graph elements.

Diagram	$\|S\|$	$\|V^o\|$	$\|E^o\|$	ImPrEd Z (s)	ImPrEd G (s)	Total (s)	ImPrEd R %
IMDb 7	7	322	0	0.40	13.87	15.22	93.8
IMDb 60	60	2236	0	8.20	201.49	253.85	82.6
Platelet	6	—	—	—	—	—	—
Gene interaction	10	176	296	0.26	8.73	9.54	94.2
Carsonella	35	224	334	1.06	22.67	25.89	91.6

Table 7.1: Statistics and computation time for the Euler representations showed in this chapter. The columns contain, in the order from left to right, the number of clusters, the number of original graph nodes, the number of original graph edges, the execution time of the first execution of ImPrEd, the execution time of the second execution of ImPrEd, the total computation time, and the ratio between the time required by the two executions of ImPrEd and the total computation time. Unfortunately, most statistics for Platelet are not available.

Path-Preserving Meta-Nodes in EulerView

We implemented path-preserving meta-nodes in EulerView. For each diagram zone, we study the connected component of their induced subgraphs. The components composed of more than one node are associated with new meta-nodes.

In the diagram, meta-nodes can be opened and closed by double-clicking on them. Also, all the meta-nodes contained in the performed selection can be opened and closed by pressing the keys "O" and "C".

7.2 Examples of Euler Representations

In this section we show some examples of Euler representations generated by EulerView. All the diagrams come from real world-data and can be used to extract significant information.

In the following, we present individually each diagram and the data used to generate them. Table 7.1 reports statistics such as the number of clusters, original nodes and edges, and the computation time required to generate these diagrams. We also report the time required by each graph optimisation, as well as the ratio between the sum of the optimisation times over the total generation time, to clarify how the computation time is mostly given by the executions of ImPrEd.

7.2.1 IMDb

The diagrams called *IMDb* are based on data extracted from the IMDb website [61]. IMDb provides a chart of the top 250 films voted by the website's users and information on a great number of films. In 2009, we extracted the chart and the full cast for the films in the first 70 positions.

We constructed a few diagrams in which the films are the clusters, the actors are the original graph nodes and the overlaps indicate that the movies shared part of their cast.

The diagram *IMDb 7* features seven films and 322 actors. We manually chose the films to provide a relatively small and compact diagram to explain the generation procedure and the interaction features. The diagram is shown in figure 4.14 on page 76 and in the figures at the beginning of this chapter.

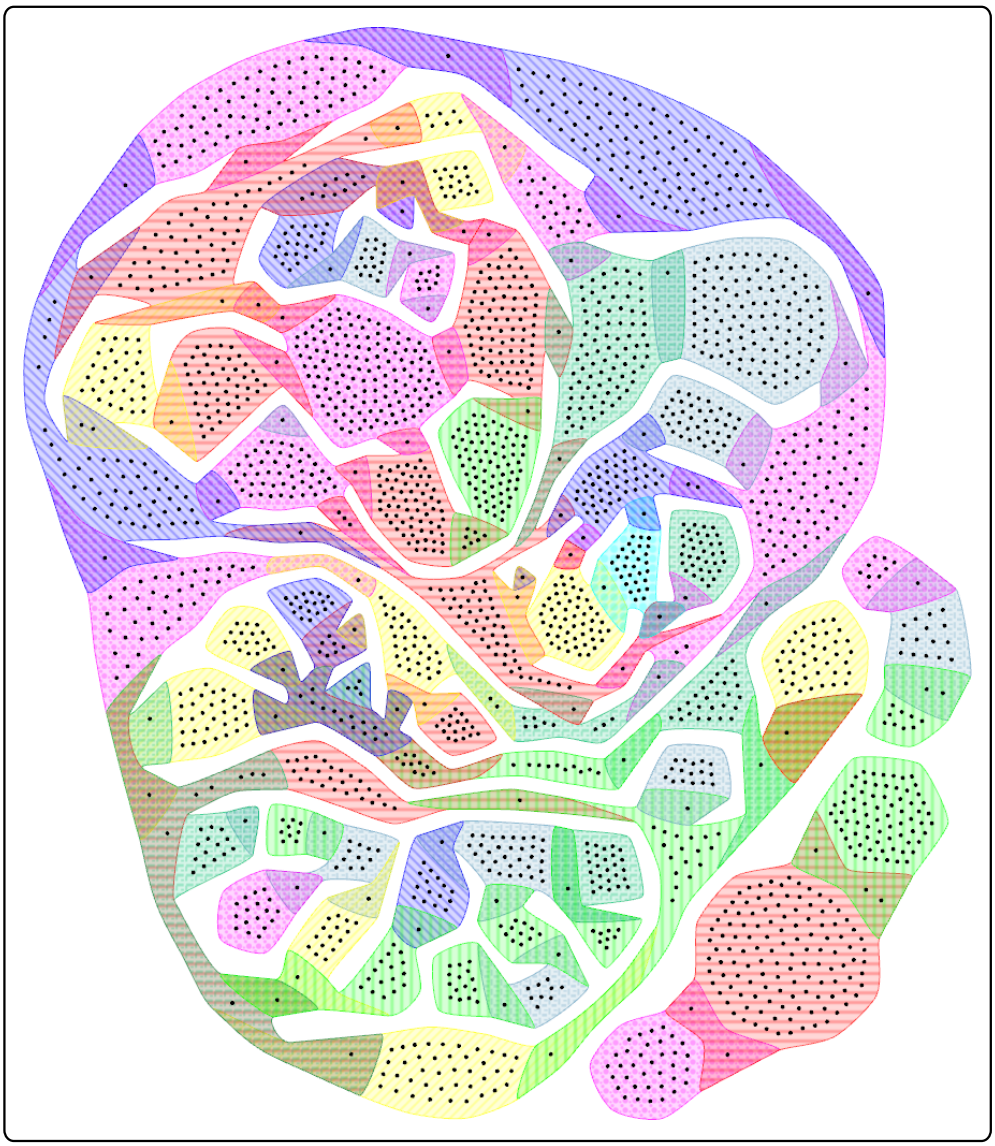


Figure 7.8: Euler representation of the IMDb 60 data set. The diagram has been rotated 90° anticlockwise to better fit the page.

The diagram *IMDb 60* features 60 films and a total of 2263 actors, and it is shown in figure 7.8. The films depicted are all those of the top 70 list who intersected with each other. In other words, the films that were excluded would all have appeared isolated in the diagram. We generated the diagram to show the ability of the method to scale to relatively large input instances.

The Diagrams. The diagram IMDb 7 is pleasant and readable, but it shows how the cluster regions could benefit from more regular shapes (see figures 7.3 and 7.5). Such irregular regions are necessary when drawing large and complex diagrams, but are not required for small and simple ones.

The diagram IMDb 60 also exhibits high aesthetics, especially when considering the large number of clusters and original graph nodes it features. In particular, the diagram appears to be well spaced and organised. The zones seem to be well sized, since the area of the zones that are not stretched for connectivity issues look quite proportional to the number of contained elements.

Clearly, such a pleasant result for a diagram of more than 60 clusters is also due to the data set, that does not present very complex overlaps.

7.2.2 Platelet

The diagram called *platelet* is constructed using the results from proteomics experiments obtained from several biological studies. Proteomics experiments can be used to detect proteins which are present in a cell or a tissue. In this case they are used to obtain a list of proteins which are present in blood platelets, which are a cell type essential for hemostasis.

Platelet proteomics experiments detect only a subset of the total proteins. Also, the proteins which are detected depend on the design of the experiment. Previous experiments have been targeted at a specific class of proteins, such as membrane, low abundance and peripheral membrane. Many of these experiments have expanded the known platelet proteome though there is a large amount of redundancy between them.

The Diagram. The diagram in figure 7.9 shows the overlaps between five platelet proteomics data sets [50, 52, 67, 68, 82] and one platelet specific RNA profile [112]. From this diagram, it is clear that each proteomics data set is contributing a large amount of unique proteins. The RNA profile (purple) identifies proteins which are specific to Megakaryocytes versus other blood cells. As platelets are derived from Megakaryocytes, this is used to identify proteins which may be uniquely expressed in blood platelets versus other cell types.

7.2.3 Gene Interaction

The diagram called *gene interaction* displays clusters over a gene interaction network. Itoh et al. [62] presented a method for visualising overlapping graph clusters that does not rely on Euler diagrams, but on the element positioning and colouring in a node-link diagram. One of the examples they showed involved a protein-protein interaction network featuring 6152 genes and 7564 interactions between them. Also, the genes were characterised by their response to 173 types of stress conditions.

Over this huge network, ten overlapping clusters have been extracted, so that each group contains genes with common expression or repression conditions. The clusters contained a total of 176 genes, having 296 interactions between them.

The Diagram. The diagram in figure 7.10 shows the Euler representation built on the ten clusters, and on the subgraph induced by the nodes they contain.

In this diagram, it is interesting to note how the edges of the original graph contribute to the shape of the final diagram. Even though their reduced weight does not allow them to drastically model the overall structure of the drawing, they tend to move the nodes having edges that exit the zone along the zone borders. Also, the zones seem to be distributed in a way that reduces the length of extra-zone edges.

It is also interesting to note how the meta-nodes can help extracting insight on the connectivity of the nodes. For example, by closing the meta-nodes, the very low

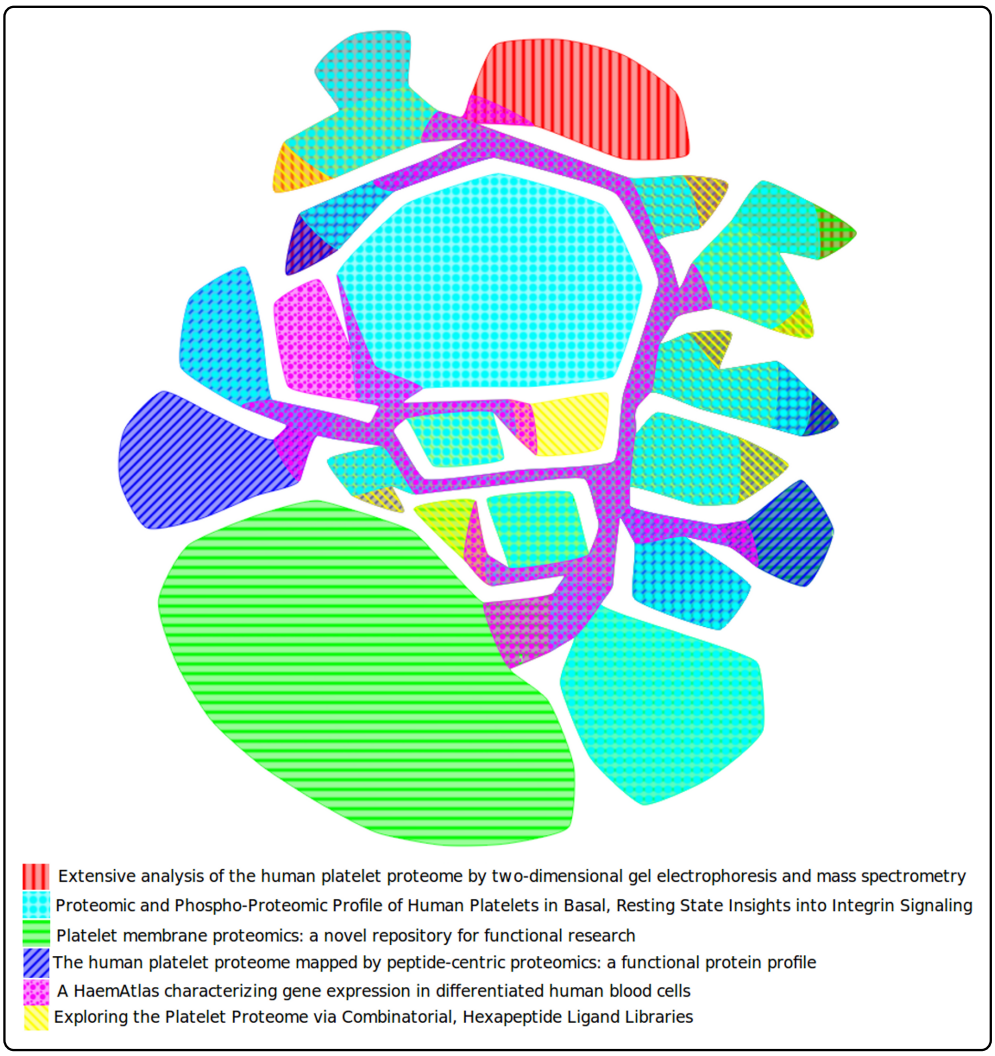


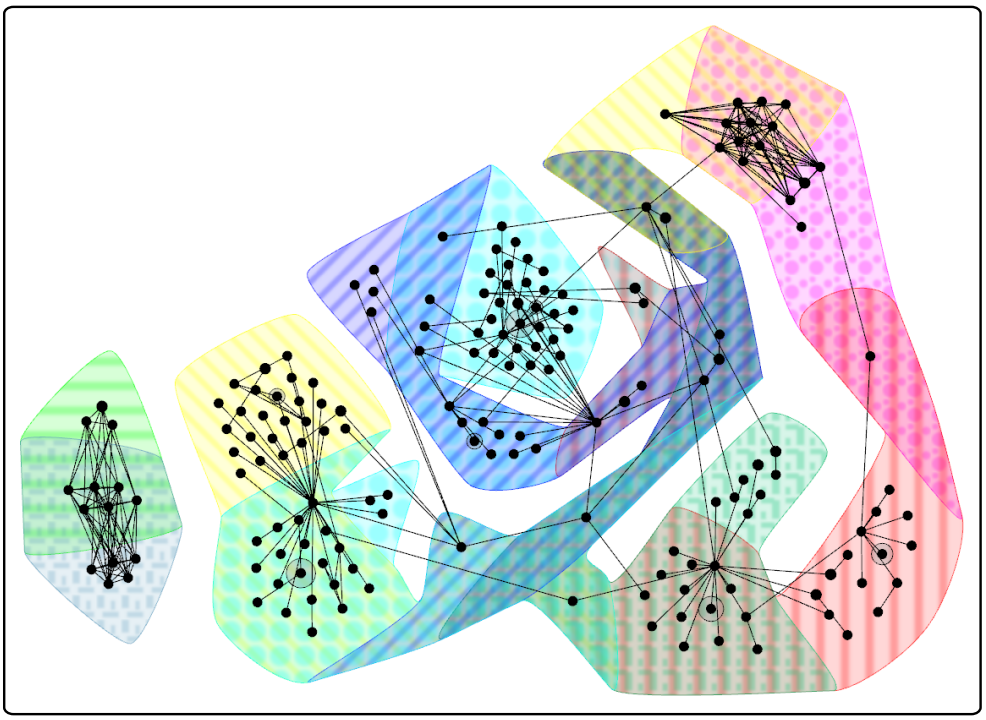
Figure 7.9: Euler representation of the Platelet data set (Courtesy of Kevin O'Brian).

inner connectivity of the yellow zone in the centre-left becomes much more evident. Opening the meta-nodes, we can note how this is due to the presence of a hub node in the zone just below. Even though this observation was possible without the presence of meta-nodes, the reduced number of elements and edge cluttering they induce can be helpful, especially when considering larger diagrams.

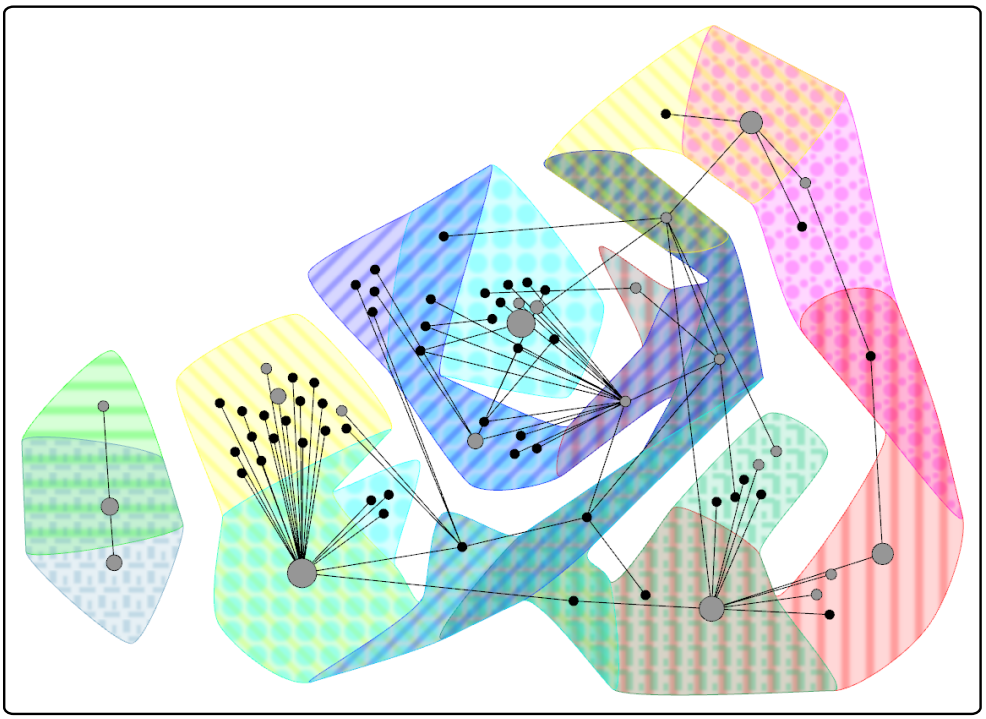
7.2.4 Carsonella

The diagram we call *Carsonella* is constructed on the metabolic network of the bacteria *Candidatus Carsonella ruddii* [70]. Metabolic networks represent the chemical reactions of the metabolism of the organism. For this reason, the nodes can either represent *metabolites*, that are the components of the chemical reactions, or *enzymes*, that are the proteins that trigger such reactions.

Metabolic networks are usually clustered in *pathways*, that are groups of reactions



(a)



(b)

Figure 7.10: Euler representation of the gene interaction data set. **(a)** The diagram with all meta-nodes opened. **(b)** The diagram with all meta-nodes closed.

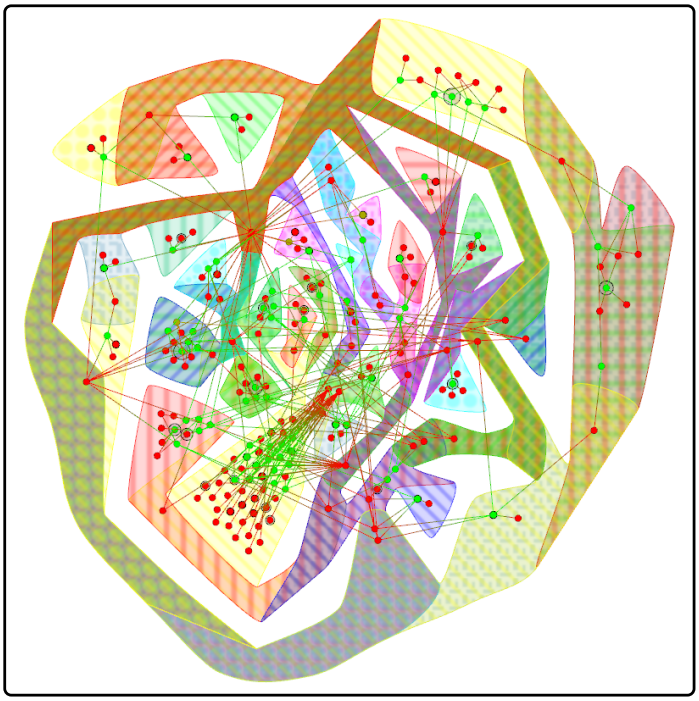
finalised to a specific function. Since certain reactions and products are used in multiple pathways, the pathways overlap with each other. To reflect the different nature of the network nodes, these have different colours. The red nodes are the metabolites, and the green nodes are the enzymes.

The metabolic network of the bacteria contains 35 pathways, that groups a total 224 nodes and 335 edges of the original graph.

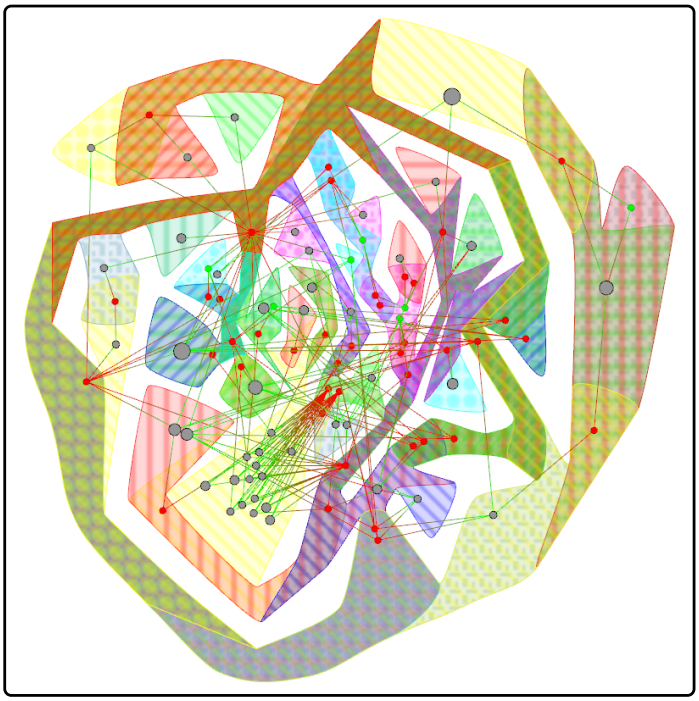
The Diagram. The diagram generated by EulerView is shown in figure 7.11. The first figure shows the graph with all the meta-nodes opened, meaning that all the elements of the original graph are currently displayed. In the second figure, all the meta-nodes have been closed. Closing the meta-nodes allow to sensibly reduce the cluttering of the diagram, while keeping all the connections between zones visible.

In the diagram, we can easily identify a zone shared by a very high number of clusters, recognisable by a texture that presents many different patterns. If we investigate the zone and its content, we notice that the node is water and that this component is used in 20 pathways (see figure 7.12a).

Also, the graph shows three red nodes having a very high degree. In figure 7.12b, the node near the tooltip is ATP, while the two red nodes above it are AMP and diphosphate, which are all very important molecules in the metabolism. As they do not belong to the same zone, we can consult the tooltips to discover that the ATP is in the same three pathways of the others, and in three other pathways.

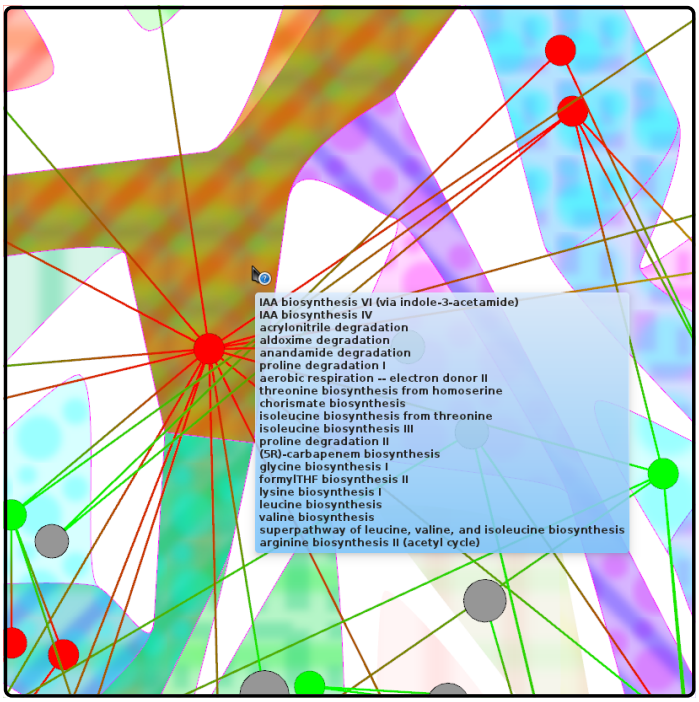


(a)

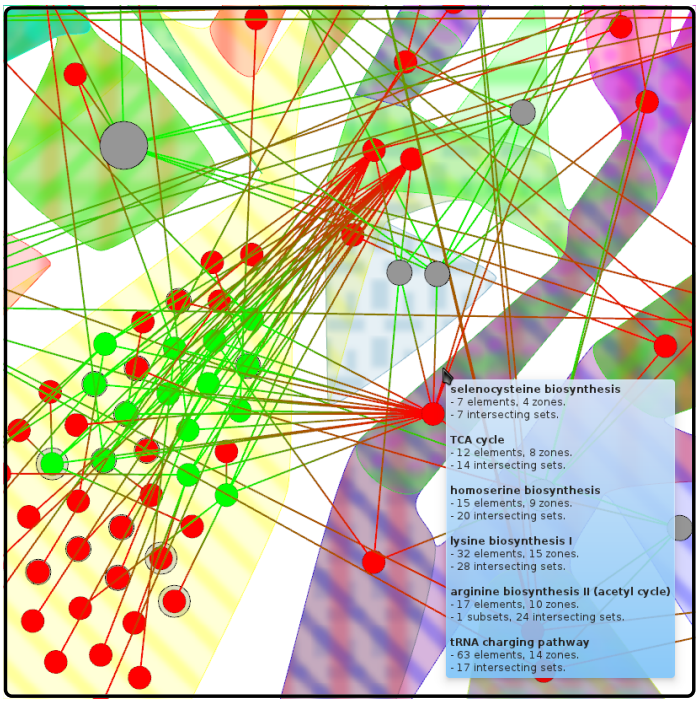


(b)

Figure 7.11: Euler representation of the Carsonella data set. **(a)** The diagram with all the meta-nodes opened. **(b)** The diagram with all the meta-nodes closed.



(a)



(b)

Figure 7.12: Details of the Euler representation of Carsonella. **(a)** A zone of high overlap that contains a single reaction component: water. **(b)** Three nodes of high degree (the red node near the tooltip, and the two red nodes above), corresponding to important components of the metabolism.

Chapter 8

Conclusions

In this final chapter, we draw the conclusions on the work done, the results obtained and on the directions for future work.

First, we summarise the work presented in this thesis. In particular, we remark the problems we aimed to solve and the solutions we developed.

Second, we state our contributions to the field and present the papers we published on the topic.

Third, we discuss the advantages and disadvantages of the visualisation we proposed and of its generation method.

Finally, we close the thesis by proposing several ways of extending and improving the work presented.

8.1 Aims and Realisation

In this thesis, we aimed to provide a method for the visual analysis of overlapping sets and fuzzy graph clusterings. We chose to base the resulting visualisation on Euler diagrams, due to their high intuitivity and their solid perception properties.

The solution we proposed consists of a form of diagrams, a procedure that generates them, and interaction features that allow to increase the potentials of the visualisation in the data analysis.

The Visualisation. The diagrams we construct, called Euler representations, show high similarity with Euler diagrams as commonly intended, since both rely on the same metaphors to express the set intersection and element containment.

The main difference between them is that Euler representations have the possibility of depicting a set with disconnected portions of the plane. Such a difference is motivated by the necessity of producing a diagram for every input, since there exist cases for which standard Euler diagrams do not exist.

The Procedure. The procedure that generates Euler representations receives a fuzzy clustered graph as input, and produces the final drawing through a sequence of five main steps:

- *Zone graph construction.* We identify the regions that we must depict in the final diagram (the nodes of the zone graph) and we enforce adjacency between them (through edges of the zone graph).

- *Zone graph drawing.* We identify the general positions of the regions in the final diagram by computing and optimising a planar drawing of the zone graph.
- *Grid graph construction.* We compute the boundaries of the regions in the final diagram by enclosing the zone graph nodes and edges within another graph, called grid graph.
- *Grid graph drawing.* We insert the original elements in the faces of the grid graph, and we optimise the current drawing to obtain the positions of the elements and set boundaries.
- *Cluster curve depiction.* Finally, we depict the set boundaries and we apply some graphical techniques to help the user discern the sets in correspondence to the intersections.

Graph Layout Improvement. The procedure requires to optimise the aesthetics of two graphs, while preserving the reciprocal positions of their elements. `PrEd` is a force-directed graph drawing algorithm that allows to improve the layout of a graph without altering these properties, or more precisely, while impeding nodes from crossing edges when moving.

Due to the importance of the graph optimisation phases on the quality of the final visualisation, and due to their weight in the total computation time of the diagram generation procedure, we decided to develop an algorithm that improves `PrEd`. `ImPrEd` preserves much of the original algorithm, but proved to be significantly faster, more reliable on the input parameters, and able to produce better drawings.

`ImPrEd` also has a couple of additional features that are particularly useful for generating Euler diagrams, but that can be used even on normal graphs. The most noticeable is flexible edges, that makes edges able to adapt to the stress exerted on them by inserting or removing bends.

Interaction. Once the visualisation is generated, a few interaction techniques are provided to facilitate the data analysis. In particular, these techniques include the possibility of identifying the sets that are present in a given intersection, visualise their statistics and show or hide sets according to the portion of the diagram of current interest.

Also, in order to reduce an eventual clutter caused by the input graph edges, the algorithm computes meta-nodes that allow to encompass the connected components inside a region into a single entity. The meta-nodes preserve the paths between elements in different regions, meaning that even when they are closed they do not lead the user to wrong conclusions about the connectivity of the input graph.

8.2 Contributions

To accomplish the aim of the thesis, we had to address several issues that were only partially covered in the literature. First, we needed to identify a class of diagrams that had no undrawable instances. Second, we needed to provide a generation method sufficiently fast, reliable and able to handle large diagrams. Third, we had to implement the algorithm to verify its applicability and the quality of the diagrams produced.

The major contributions provided when solving these problems are explained in the remaining part of the section. For each contribution, we will mention both the published papers and the sections of this thesis in which the topic is treated.

Drawability Issues. In order to always be able to produce a diagram, we studied which Euler diagram properties we had to violate and which solutions we could use to mitigate the loss of readability that this would cause [90]. Although drawability issues were already considered in previous work, there seems to be little concern about undrawable instances, even if this precluded their application of Euler diagram generation methods to visual data analysis.

The contents of this study are mainly present in section 3.3.

> [90] Paolo Simonetto and David Auber. 'Visualise Undrawable Euler Diagrams'. In: *International Conference on Information Visualisation (IV08)*. Ed. by Ebad Banissi, Liz J. Stuart, Mikael Jern et al. IEEE Computer Society, 2008, pp. 594–599.

Automatic Generation of Euler Representations. As a result of the previous considerations, we designed a generation procedure that produces less restrictive Euler diagrams, called Euler representations.

The generation procedure is divided into five phases. We provided an heuristic [89] to solve the first step, that is the generation of the zone graph, and the details on how to proceed on the following four steps [91]. The overall procedure is quite similar to that of several other methods, but it substantially differs on the identification and depiction of the set boundaries. Also, the construction of the backbone graph, called zone graph in our method, is generally very scarcely detailed in the literature.

In this thesis, the contents of the two articles are sketched in section 4.2 and detailed in chapter 5.

> [89] Paolo Simonetto and David Auber. 'An Heuristic for the Construction of Intersection Graphs'. In: *International Conference on Information Visualisation (IV09)*. Ed. by Ebad Banissi, Liz J. Stuart, Theodor G. Wyeld et al. IEEE Computer Society, 2009, pp. 673–678.
>
> [91] Paolo Simonetto, David Auber and Daniel Archambault. 'Fully Automatic Visualisation of Overlapping Sets'. In: *Computer Graphics Forum (EuroVis09)* 28.3 (June 2009), pp. 967–974.

ImPrEd. Since our procedure for the generation of Euler representations relies heavily on `PrEd` to produce the final drawing, we spent considerable effort in improving it. `ImPrEd` [92] was developed to improve its execution time, drawing quality and the reliability of the input parameters.

To obtain these results, we added several features to the original algorithm:

- *Force and movement cooling.* This feature is inspired by similar approaches in other force-directed algorithms. However, the force system and the cooling policy had to be adapted to the very different behaviour of `PrEd` and `ImPrEd`, given the node movement restriction.
- *Surrounding edges.* The possibility of computing the surrounding edges had already been suggested by `PrEd`'s author as future work. However, an algorithm for this purpose had not been provided. The method we proposed for computing the surrounding edges is sufficiently fast and generates a significant speed up in terms of the computation time.

- *Flexible and crossable edges.* Crossable edges represent a natural addition to the `PrEd` algorithm, and it did not require substantial efforts to integrate them into the algorithm. On the other hand, flexible edges proved to be a very interesting and quite innovative addition, that could prove to be useful even outside the generation of Euler diagrams.
- *QuadTrees, weights and new maximal movement rules.* These features can be considered minor contributions, either because they consist of the mere implementation of well known techniques, such as QuadTrees, or because they did not require significant effort to be designed or implemented.

The content of this article is discussed in chapter 6.

> [92] Paolo Simonetto et al. 'ImPrEd: An Improved Force-Directed Algorithm that Prevents Nodes from Crossing Edges'. In: *Computer Graphics Forum (EuroVis11)* 30.3 (June 2011), pp. 1071–1080.

Software Implementation and Interaction. We developed a Tulip plug-in, called EulerView, that can be used to generate and interact with Euler representations. The plug-in is freely available and will hopefully be integrated with Tulip in a future version of the software.

Although the interaction features developed so far are quite basic, they already allow to facilitate the analysis of the diagrams and the extraction of salient information.

These early results are contained in chapter 7.

Euler Diagram Theory and Visualisation. Finally, there are a last couple of minor contribution in this thesis. The first consists of the investigation on the perceptual reasons that make Euler diagrams such a powerful tool in the visual analysis of data. The conclusions we draw on these aspects are reported in section 2.3.1.

The second is related to the Euler theory presented in section 3.3. The concepts presented in that section had been defined and explained by every authors that proposed an algorithm for the generation of Euler diagrams. However, the use of different assumptions and terms contributed to make the actual differences between the methods, the kind of diagram generated and the undrawable class of each algorithm, unclear. In this thesis, we tried to clarify these concepts by reporting, with uniform definitions, the characteristics of each approach and the class of diagrams they generate.

8.3 Results

In section 7.2 we showed some examples of the diagrams generated with the procedure and software we developed. The diagrams are generally comprehensible and aesthetically pleasant. However, as expected, the readability of the final diagram is very dependent on the way the sets intersect, which has an even greater effect than the number of sets or elements. Already in section 2.3.3, we mentioned this problem: we can generate readable Euler diagrams for hundreds of sets and thousands of elements, but we cannot generate very clear Euler diagrams that show all the 36 possible intersections of six sets.

Intersections Complexity. Diagrams characterised by relatively simple set intersections, such as those shown in figures 4.14 on page 76 (IMDb 7) and 7.8 on page 154 (IMDb 60), are very informative and fairly easy to read. Also, in this case the graph size has relatively little impact on the graph comprehensibility, which is quite well preserved at the increase of the input size.

Diagrams characterised by complex set intersections, such as that in figure 7.11 on page 159 (Carsonella), rely more heavily on interaction to compensate for the lower readability. In fact, when dozens of sets overlap in the same region, even the textures are very difficult to distinguish.

Computation Times. The computation times required for the generation of Euler representations are reasonable, even when considering relatively large diagrams, if the diagram needs to be generated only once.

Applications that require both high interaction with the user and the continuous generation of diagrams can benefit from our method only if the number of sets and elements is limited. In such a scenario, computation times inferior to ten seconds required by a diagram with up to a dozen of sets and two hundred elements, would be acceptable. Computation times of about four minutes, required by diagrams of the size of IMDb 60, would not be acceptable in this scenario.

Concurrency and Shape of the Clusters. The addition of high levels of concurrency significantly contributed to the reduction of the computation times, but at the expense of the drawing quality. The concurrency of the curves is an obstacle to the identification of the cluster regions, which also tends to be characterised by an irregular shape.

The irregularity of the shapes is evident in particular on small and simple diagrams, such as that in figure 7.3 on page 148, since it would be possible to find more readable configurations. The problem is progressively less evident with the increase of the diagram size and complexity, since the non-regular shapes generated by our method are necessary to draw such diagrams.

The high cluster curve concurrency presents a similar pattern. On small and simple diagrams, it is mostly unneeded and reduces the comprehension of the diagram. However, with the increase of the complexity of the diagrams, the concurrency might reduce an excessive amount of the information displayed. For example, twenty or more non-concurrent curves are not necessarily more comprehensible than superimposed ones, as shown by recent success of edge bundles [60].

8.4 Future Work

We conclude the thesis by providing several directions for future improvements to the method. We classify them into three categories: the improvement of the diagram readability, the reduction of the computation time, and extensions of the method.

Drawing Readability. The best way of improving the readability of the generated diagrams would be to produce more regular cluster curves. There are several ways of obtaining this result:

- We could enforce a higher connectivity of the zone graph nodes that belong to the same cluster. This would generate cluster regions with a more compact

shape, rather than the current, stretched ones. Since this method would create a large number of holes in the cluster regions, it would be necessary to remove them without altering the current diagram.

- We could insert additional forces in `ImPrEd` to promote a more regular positioning of the grid graph nodes that belong to the same cluster curve, inducing them to assume a round or elliptical shape. This method would require a very good balance of the forces, that could be extremely difficult to reach for the many conditions that need to be simultaneously enforced.

- We could smooth the boundaries immediately before the depiction of the cluster curves, taking care not to alter the current diagram. This method would be relatively fast and simple, but its applicability would be highly dependent on the graph layout.

The method would also benefit from reducing the high level of concurrency currently imposed. This could also be obtained in several ways:

- We could replace the current method based on the grid graph with a contour routing approach, such as that proposed by Rodgers et al. [85], and optimise the results with `ImPrEd`. However, this method might cause a significant increase in the running time of `ImPrEd`, since each curve is now considered individually. Also, we would need to re-work the expansion mechanism for flexible edges, as they would need to be applicable to crossing edges.

- We could split the concurrent curves after the application of `ImPrEd`. With a high value of γ we can ensure enough space around the zone boundaries, and draw the cluster curves sharing the same edge as parallel lines (similarly to what is done in transit maps). This method would require computing the order in which the lines must appear, and might not be as efficient as the previous in terms of readability.

Running Time. Since the use of `ImPrEd` is responsible for about 80% to 95% of the total running time, it is still important to focus on the efficiency of the graph layout optimisation phase. We could approach the problem in the following ways:

- We could attempt to reduce the complexity of `ImPrEd`, which would therefore be faster and able to scale to larger input. This approach would only present benefits, but a complexity reduction might be difficult to achieve.

- We could reduce the input instance, using a multilevel approach typical of modern force-directed algorithms (see the related paragraph in section 3.2.3). For instance, we could enclose all the original graph nodes contained in the same zone into a single meta-node, and split the meta-node into smaller entities with the progression of the computation. This would greatly speed up the process, but it would require to specialise `ImPrEd` for the optimisation of Euler diagrams.

- We could rely on different kinds of algorithms, such as those of Dwyer et al. [26–29], that seem to be more efficient and easier to tune to enforce a particular constraint.

Extensions. The method discussed in this thesis would also benefit from extensions and new features, in particular for what regards the interaction with the user:

- We could give the possibility of redrawing a diagram using only a sub-set of selected clusters. In order to preserve the mental map, we could start from the diagram generated for the whole graph, re-optimise its layout and merge eventual disconnected cluster regions whenever possible.
- We could also further extend the previous point by drawing the diagrams on-demand. Instead of drawing a large diagram containing all clusters and elements, we could give the possibility to the user to select the clusters of current interest and produce a drawing for those clusters only. In order to preserve the mental map and provide a rough overview of the cluster relationships, we could design a visualisation based on the zone graph.

Appendix A

Biographies

This thesis is inspired by the work of major mathematicians, physicians and philosophers. In this section, we pay the right tribute to their talent by presenting a summary of their life and scientific contributions.

We first present the biography of Leonhard Euler [33, 77]. Euler have been cited as the inventor of the earlier form of diagrams, although some evidences proof their earlier usage [57]. Then, we present a biography of John Venn [77], who formalised and contributed to the diffusion of these diagrams.

Leonhard Euler. Leonhard Paul Euler was born on the 15th of April 1707 in Basel, Switzerland. He developed his extraordinary talent in mathematics thanks to the teachings of Johann Bernoulli, who was then regarded as Europe's foremost mathematician.

Figure A.1: Leonhard Euler

In May 1727 Euler moved to Russia to study and work at the Academy of Sciences at St. Petersburg, recently established by Peter I (also known as Peter the Great) to promote Russian education and to close the scientific gap between Russia and Europe. Here, Euler swiftly rose through the ranks in the academy until becoming head of the mathematics department in 1733. However, the increasing hostility that foreign researches had to face since the death of Catherine I — who succeeded the husband Peter I at the lead of the Russia empire and who shared his will in developing Russian science and art — eventually made Euler decide to leave the country.

In June 1741 Euler took a place at Berlin Academy offered by Frederick the Great of Prussia. In the 25 years he spent in Berlin, he published 380 articles and some of his most important works: *Introductio in analysin infinitorum* on mathematical functions, *Institutiones calculi differentialis* on differential calculus and *Vollständige Anleitung*

zur Algebra (Elements of Algebra) the first books to set out algebra in its current form. He also accepted to tutor Frederick's niece, the princess of Anhalt-Dessau, through a long series or letters on different topics of physics and mathematics [33]. Some of these letters contain the diagrams that are generally considered the ancestor of modern Euler diagrams.

In 1766, the improvement of the Russian situation under Catherine II (also known as Catherine the Great) and the deterioration of the relationship with Frederick of Prussia made him decide to return to the Academy of St. Petersburg, where it stayed and studied for the rest of his life. Euler died on the 18th of September 1783.

John Venn. John Venn was born on the 4th of August 1834 in Hull, Yorkshire, England. His father Henry played a prominent role in the evangelical Christian movement, and educated his son into the Christian ministry. In October 1853 he entered Gonville and Caius College, Cambridge, where he pursues his mathematical studies. In the second year, he was awarded a mathematics scholarship and in 1857 he graduated with excellent results.

Figure A.2: John Venn

In 1859 Venn was ordered priest and in 1862 he returned at Gonville and Caius College as lecturer in moral sciences. There, he primarily studied and taught logic and probability theory.

The results of his studies are collected in three main texts. In *The Logic of Chance* (1866) he introduced the frequency interpretation of probability, that criticised the classical interpretation of probability proposed by Pierre-Simon Laplace. In *Symbolic Logic* (1881) [106] he presents diagrams that illustrate Boole's mathematical logic. These diagrams were subsequently named after him, and represent the contribution for which Venn is mostly known. Finally, in *The Principles of Empirical Logic* (1889) he provides an overview of the field of logic and make further use of these diagrams.

For his merits, in 1883 Venn was elected a fellow of the Royal Society and was awarded a Sc.D. by Cambridge. John Venn died on the 4th of April 1923.

Bibliography

[1] Alex T. Adai, Shailesh V. Date, Shannon Wieland and Edward M. Marcotte. 'LGL: Creating a Map of Protein Function with an Algorithm for Visualizing Very Large Biological Networks'. In: *Journal of Molecular Biology* 340.1 (Jan. 2004), pp. 179–190 (cit. on p. 47).

[2] Daniel Archambault, Tamara Munzner and David Auber. 'GrouseFlocks: Steerable Exploration of Graph Hierarchy Space'. In: *IEEE Transactions on Visualization and Computer Graphics* 14.4 (July 2008), pp. 900–913 (cit. on p. 150).

[3] David Auber. 'Tulip: a Huge Graph Visualisation Framework'. In: *Graph Drawing Softwares*. Ed. by Petra Mutzel and Michael Jünger. Mathematics and Visualization. Springer-Verlag, 2003, pp. 105–126 (cit. on pp. 15, 145).

[4] David Auber. 'Tulip'. In: *International Symposium on Graph Drawing (GD01)*. Ed. by Petra Mutzel, Michael Jünger and Sebastian Leipert. Vol. 2265. Lecture Notes in Computer Science. Springer, 2002, pp. 488–491 (cit. on pp. 15, 145).

[5] Florence Benoy and Peter Rodgers. 'Evaluating the Comprehension of Euler Diagrams'. In: *International Conference on Information Visualisation (IV07)*. Ed. by Ebad Banissi, Remo Aslak Burkhard, Georges Grinstein et al. IEEE Computer Society, 2007, pp. 771–780 (cit. on p. 53).

[6] François Bertault. 'A Force-Directed Algorithm that Preserves Edge Crossing Properties'. In: *Information Processing Letters* 74.1–2 (Apr. 2000), pp. 7–13 (cit. on pp. 47, 63, 84, 101, 105, 137, 139).

[7] David Brewster and John Griscom. *Letters of Euler on Different Subjects in Natural Philosophy Addressed to a German Princess*. New York: J. & J. Harper, 1837 (cit. on p. 17).

[8] Stefan Bruckner, Sören Grimm, Armin Kanitsar and Meister E. Gröller. 'Illustrative Context-Preserving Exploration of Volume Data'. In: *IEEE Transactions on Visualization and Computer Graphics* 12.6 (Nov. 2006), pp. 1559–1569 (cit. on p. 14).

[9] Heorhiy Byelas and Alexandru Telea. 'Texture-based visualization of metrics on software architectures'. In: *ACM Symposium on Software Visualization (SoftVis08)*. 2008, pp. 205–206 (cit. on p. 99).

[10] Stuart K. Card, Jock Mackinlay and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999 (cit. on pp. 4, 13).

[11] Chaomei Chen. 'Information Visualization'. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.4 (2010), pp. 387–403 (cit. on p. 13).

[12] Stirling Christopher Chow. 'Generating and drawing area-proportional Euler and Venn diagrams'. PhD thesis. Greater Victoria, British Columbia, Canada: University of Victoria, 2007 (cit. on pp. 53, 55, 56, 63).

[13] Marek Chrobak and Thomas H. Payne. 'A Linear-time Algorithm for Drawing a Planar Graph on a Grid'. In: *Information Processing Letters* 54.4 (1995), pp. 241–246 (cit. on p. 45).

[14] Jonathan D. Cohen. 'Drawing Graphs to Convey Proximity: An Incremental Arrangement Method'. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 4.3 (Sept. 1997), pp. 197–229 (cit. on p. 47).

[15] Christopher Collins, Gerald Penn and Sheelagh Carpendale. 'Bubble sets: Revealing Set Relations with Isocontours Over Existing Visualizations'. In: *IEEE Transactions on Visualization and Computer Graphics (InfoVis09)* 15.6 (Nov. 2009), pp. 1009–1016 (cit. on pp. 71, 72).

[16] Nicolas de Condorcet and Sylvestre François Lacroix. *Lettres de Euler à une princesse d'Allemagne sur divers sujets de physique et de philosophie*. Paris: Royez, 1789 (cit. on p. 17).

[17] Ron Davidson and David Harel. 'Drawing Graphs Nicely Using Simulated Annealing'. In: *ACM Transactions on Graphics (TOG)* 15.4 (1996), pp. 301–331 (cit. on pp. 46, 115).

[18] Hubert De Fraysseix, János Pach and Richard Pollack. 'How to draw a planar graph on a grid'. In: *Combinatorica* 10.1 (1990), pp. 41–51 (cit. on pp. 44, 45, 84).

[19] Edmund Dengler and William Cowan. 'Human Perception of Laid-Out Graphs'. In: *International Symposium on Graph Drawing (GD98)*. Ed. by Sue Whitesides. Vol. 1547. Lecture Notes in Computer Science. Springer, 1998, pp. 441–443 (cit. on p. 44).

[20] Agnès Desolneux, Lionel Moisan and Jean-Michel Morel. 'Gestalt Theory and Computer Vision'. In: *Seeing, Thinking and Knowing*. Ed. by Arturo Carsetti. Vol. 38. Theory and Decision Library. Springer, 2004, pp. 71–101 (cit. on p. 6).

[21] Giuseppe Di Battista and Roberto Tamassia. 'Incremental Planarity Testing'. In: *Annual Symposium on Foundations of Computer Science*. 1989, pp. 436–441 (cit. on p. 86).

[22] Giuseppe Di Battista and Roberto Tamassia. 'On-line Graph Algorithms with SPQR-Trees'. In: *International Colloquium on Automata, Languages and Programming*. Ed. by Michael S. Paterson. Vol. 443. Lecture Notes in Computer Science. Springer, 1990, pp. 598–611 (cit. on p. 86).

[23] Giuseppe Di Battista and Roberto Tamassia. 'On-line Maintenance of Triconnected Components with SPQR-Trees'. In: *Algorithmica* 15.4 (1996), pp. 302–318 (cit. on p. 86).

[24] Giuseppe Di Battista, Peter Eades, Roberto Tamassia and Ioannis G. Tollis. 'Algorithms for Drawing Graphs: an Annotated Bibliography'. In: *Computational Geometry: Theory and Applications* 4.5 (1994), pp. 235–282 (cit. on p. 41).

[25] Giuseppe Di Battista, Peter Eades, Roberto Tamassia and Ioannis G. Tollis. *Graph Drawing; Algorithms for the Visualization of Graphs*. Prentice Hall, July 1998 (cit. on pp. 29, 41, 46).

[26] Tim Dwyer. 'Scalable, Versatile and Simple Constrained Graph Layout'. In: *Computer Graphics Forum (EuroVis09)* 28.3 (June 2009), pp. 991–998 (cit. on pp. 47, 166).

[27] Tim Dwyer and Yehuda Koren. 'Dig-CoLa: Directed Graph Layout through Constrained Energy Minimization'. In: *IEEE Symposium on Information Visualization (InfoVis05)*. Ed. by Matt Ward and John Stasko. IEEE Computer Society, 2005, pp. 65–72 (cit. on pp. 47, 166).

[28] Tim Dwyer, Yehuda Koren and Kim Marriott. 'IPSep-CoLa: An Incremental Procedure for Separation Constraint Layout of Graphs'. In: *IEEE Transactions on Visualization and Computer Graphics (InfoVis06)* 12.5 (Sept. 2006), pp. 821–828 (cit. on pp. 47, 166).

[29] Tim Dwyer, Kim Marriott and Michael Wybrow. 'Topology Preserving Constrained Graph Layout'. In: *International Symposium on Graph Drawing (GD08)*. Ed. by Ioannis G. Tollis and Maurizio Patrignani. Vol. 5417. Lecture Notes in Computer Science. Springer, 2009, pp. 230–241 (cit. on pp. 47, 166).

[30] Peter Eades. 'A Heuristic for Graph Drawing'. In: *Congressus Numerantium* 42 (1984), pp. 149–160 (cit. on p. 46).

[31] Peter Eades. 'Graph Drawing Methods'. In: *Conceptual Structures: Knowledge Representation as Interlingua*. Ed. by Peter W. Eklund, Gerard Ellis and Graham Mann. Lecture Notes in Computer Science. Springer, 1996, pp. 40–49 (cit. on p. 41).

[32] Peter Eades, Wei Lai, Kazuo Misue and Kozo Sugiyama. 'Preserving the Mental Map of a Diagram'. In: *International Conference on Computational Graphics and Visualization Techniques (CompuGraphics91)*. Ed. by H. P. Santo. 1991, pp. 24–33 (cit. on p. 49).

[33] Leonhard Paul Euler. *Lettres à une princesse d'Allemagne sur divers sujets de physique et de philosophie*. St. Petersburg: Académie Imperiale des Sciences, 1768–1772 (cit. on pp. 17, 55, 169, 170).

[34] *FH Freedom of the Press*. Freedom House. 2010. URL: `http://www.freedomhouse.org/template.cfm?page=16` (visited on 10/12/2010) (cit. on p. 23).

[35] István Fáry. 'On Straight Line Representation of Planar Graphs'. In: *Acta Scientiarum Mathematicarum* 11.4 (1948), pp. 229–233 (cit. on p. 44).

[36] Jean-Daniel Fekete, Jarke J. van Wijk, John Stasko and Chris North. 'The Value of Information Visualization'. In: *Information Visualization*. Ed. by Andreas Kerren, John Stasko, Jean-Daniel Fekete and Chris North. Vol. 4950. Lecture Notes in Computer Science. Springer, Sept. 2008, pp. 1–18 (cit. on p. 13).

[37] Andrew Fish and Gem Stapleton. 'Defining Euler Diagrams: Choices and Consequences'. In: *International Workshop on Euler Diagrams (Euler05)*. 2005, pp. 34–37 (cit. on p. 53).

[38] Andrew Fish and Gem Stapleton. 'Formal Issues in Languages Based on Closed Curves'. In: *International Conference on Distributed Multimedia Systems, Visual Languages and Computing*. 2006, pp. 161–167 (cit. on p. 53).

[39] Jean Flower, Andrew Fish and John Howse. 'Euler diagram generation'. In: *Journal of Visual Languages and Computing* 19.6 (Dec. 2008), pp. 675–694 (cit. on pp. 53, 56, 60, 63, 65).

[40] Jean Flower and John Howse. 'Generating Euler Diagrams'. In: *International Conference on Diagrams (Diag02)*. Ed. by Mary Hegarty, Bernd Meyer and N. Hari Narayanan. Vol. 2317. Lecture Notes in Computer Science. Springer, 2002, pp. 285–299 (cit. on pp. 59, 64).

[41] Jean Flower, Peter Rodgers and Paul Mutton. 'Layout Metrics for Euler Diagrams'. In: *International Conference on Information Visualisation (IV03)*. Ed. by Ebad Banissi, Katy Börner, Chaomei Chen et al. IEEE Computer Society, 2003, pp. 272–280 (cit. on p. 66).

[42] Arne Frick, Andreas Ludwig and Heiko Mehldau. 'A Fast Adaptive Layout Algorithm for Undirected Graphs'. In: *International Symposium on Graph Drawing (GD94)*. Ed. by Roberto Tamassia and Ioannis G. Tollis. Vol. 894. Lecture Notes in Computer Science. Springer, 1995, pp. 388–403 (cit. on pp. 46, 115).

[43] Michael Friendly. 'A Brief History of Data Visualization'. In: *Handbook of Data Visualization*. Ed. by Chun-houh Chen, Wolfgang H ardle and Antony Unwin. Vol. 2. Springer Handbooks of Computational Statistics. Springer, 2008, pp. 15–56 (cit. on p. 9).

[44] Michael Friendly and Daniel J. Denis. *Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization*. 2001. URL: `http://datavis.ca/milestones/` (visited on 11/2010) (cit. on p. 9).

[45] Thomas M.J. Fruchterman and Edward M. Reingold. 'Graph Drawing by Force-Directed Placement'. In: *Software: Practice and Experience* 21.11 (Nov. 1991), pp. 1129–1164 (cit. on pp. 46, 47, 125).

[46] Pawel Gajer, Michael T. Goodrich and Stephen G. Kobourov. 'A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs'. In: *Computational Geometry: Theory and Applications* 29.1 (Sept. 2004), pp. 3–18 (cit. on p. 47).

[47] Pawel Gajer and Stephen G. Kobourov. 'GRIP: Graph Dawing with Intelligent Placement'. In: *Journal of Graph Algorithms and Applications* 6.3 (2002), pp. 203–224 (cit. on p. 47).

[48] Emden R. Gansner, Yifan Hu and Stephen G. Kobourov. 'GMap: Visualizing Graphs and Clusters as Maps'. In: *IEEE Pacific Visualization Symposium (PacificVis10)*. Ed. by Stephen C. North, Han-Wei Shen and Jarke J. van Wijk. IEEE Computer Society, 2010, pp. 201–208 (cit. on pp. 71, 72).

[49] Emden Gansner, Yehuda Koren and Stephen C. North. 'Graph Drawing by Stress Majorization'. In: *International Symposium on Graph Drawing (GD04)*. Ed. by János Pach. Vol. 3383. Lecture Notes in Computer Science. Springer, 2004, pp. 239–250 (cit. on p. 48).

[50] Angel García et al. 'Extensive Analysis of the Human Platelet Proteome by Two-Dimensional Gel Electrophoresis and Mass Spectrometry'. In: *Proteomics* 4.3 (2004), pp. 656–668 (cit. on p. 155).

[51] Martin Grueber and Tim Studt. '2011 Global R&D Funding Forecast: Stability Returns to R&D Funding'. In: *R&D Magazine* (Dec. 2010) (cit. on p. 22).

[52] Luc Guerrier et al. 'Exploring the Platelet Proteome via Combinatorial, Hexapeptide Ligand Libraries'. In: *Journal of Proteome Research* 6.11 (2007), pp. 4290–4303 (cit. on p. 155).

[53] Carsten Gutwenger and Petra Mutzel. 'Planar Polyline Drawings with Good Angular Resolution'. In: *International Symposium on Graph Drawing (GD98)*. Ed. by Sue Whitesides. Vol. 1547. Lecture Notes in Computer Science. Springer, 1998, pp. 167–182 (cit. on p. 46).

[54] Stefan Hachul and Michael Jünger. 'Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm'. In: *International Symposium on Graph Drawing (GD04)*. Ed. by János Pach. Vol. 3383. Lecture Notes in Computer Science. Springer, 2004, pp. 285–295 (cit. on p. 47).

[55] Ronny Hadany and David Harel. 'A Multi-Scale Algorithm for Drawing Graphs Nicely'. In: *Discrete Applied Mathematics* 113.1 (2001), pp. 3–21 (cit. on p. 47).

[56] Dong-Han Ham. 'The State of the Art of Visual Analytics'. In: *EU-Korea Conference on Science and Technology (EKC09)*. Ed. by Joung Hwan Lee, Habin Lee and Jung-Sik Kim. Vol. 135. Springer Proceedings Physics. Springer, 2010, pp. 213–222 (cit. on p. 13).

[57] William R. Hamilton. *Lectures on Metaphysics and Logic*. Edited by Henry L. Mansel and John Veitch. Gould and Lincoln, 1859 (cit. on pp. 17, 169).

[58] David Harel and Yehuda Koren. 'A Fast Multi-scale Method for Drawing Large Graphs'. In: *Journal of Graph Algorithms and Applications* 6.3 (2002), pp. 179–202 (cit. on p. 47).

[59] Ivan Herman, Guy Melançon and M. Scott Marshall. 'Graph Visualization and Navigation in Information Visualization: A Survey'. In: *IEEE Transactions on Visualization and Computer Graphics* 6.1 (2000), pp. 24–43 (cit. on pp. 41, 44).

[60] Danny Holten. 'Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data'. In: *IEEE Transactions on Visualization and Computer Graphics (InfoVis06)* 12.5 (Sept. 2006), pp. 741–748 (cit. on pp. 14, 165).

[61] *IMDb Top 250 Movies as Voted by Our Users*. The Internet Movie Database. 2011. URL: `http://uk.imdb.com/chart/top` (visited on 29/09/2011) (cit. on p. 153).

[62] Takayuki Itoh, Chris Muelder, Kwan-Liu Ma and Jun Sese. 'A Hybrid Space-Filling and Force-Directed Layout Method for Visualizing Multiple-Category Graphs'. In: *IEEE Pacific Visualization Symposium (PacificVis09)*. Ed. by Peter Eades, Thomas Ertl and Han-Wei Shen. IEEE Computer Society, 2009, pp. 121–128 (cit. on p. 155).

[63] Dieter Jungnickel. *Graphs, Networks and Algorithms*. 3rd ed. Vol. 5. Algorithms and Computation in Mathematics. Springer, Nov. 2007 (cit. on p. 29).

[64] Tomihisa Kamada and Satoru Kawai. 'An Algorithm for Drawing General Undirected Graphs'. In: *Information Processing Letters* 31.1 (1989), pp. 7–15 (cit. on pp. 46, 48).

[65] Daniel Keim et al. 'Visual Analytics: Definition, Process, and Challenges'. In: *Information Visualization*. Ed. by Andreas Kerren, John Stasko, Jean-Daniel Fekete and Chris North. Vol. 4950. Lecture Notes in Computer Science. Springer, Sept. 2008, pp. 154–175 (cit. on p. 13).

[66] Kurt Koffka. *Principles of Gestalt Psychology*. Harcourt, Brace & Co., 1935 (cit. on p. 6).

[67] Urs Lewandrowski et al. 'Platelet Membrane Proteomics: A Novel Repository for Functional Research'. In: *Blood* 114.1 (2009), e10–e19 (cit. on p. 155).

[68] Lennart Martens et al. 'The Human Platelet Proteome Mapped by Peptide-Centric Proteomics: A Functional Protein Profile'. In: *Proteomics* 5.12 (2005), pp. 3193–3204 (cit. on p. 155).

[69] Cathleen McGrath, Jim Blythe and David Krackhardt. 'The Effect of Spatial Arrangement on Judgments and Errors in Interpreting Graphs'. In: *Social Networks* 19.3 (1997), pp. 223–242 (cit. on p. 44).

[70] *Metabolic Data of Endosymbiontic, Parasitic and Free Bacteria*. Université Claude Bernard Lyon 1. 2011. URL: `http://pbil.univ-lyon1.fr/software/symbiocyc/index.php` (visited on 29/09/2011) (cit. on p. 156).

[71] Miriah Meyer, Bang Wong, Mark Styczynski, Tamara Munzner and Hanspeter Pfister. 'Pathline: A Tool For Comparative Functional Genomics'. In: *Computer Graphics Forum (EuroVis10)* 29.3 (June 2010), pp. 1043–1052 (cit. on p. 14).

[72] Luana Micallef and Peter Rodgers. 'Force-Directed Layout for Euler Diagrams'. In: *Compendium of IEEE Information Visualization 2009 (InfoVis'09)*. Vol. 15. IEEE Computer Society, Oct. 2009 (cit. on pp. 66, 67).

[73] Kazuo Misue, Peter Eades, Wei Lai and Kozo Sugiyama. 'Layout Adjustment and the Mental Map'. In: *Journal of Visual Languages and Computing* 6.2 (1995), pp. 183–210 (cit. on p. 49).

[74] Shin-ichi Nakano. 'Planar Drawings of Plane Graphs'. In: *IEICE Transactions on Information and Systems* E83-D.3 (Mar. 2000), pp. 384–391 (cit. on p. 45).

[75] Takao Nishizeki and Md. Saidur Rahman. *Planar Graph Drawing*. Vol. 12. Lecture Notes Series on Computing. World Scientific, Sept. 2004 (cit. on pp. 41, 44).

[76] Donald A. Norman. *Things that Make Us Smart*. Addison-Wesley, 1994 (cit. on p. 4).

[77] John J. O'Connor and Edmund F. Robertson. *MacTutor History of Mathematics archive*. School of Mathematics and Statistics, University of St. Andrews, Scotland. 1991. URL: `http://www-history.mcs.st-and.ac.uk/` (visited on 11/2010) (cit. on p. 169).

[78] OECD. *Science, Technology and Industry Outlook 2008*. OECD Publishing, Oct. 2008 (cit. on p. 22).

[79] Helen C. Purchase. 'Which Aesthetic has the Greatest Effect on Human Understanding?' In: *International Symposium on Graph Drawing (GD97)*. Ed. by Giuseppe Di Battista. Vol. 1353. Lecture Notes in Computer Science. Springer, 1997, pp. 248–261 (cit. on p. 44).

[80] Helen C. Purchase, David A. Carrington and Jo-Anne Allder. 'Empirical Evaluation of Aesthetics-based Graph Layout'. In: *Empirical Software Engineering* 7.3 (2002), pp. 233–255 (cit. on p. 44).

[81] Aaron Quigley and Peter Eades. 'FADE: Graph Drawing, Clustering, and Visual Abstraction'. In: *International Symposium on Graph Drawing (GD00)*. Ed. by Joe Marks. Vol. 1984. Lecture Notes in Computer Science. Springer, 2001, pp. 197–210 (cit. on pp. 47, 125).

[82] Amir H. Qureshi et al. 'Proteomic and Phospho-Proteomic Profile of Human Platelets in Basal, Resting State: Insights into Integrin Signaling'. In: *PLoS ONE* 4.10 (Oct. 2009), e7627–e7642 (cit. on p. 155).

[83] Nathalie H. Riche and Tim Dwyer. 'Untangling Euler Diagrams'. In: *IEEE Transactions on Visualization and Computer Graphics (InfoVis10)* 16.6 (Nov. 2010), pp. 1090–1099 (cit. on pp. 69, 70).

[84] Peter Rodgers, Leishi Zhang and Andrew Fish. 'General Euler Diagram Generation'. In: *International Conference on Diagrams (Diag08)*. Ed. by Gem Stapleton, John Howse and John Lee. Vol. 5223. Lecture Notes in Computer Science. Springer, 2008, pp. 13–27 (cit. on pp. 55, 56, 58, 65).

[85] Peter Rodgers, Leishi Zhang, Gem Stapleton and Andrew Fish. 'Embedding Wellformed Euler Diagrams'. In: *International Conference on Information Visualisation (IV08)*. Ed. by Ebad Banissi, Liz J. Stuart, Mikael Jern et al. IEEE Computer Society, 2008, pp. 585–593 (cit. on pp. 61, 65, 66, 166).

[86] Satu Elisa Schaeffer. 'Graph clustering'. In: *Computer Science Review* 1.1 (2007), pp. 27–64 (cit. on p. 1).

[87] Walter Schnyder. 'Embedding Planar Graphs on the Grid'. In: *SODA*. 1990, pp. 138–148 (cit. on p. 44).

[88] Ben Shneiderman. 'The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations'. In: *IEEE Symposium on Visual Languages (VL96)*. July 1996, pp. 336–343 (cit. on p. 13).

[89] Paolo Simonetto and David Auber. 'An Heuristic for the Construction of Intersection Graphs'. In: *International Conference on Information Visualisation (IV09)*. Ed. by Ebad Banissi, Liz J. Stuart, Theodor G. Wyeld et al. IEEE Computer Society, 2009, pp. 673–678 (cit. on p. 163).

[90] Paolo Simonetto and David Auber. 'Visualise Undrawable Euler Diagrams'. In: *International Conference on Information Visualisation (IV08)*. Ed. by Ebad Banissi, Liz J. Stuart, Mikael Jern et al. IEEE Computer Society, 2008, pp. 594–599 (cit. on pp. 55, 70, 163).

[91] Paolo Simonetto, David Auber and Daniel Archambault. 'Fully Automatic Visualisation of Overlapping Sets'. In: *Computer Graphics Forum (EuroVis09)* 28.3 (June 2009), pp. 967–974 (cit. on pp. 56, 58, 138, 163).

[92] Paolo Simonetto, Daniel Archambault, David Auber and Romain Bourqui. 'ImPrEd: An Improved Force-Directed Algorithm that Prevents Nodes from Crossing Edges'. In: *Computer Graphics Forum (EuroVis11)* 30.3 (June 2011), pp. 1071–1080 (cit. on pp. 163, 164).

[93] Mads Søgaard. *Gestalt Principles of Form Perception*. Interaction-Design.org. 2010. URL: `http://www.interaction-design.org/encyclopedia/gestalt_principles_of_form_perception.html` (visited on 11/2010) (cit. on p. 8).

[94] Gem Stapleton, John Howse, Peter Rodgers and Leishi Zhang. 'Generating Euler Diagrams from Existing Layouts'. In: *Layout of Software Engineering Diagrams (LED08)*. Ed. by Andrew Fish, Alexander Knapp and Harald Störrle. Vol. 13. Electronic Communications of the EASST. The European Association for the Study of Science and Technology, Sept. 2008, pp. 16–31 (cit. on p. 67).

[95] Gem Stapleton, Peter Rodgers, John Howse and Leishi Zhang. 'Inductively Generating Euler Diagrams'. In: *IEEE Transactions on Visualization and Computer Graphics* 17.1 (Jan. 2011), pp. 88–100 (cit. on p. 67).

[96] Gem Stapleton, Leishi Zhang, John Howse and Peter Rodgers. 'Inductively Generating Euler Diagrams'. In: *IEEE Transactions on Visualization and Computer Graphics* 17.7 (July 2011), pp. 1020–1032 (cit. on p. 68).

[97] Gem Stapleton, Peter Rodgers, John Howse and John Taylor. 'Properties of Euler diagrams'. In: *Layout of Software Engineering Diagrams (LED07)*. Ed. by Andrew Fish, Alexander Knapp and Harald Störrle. Vol. 7. Electronic Communications of the EASST. The European Association for the Study of Science and Technology, Sept. 2007, pp. 2–16 (cit. on pp. 52, 53).

[98] Sherman K. Stein. 'Convex Maps'. In: *Proceedings of the American Mathematical Society* 2.3 (1951), pp. 464–466 (cit. on p. 44).

[99] *THE World University Rankings*. Times Higher Education. 2010. URL: `http://www.timeshighereducation.co.uk` (visited on 10/12/2010) (cit. on p. 23).

[100] *TI Corruption Perception Index*. Transparency International. 2010. URL: `http://www.transparency.org/policy_research/surveys_indices/cpi` (visited on 10/12/2010) (cit. on p. 23).

[101] Alexandru C. Telea. *Data Visualization: Principles and Practice*. A.K. Peters, 2008 (cit. on pp. 11–13).

[102] James J. Thomas and Kristian A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society Press, 2005 (cit. on p. 13).

[103] William T. Tutte. 'How to Draw a Graph'. In: *Proceedings of the London Mathematical Society* 3.13 (1963), pp. 743–768 (cit. on pp. 46, 63).

[104] John Venn. 'On the Diagrammatic and Mechanical Representation of Propositions and Reasonings'. In: *Dublin Philosophical Magazine and Journal of Science* 10.59 (1880), pp. 1–18 (cit. on p. 17).

[105] John Venn. *Symbolic Logic: Revised and Rewritten*. 2nd ed. AMS Chelsea Publishing, Sept. 2007 (cit. on pp. 24, 25).

[106] John Venn. *Symbolic Logic*. London: MacMillan & Co., 1881 (cit. on pp. 17, 170).

[107] Anne Verroust and Marie-Luce Viaud. 'Ensuring the Drawability of Extended Euler Diagrams for up to 8 Sets'. In: *International Conference on Diagrams (Diag04)*. Ed. by Alan F. Blackwell, Kim Marriott and Atsushi Shimojima. Vol. 2980. Lecture Notes in Computer Science. Springer, 2004, pp. 128–141 (cit. on pp. 53, 56, 57, 64, 65).

[108] Kurt Wagner. 'Bemerkungen zum Vierfarbenproblem'. In: *Jahresbericht der Deutschen Mathematiker-Vereinigung* 46.1 (1936), pp. 26–32 (cit. on p. 44).

[109] Chris Walshaw. 'A Multilevel Algorithm for Force-Directed Graph Drawing'. In: *Journal of Graph Algorithms and Applications* 7.3 (2003), pp. 253–285 (cit. on p. 47).

[110] Colin Ware. 'Design as Applied Perception'. In: *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. Ed. by John M. Carroll. Morgan Kaufmann, May 2003, pp. 11–26 (cit. on p. 8).

[111] Colin Ware. *Information Visualization: Perception for Design*. 2nd ed. Morgan Kaufmann, 2004 (cit. on pp. 5, 99).

[112] Nicholas A. Watkins et al. 'A HaemAtlas: characterizing gene expression in differentiated human blood cells'. In: *Blood* 113.19 (2009), e1–e9 (cit. on p. 155).

[113] Christian Weise. *Nucleus Logicæ Weisianæ*. 1712 (cit. on p. 17).

[114] Dominic J.A. Welsh and Martin B. Powell. 'An Upper Bound for the Chromatic Number of a Graph and its Application to Timetabling Problems'. In: *The Computer Journal* 10.1 (1967), pp. 85–86 (cit. on p. 99).

Index