TU WIEN Informatics

A Holistic Approach for Metabolic Pathway Visualization

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Visual Computing

eingereicht von

Stefanie Mistelbauer, BSc
Matrikelnummer 00727540

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dr. Hsiang-Yun Wu

Wien, 13. März 2023

Stefanie Mistelbauer	Eduard Gröller

TU WIEN Informatics

A Holistic Approach for Metabolic Pathway Visualization

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Visual Computing

by

Stefanie Mistelbauer, BSc
Registration Number 00727540

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dr. Hsiang-Yun Wu

Vienna, 13th March, 2023

Stefanie Mistelbauer	Eduard Gröller

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Stefanie Mistelbauer, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 13. März 2023

Stefanie Mistelbauer

Danksagung

Ich möchte meiner Mitbetreuerin, Hsiang-Yun Wu, für ihre tatkräftige Unterstützung und Beratung bei der Erstellung dieser Arbeit danken. Ich habe viel von ihrer Erfahrung und ihrem Fachwissen gelernt. Insbesondere haben die vielen Diskussionsstunden mein Interesse für das Gebiet der biologischen Netzwerkvisualisierung geweckt. Ich möchte auch meinem Betreuer, Eduard Gröller, für seine wertvolle Unterstützung, seine Vorschläge und sein konstruktives Feedback während des Schreibens dieser Arbeit danken.

Ein besonderer Dank geht an meinen Ehemann Gabriel, der mich während dieser Arbeit motiviert und ermutigt hat. Nicht zuletzt bin ich meiner Schwester und meinen Eltern für ihre Unterstützung während meines Studiums an der Universität dankbar.

Acknowledgements

I would like to thank my co-supervisor, Hsiang-Yun Wu, for her extensive support and advice during the writing of this thesis. I learned a lot from her experience and technical knowledge. Particularly were the many hours of discussion, which got me interested in the field of biological network visualization. I would also like to thank my supervisor, Eduard Gröller, for his valuable guidance, input, and constructive feedback during the writing of this thesis.

Special thanks goes to my husband Gabriel, for keeping me motivated and encouraged throughout this thesis. Last but not least, I am grateful to my sister and parents, for their support during my study at the university.

Kurzfassung

Stoffwechselwege stellen miteinander verbundene Reaktionen chemischer Entitäten dar, die in Zellen stattfinden. Diese Wege werden in domänenspezifischen Notationen dargestellt, welche in den Biowissenschaften dem Wissensaustausch dienen. Da sie Tausende von Knoten enthalten können, sind automatische Anordnungen erforderlich, die die Bedeutung dieser Wege bewahren. Es gibt viele Algorithmen zum Zeichnen von Graphen, darunter hierarchische, topologisch-metrische, kraftbasierte und bedingungsbasierte Ansätze. Diese berücksichtigen in der Regel nur eine Teilmenge der Anforderungen, die für eine getreue Visualisierung von Stoffwechselwegen erforderlich sind, und unterstützen selten domänenspezifische Notationen. In dieser Arbeit stellen wir einen ganzheitlichen Ansatz zur Visualisierung von Stoffwechselwegen vor, der mit der Systems Biology Graph Notation (SBGN) konform ist. Unser Ansatz beginnt mit dem Laden eines Stoffwechselweges und dessen Abbildung auf einen gebündelten Graphen, um die Hierarchie seiner subzellulären Positionen zu modellieren. Die Knoten werden anschließend durch vektorisierte Stress-Majorisierung unter Verwendung domänenspezifischer Bedingungen in einem mehrstufigen Aufbau angeordnet. Dies führt zu einer SBGN-konformen Anordnung. Um bestimmte Reaktionen an subzellulären Orten zu unterscheiden, haben wir eine Visualisierungstechnik entwickelt, die in Analogie zu einem elastischen Band unterschiedliche Formen erzeugt. Um große Netzwerke zu erforschen, bieten wir eine Expansions- und Kollaps-Interaktion in Kombination mit einer Motiv-Vereinfachung. Wir bestimmen den Grad der Übereinstimmung der Anordnung mit der SBGN, indem wir domänenspezifische Qualitätsmetriken vorschlagen. Unsere Ergebnisse zeigen, dass die Formulierung von SBGN-spezifischen Bedingungen im Rahmen der vektorisierten Stress-Majorisierung machbar ist. Schließlich bestätigt unsere Auswertung, dass unser Anordnungsansatz Stoffwechselwege getreu darstellen kann.

Abstract

Metabolic pathways represent interconnected reactions of chemical entities, which take place within cells. These pathways are represented in domain-specific notations, which are used for knowledge exchange in the life sciences. Since they can contain thousands of nodes, automatic layouts are required that conserve the meaning of these pathways. There are many graph drawing algorithms including hierarchical, topology-shape-metric, force-directed, and constraint-based approaches. They typically consider only a subset of the requirements needed to faithfully visualize metabolic pathways and rarely support domain-specific notations. In this work, we present a holistic approach to visualize metabolic pathways compliant with the Systems Biology Graph Notation (SBGN). Our approach starts with loading a metabolic pathway and mapping it to a clustered graph structure to model the hierarchy of subcellular locations. The nodes are then arranged through vectorized stress majorization using domain-specific constraints in a multilevel setup. This leads to a SBGN-compliant layout. To distinguish certain reactions at subcellular locations, we developed a visualization technique that produces distinct shapes in analogy to an elastic band. To explore large pathways, we provide an expand and collapse interaction in combination with motif simplification. We determine the degree of the layout's compliance with the SBGN by proposing domain-specific quality metrics. Our results demonstrate that the formulation of SBGN-specific constraints in the framework of vectorized stress majorization is feasible. Finally, our evaluation corroborates that our layout approach can faithfully represent metabolic pathways.

Contents

CHAPTER 1

Introduction

In this chapter, the motivation for this thesis is presented and the problem statement is defined. An overview of the contributions is given and finally a short outline of the work.

1.1 Motivation

Graphs and networks are widely used to describe biochemical reactions in life sciences such as biology and chemistry. For example, a series of successive reactions within a cell are connected to form metabolic pathways. Those pathways transform, produce, or consume biochemical compounds, called metabolites, along the way. One cell is home to many metabolic pathways and linked together they form the metabolic network of a cell [80]. In short the transformations of metabolites into each other can be represented as metabolic networks [71]. Other biological processes have their own network representations, e.g., protein interaction networks or gene regulatory networks [80].

Visualizations of these biological networks support domain experts in gaining insights into biochemical processes [14] and in communicating this knowledge. To this end the life science community developed diverse visual representations. One of them is the Systems Biology Graph Notation (SBGN), which aspires to provide a unified and standardized visual language [77].

Biological networks are complex networks [76], that can contain several thousand nodes and edges [14], and are constantly growing [71]. This makes navigation, analysis, and comprehension very challenging [45]. Therefore, an automated visualization approach is needed [90] that supports SBGN and promotes understanding.

1.2 Problem Statement

On the one hand, there are existing software tools and libraries to visualize graphs and network, but they do not usually support the established drawing conventions of life sciences [14]. On the other hand, there are more than 170 specialized tools for modeling, analyzing, and visualizing biological pathways [14, 71]. However, most of these specialized tools only support standard force-directed or hierarchical graph drawing techniques and do not include state-of-the-art layout approaches. Kerren et. al. stated in 2002 that these standard techniques and even a combination of them do not yield visualizations, which satisfy the established conventions of biology and chemistry [71]. As a result, such visualizations tend to be difficult to understand [71]. Consequently, there is an open need for automatic layouts of biological networks with state-of-the-art techniques in the life sciences' drawing conventions [14].

In addition to the drawing conventions, an important aspect of metabolic networks is their nested nature. Biochemical processes take place within cells, subcellular locations, or molecular complexes. This is modeled through nested compound structures. Including this information yields additional insight into the reactions, because metabolites can only interact directly with others in the same compound structure. Such structures are rarely considered during automated layout calculations [45]. The integration of the metabolites' subcellular locations into the overall layout and conveying the nested structure of metabolic networks are still open problems [14].

Visualizing large networks is a common problem that exists throughout all domains and needs to be addressed for metabolic networks as well [14, 45]. Possible solutions are complexity reduction techniques such as focus+context [71, 104], expand+collapse [14], hierarchical clustering and nesting [45], visual aggregation [47], semantic zooming [71], or motif simplification [47, 106]. However, these techniques often change the topology and the existing layout, which affects the user's ability to retain the mental map. Therefore, a combined solution must be researched that not only reduces complexity, but also preserves the reader's mental map [14].

There are already works that study these problems independently with idealized data or data not specific to an application domain. Moreover, these works do not consider these problems as parts of a holistic problem that should be solved together.

1.3 Contributions

There is an open need for a layout strategy that takes the graphical notation of life sciences into account, visualizes a metabolic pathway's compound structure and data flow while keeping the complexity of the network minimal. In this work, we provide a rather holistic approach for the visualization of metabolic pathways. Our contributions are the following:

C1 A graph drawing technique that adheres to the SBGN and considers subcellular locations is proposed. This is accomplished by formulating domain-specific layout constraints in the general graph drawing framework of vectorized stress majorization [103] to arrange metabolic pathways. Because metabolic pathways can consist of disconnected components and disjoint subgraphs within subcellular locations, we augment the vectorized stress majorization with a multilevel layout approach.

C2 The visualization of large networks is improved twofold. Firstly, we provide a tailored cluster visualization technique for subcellular locations, called *elastic band*, that generates distinguishable shapes through analytical implicit blending. Secondly, we use these shapes to abstract subgraphs within subcellular locations to *motifs*. These motifs facilitate preserving the mental map and reduce the complexity of large metabolic pathways by means of an expand and collapse interaction.

C3 An end-to-end approach to create a metabolic pathway visualization in the SBGN is described. Firstly, a unified definition framework for graphs, networks, and graph drawing requirements is given, since the literature uses different terminology. Secondly, a holistic pipeline for creating an SBGN drawing is presented. This includes a hierarchical network structure that serves as the foundation for our SBGN-specific graph drawing and interaction techniques. Finally, we introduce quality metrics to evaluate the *SBGN-ness* of a drawing. Novel problems encountered during the development of this holistic approach are formulated.

We reference these contributions at the appropriate places in this thesis.

1.4 Outline of the Thesis

The state-of-the-art of relevant fields is reviewed in Chapter 2. The definitions of graphs and networks are reviewed and set into context of biological pathways. An overview of the visual mapping of networks to an application domain is given, along with an explanation of the SBGN. Graph drawing requirements and the domain-specific layout requirements of SBGN are discussed and analyzed, followed by an overview of techniques for graph drawing and visualizing cluster information. Various databases for biological pathway

data and existing software support are listed at the end of this chapter. In Chapter 3 we present our pipeline to generate visualizations of metabolic networks in SBGN. It consists of the following seven steps: acquire and prepare, load and preprocess, data structure, layout, postprocess, visualize, and reduce complexity. Subsequently, we describe every step in detail, including its relevant graph drawing requirements, examples, and potential alternative approaches. In the final part of this chapter we describe our implementation. In Chapter 4 we present metabolic networks arranged with our approach and discuss them in comparison with manually-curated layouts from domain experts. To evaluate the SBGN-ness of our results, we introduce several quality metrics. In the final Chapter 5, we conclude our work. We also discuss open problems and future improvements.

CHAPTER 2

Related Work

The first visualizations of metabolic networks were hand-made and appeared in textbooks and on posters. In the 1990ies, electronic information systems such as the Kyoto Encyclopedia of Genes and Genomes (KEGG) [69] emerged, which is an online database of manually created maps of biological systems such as biological processes. They were viewable but not changeable and, therefore, only static visualizations. Fueled by the accessibility of biological networks through these online databases, the visualization of biological networks has since evolved to dynamic visualizations, that aim to be generated automatically and on-demand, are interactive, and can be visually analyzed and explored.

The purpose of visualizing a biological network is to promote understanding of biomolecules' interactions to the reader. This can by achieved by translating the underlying graph structure of a biological network into a diagram [19, 106]. A self-explanatory way to visualize a graph is a *node-link diagram* [98], where data entities are depicted as nodes and relationships as lines [23, 65]. The placement of these nodes and links must be done in a meaningful way [84], as it has a significant impact on the readers ability to understand the visualized biological system [101]. An arbitrary placement, would hide the underlying structure of the data [84]. Since biological networks are multivariate networks and not just simple graph structures, it is not sufficient to map them to simple node-link diagrams and apply a standard drawing algorithm to obtain a meaningful placement [71]. The domain-specific information of biological networks can be translated into visual representations using visual metaphors. Such a translation is called *visual mapping* [71]. The definition of a node's appearance is often depending on the application domain, e.g., biological networks typically use glyph-based definitions [71]. Incorporating this contextual information can improve biological network diagrams in general [19].

Beside node-link diagrams, there are other ways to visualize graphs such as space-filling techniques, treemaps, matrix representations, or 3D layouts [98]. However these are rarely used to visualize biological networks and are only mentioned for completeness. The knowledge and techniques from many different research areas are combined to provide

the foundation for biological network visualization, such as graph theory, information visualization, and graph drawing [71]. There are a numerous surveys on these topics, including the following:

- Tarawneh et al. [98] gives a general introduction to graph visualization techniques,
- Kerren et al. [71] focus on biological network visualizations, and
- Gibson et al. [65] provide an overview of general force-directed layouts, dimensionality reduction, multilevel techniques, computational improvements, and comparisons.

In the subsequent sections, we provide an overview of the relevant foundations and describe the state-of-the-art techniques for visualizing biological networks and metabolic pathways.

2.1 Graph and Network Definitions

In many areas of the life sciences networks are used to model relational data [71]. Graphs and networks are also mathematical concepts and common data structures [68, 98]. Since the definitions and notations of graphs and networks slightly vary in the literature [22, 33, 46, 61, 71, 92, 96, 97, 98, 106], we provide a brief outline for the used definitions and notation **(C3)** throughout this thesis:

Definition 2.1.1 (Graph)**.** A *graph* $G(V, E)$, or in short G, consists of V, the finite set of *vertices*, and E, the finite set of *edges*. An edge consists of a pair of vertices (u, v) with $u, v \in V$, and describes a binary relationship between them.

Definition 2.1.2 (Undirected and directed)**.** If an edge is an unordered vertex pair $\{u, v\}$, where $\{u, v\} = \{v, u\}$, then the graph is called *undirected*. If an edge is an ordered vertex pair (u, v), where $(u, v) \neq (v, u)$, then the graph is called *directed*. A directed graph is also called *digraph*.

Definition 2.1.3 (Simple)**.** G is called *simple* if E does not contain self-loops and every edge is unique.

Definition 2.1.4 (Path)**.** A connected sequence of edges is called *path*.

Definition 2.1.5 (Acyclic)**.** If there is no possible path for any vertex to itself, the graph is *acyclic*.

Definition 2.1.6 (Directed acyclic graph)**.** If the graph is acyclic as well as a directed graph, it is called *Directed Acyclic Graph (DAG)*.

Definition 2.1.7 (Connected)**.** A graph is *connected* if there is a path between u and v for each pair of vertices (u, v).

Definition 2.1.8 (Bipartite)**.** A graph G is *bipartite* if it is 2-colorable.

Definition 2.1.9 (Neighborhood)**.** The adjacent vertices of a vertex are called its *neighborhood*.

Definition 2.1.10 (Degree)**.** The *degree* of a vertex is the number of its neighbors.

Definition 2.1.11 (Subgraph)**.** If $G \cup G' = G$, then G' is a *subgraph* of G. A subgraph $G'(V', E')$ can be *induced* by $V' \subseteq V$, then the set of edges E' consists of edges where $(u, v) \in V'$. A subgraph $G'(V', E')$ of $G(V, E)$ can also be *induced* by $E' \subseteq E$, then the set V' consists of vertices incident to E'.

Definition 2.1.12 (Hypergraph)**.** The *hypergraph* $H = (V, E)$ consists of the finite vertex set V and the *hyperedge* set E, where $e \in E$ is a subset $\{u_0, \ldots, u_n\}$ of V compared to a conventional edge, that consists of a pair of vertices $\{u, v\}$.

Definition 2.1.13 (Network)**.** A *network* N is a graph where each vertex and/or edge is associated with an attribute $w(v) \in \mathbb{R}$ or $w(e) \in \mathbb{R}$, respectively. This attribute can represent a length, a weight, a capacity, a cost, etc. The network is *multivariate* if it has more than one attribute attached to its vertices and/or edges.

Definition 2.1.14 (Multi layer/level network)**.** The graph is partitioned into layers (levels), such that a node of a level is connected to a node residing in the next level, resulting in a multilevel digraph.

Definition 2.1.15 (Clustered graph)**.** A *clustered graph* is a multivariate network, where every vertex is part of at least one cluster. Cluster membership can be encoded as vertex attribute. Clusters can either be disjoint or form a hierarchy, with the latter being represented as *cluster tree*.

Definition 2.1.16 (Embedding)**.** The mapping of a graph into $\mathbb{R}^2$ is also called its *embedding*.

Definition 2.1.17 (Drawing)**.** A graph can be represented as a *drawing* Γ in $\mathbb{R}^2$ if its vertices are mapped to points on a plane $\Gamma(v)$ and if its edges are mapped to simple open Jordan curves $\Gamma(u, v)$. A Jordan curve has no self-intersections in $\mathbb{R}^2$.

Definition 2.1.18 (Layout)**.** An arrangement of vertices and edges in a drawing is called layout [84]. Whenever we use the term graph drawing in this thesis, we assume that such a drawing includes a layout.

Definition 2.1.19 (Network diagram)**.** A drawing of a network, is called a *network diagram* or *a network map*. We refer to the visualization of a network as network diagram.

Definition 2.1.20 (Planar)**.** A graph is *planar* if it assumes a drawing without edge crossings. A planar graph separates a plane into *faces*.

2.2 Biological Pathways as Graph Structure

Biological pathways are complex real world networks, which cannot be completely represented by one simple graph structure. To support the multivariate nature of biological pathways, a suitable data structure needs to be chosen [71]. The graph structure is also shaped by the underlying application, use-case, or questions being addressed.

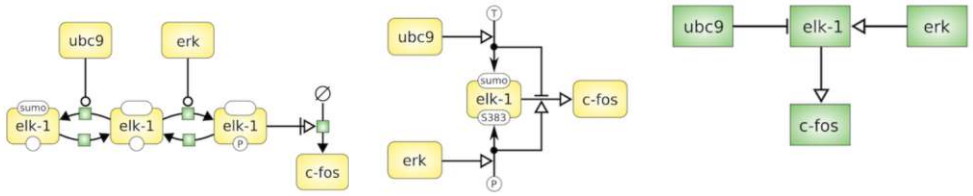
Simple graphs are used for substrate graphs, where every vertex represents a substrate, and reaction graphs, where every node stands for a reaction [106]. Metabolic networks can be represented as directed graphs, to convey which entities are consumed to produce other entities [71]. Another common way to map biological pathways to a graph structure is by mapping, both metabolites and reactions to vertices. This leads to a bipartite graph. Alternatively, reactions can be mapped to edges instead of vertices, which leads to a hypergraph, since more than two entities can be involved in a reaction. A hypergraph can usually be converted into a bipartite graph and vice-versa.

Wu et al. [106] proposed a multilayer graph, which is a graph $G = (V, E)$ with additional layers $L = \{l_1, l_2, \ldots, l_n\}$ representing domain-specific information. For example, subcellular locations of a metabolic network can be modeled through these layers. The multilayer graph can be transformed into other graph structures, providing access to a wide range of common graph algorithms [106].

A clustered graph, a specific transformation of a multilayer graph, can be used to show subcellular locations of reactions and metabolites. These locations are stored as vertex attributes, and reactions as well as metabolites belonging to the same subcellular location are grouped in the same cluster. Usually, clusters are made disjoint by duplicating unimportant, highly-connected entities, e.g., Adenosintriphosphat (ATP) [106]. We model biological pathways as clustered graphs throughout this thesis.

2.3 Systems Biology Graph Notation

In order for authors and readers of diagrams to communicate with each other, they must agree on the glyphs used, their arrangement rules, and how to interpret the results [77]. If there is no uniform definition for these aspects, it is a time-consuming task to become familiar with and understand each unique graphical notation, which also opens up room for misinterpretation [71]. There were many of these non-standardized visual mappings in the literature, which were only designed for special use-cases. They described biochemical interactions, inter- and intracellular signaling gene regulations, and others. Later, by a staggering increase of data, the need for a standardized, unambiguous graphical notation grew. The establishment and use of such notations made it possible for other fields to thrive, e.g., electronics industry [77], but finding a suitable visual representation for an application domain is challenging [71]. The success of a standardized notation depends on its acceptance in a diverse community as well as its fulfillment of the practical needs of that community [77]. For our use-case this community consists of biologists, biochemists,



(a) (b) (c)

Figure 2.1: The three SBGN diagrams: (a) process diagram, (b) entity relationship diagram, (c) activity diagram. They visualize the same data, but focus on different aspects of the biological information. (Images adapted from Le Novere et al. [77])

bioinformaticians, geneticists, and software engineers [77]. Several attempts were made to define a uniform representation for biological networks, e.g., Molecular Interaction Maps (MIM) and the notation used by KEGG, but none of them reached community standard [77].

Since 2006, a diverse community of biochemists, modelers, and computer scientists thrive to establish the Systems Biology Graph Notation (SBGN) as community standard for biological networks [71, 77]. This notation is specifically designed to be visually and syntactically as well as semantically consistent, concise, modular, scalable, unambiguous, and to support automated diagram generation, while still being easy to learn. With the SBGN, the community aims not only to establish an efficient and accurate representation of biological knowledge, but also to preserve this knowledge and provide open-access. This knowledge includes, for example, gene regulation, metabolism, and cellular signaling [77].

Because representing all possible reactions and interactions in one diagram would be too complex, the SBGN is designed as three complementary notations to keep the diagrams comprehensible: the *process diagram*, the *entity relationship diagram*, and the *activity diagram*. These three diagrams are projections of the same biological information, but focus on different aspects [77]. Figure 2.1 shows the same biological information depicted by these three diagrams.

The *process diagram* (see Figure 2.1a) depicts biological entities and the molecular processes and interactions in which they are consumed to produce new biological entities. It also represents the temporal flow of these transitions, the different states assumed by entities, as well as the subcellular locations they occur in. Since the process diagram represents multiple states of entities, it has more nodes than the other two diagrams [77].

While the process diagram focuses on biological entities and their transitions, the *entity relationship diagram* (see Figure 2.1b) focuses on an entities' influence on the transformation of other entities. All entities appear only once, which makes the entity relationship diagram smaller than the process diagram and, therefore, easier to follow. The temporal flow of events is not as explicitly displayed and more difficult to comprehend [77, 95].

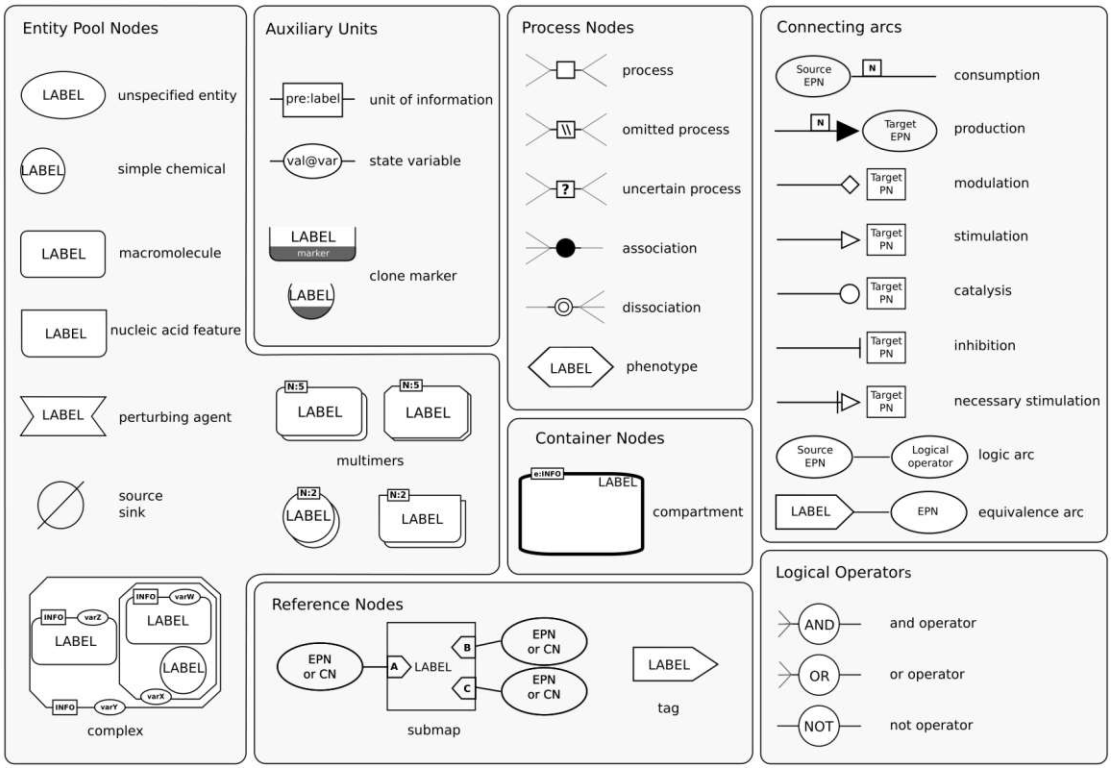


Figure 2.2: The glyphs of an SBGN process diagram. Entity pool nodes, like simple chemicals or macromolecules, are biological entities that participate in a chemical reaction. Different kinds of reactions are depicted as different process nodes. How entities react with each other is represented by connecting entity pool nodes to process nodes through connecting arcs. The different connection arc glyphs depict how the participances react with each other, e.g., an entity can be consumed while another is produced. If the reaction takes place within one cellular location it can be placed within a container node, e.g., a compartment. (Image taken from Moodie et al. [84])

The *activity diagram* (see Figure 2.1c) represents influences between entities directly and omits biochemical details of processes. Therefore, the number of nodes in this diagram is greatly reduced. Since this diagram could include ambiguities, it should only be used in addition to a process or a entity relationship diagram [77, 82].

These three diagrams of the SBGN define the visual mapping of nodes and edges, as well as their connectivity, but they do not specify their layout. We focus on progress diagrams, because they are very suitable to represent metabolic pathways [101]. The process diagram definition includes seven major types of glyphs (see Figure 2.2): (a) entity pool nodes, (b) auxiliary units, (c) process nodes, (d) container nodes, (e) reference nodes, (f) connecting arcs, and (g) logical operators [77]. In the subsequent paragraph, we only describe the most relevant glyphs for our work.

Entity pool nodes represent biological entities, whereas process nodes represent biochemical reactions. Entity pool nodes can only connect with arcs to the *handles* of process nodes. Hence, the process diagram is a visual representation of a bipartite graph [101]. Connecting arcs can represent the consumption or production of an entity through a reaction, but they can also represent the influence of an entity on a reaction. For example, an entity node connected to a process node by a catalysis arc symbolizes that its entity is a catalyst for that reaction. Multiple arcs can be connected to the same handle of a process node, which makes these arcs hyperedges and, consequently, the process diagram a hypergraph. Since arcs are read in a specific direction, the process diagram is a directed (hyper)graph. Compartments are a type of container node, which contain entity pool nodes and process nodes. An entity pool node can belong to multiple compartments, making the process diagram a clustered (hyper)graph [84].

Even though the SBGN does not encode concrete node placement, it acknowledges that a meaningful layout leads to insights about the underlying data. Therefore, the SBGN provides guidelines for the arrangement of SBGN process diagrams, in addition to their appearance. These apply to hand-drawn as well as to automatically drawn diagrams [84]. It is stated that the SBGN is independent of color and glyph size and no guidelines are provided for user interactions [84]. General and SBGN-specific requirements for the arrangement of diagrams is the topic of the next section.

2.4 Graph Drawing Requirements

To clarify the underlying meaning and structure of a diagram, the drawing of the diagram must be readable. Readability is related to aesthetics, so it is desirable to design an aesthetically pleasing diagram. Aesthetic aspects have the added effect of making the reader more receptive, and making them invest more effort in understanding [25]. Graph drawing research developed a set of requirements, whose adherence steers the creation of aesthetically pleasing graphs. They are based on intuition, experiments, on perceptual as well as Gestalt principles [25]. Usually, these requirements are quantitative measurements [71]. In the literature they are listed under different names: aesthetic heuristics [25], principles of graph drawing [65], graph drawing aesthetics, aesthetic criteria [71], and quality metrics [81, 86].

The task of arranging graphs to be pleasing and readable can be viewed from a syntactic or semantic perspective. While the syntax considers, e.g., edge crossings, node overlap, or edge length, the semantics deal with highlighting the important characteristics of the underlying data. Additionally to general semantic requirements, application-specific semantic requirements should influence the graph arrangement to support understanding [25].

In the following we list known graph drawing requirements that affect the positioning of nodes, links, or the diagram as a whole. These requirements were examined for their perceptual basis. Hence for each requirement, if known, we state the perceptual principles from which it derives [25]. We separate them into syntactic and semantic graph drawing

requirements and address the application-specific needs of the SBGN. We start with *syntactic requirements*:

R01 Node: Size [71, 84]. The nodes of a SBGN diagram have different sizes, depending on the used glyph and the encapsulated label. The SBGN regulates the placement of labels inside nodes. If possible they should be completely enclosed by their node.

R02 Node: Non-overlap [25, 65, 84, 92]. None of the graph nodes should overlap, there should be a minimum distance between them. An exception of this rule are the SBGN *Multimer* glyphs, which are visually represented by identical entity pool nodes stacked on top of each other, but are considered as a single node [84]. Nested nodes are also allowed, but they must be completely enclosed by their container node, e.g., the SBGN *complex* glyphs. These nodes can contain other entity nodes, but not process nodes or edges. This requirement aims to avoid ambiguity in the linking of nodes as well as a false impression of clustering. It originates in the perception of connectedness [25, 65]

R03 Node: Uniform distribution [65, 92, 98]. This requirement is also known as unit grid alignment. As the name suggests, the nodes are distributed on a uniform grid. This requirement prevents distortion, but it has no perceptional support [25, 65].

R04 Node: Orthogonality [65, 92, 84, 98]. It is also known as the alignment of nodes. It demands that a node's shape shall not be rotated, e.g., a rectangular node shape shall be axis-aligned. The SBGN extends this requirement to a node's label [84]. This requirement originates in the perception of orientation [25].

R05 Link: Crossings [25, 65, 68, 71, 84, 86, 92]. There is general agreement that there should be as few edge crossings as possible. If it is unavoidable, the crossings shall be as orthogonal as possible [84]. The SBGN states that edges should not cross themselves. It also states that labels associated with edges should be placed close to them, and should not cross other edges or their labels. This requirement aids the reader in finding paths, and improves the readability of the graph because it seems less complicated [65]. The perception of continuation provides the basis for this requirement [25].

R06 Link: Bends [25, 71, 84, 86, 98, 92]. In graphs as well as in SBGN diagrams, edge bends should be avoided, because they make it difficult to follow an edge. If an edge bend cannot be avoided, it should be introduced with the following considerations. The edge should be divided into equal parts. The edge should bend with a maximal angle, since a sharp angle could be perceived as two individual objects instead of one continuous entity. This requirement originates in the perceptions of similarity and continuation [25].

R07 Link: Uniform length [25, 65, 68, 71, 92]. Edges with the same length are more aesthetic, because regular structures prevent clutter. As a side effect the area of the graph is minimized [65]. This requirement is based on the perception rule of similarity [25].

R08 Link: Maximize orthogonality [25, 86, 92]. Edges should follow an imaginary grid. As a side effect, edge crossings are reduced and nodes are aligned on this grid, see (R03). Because of the perception of orientation, horizontal and vertical lines are more likely to be seen as a memorable figure [25].

R09 Node-Link-Connection: Minimize edge length [65, 84, 98]. Adjacent nodes should be close together because the perception of proximity implies a relationship, consequently non-adjacent nodes need to be further apart.

R10 Node-Link-Connection: Non-overlap [25, 65, 84, 92]. To avoid misinterpretation caused by proximity, edges and nodes which are not associated with each other need to be kept apart. Ideally, edges should also not overlap with nodes. But in a real-world application with highly-connected nodes, this is difficult to accomplish. If a node-link overlap cannot be avoided, the SBGN allows edges to pass over nodes and intersect them in two points. This requirement originates in the resolution limits of the human eye and the perception of proximity [25].

R11 Diagram: Aspect ratio [25]. The ratio of the graph should match that of its container, e.g., a screen or a page. This reduces the number of distinct shapes in the layout and thus, the complexity. This requirement has no basis in perceptional principles [25].

R12 Diagram: Minimize area [25, 71]. This is a side-effect of minimizing the edge length, and leads to the perception of a good figure [25].

These syntactic requirements were evaluated for their influence on the readability of a graph as well as a readers performance through user-studies [25, 86]. The outcome of these studies showed inconsistencies between the influence of syntactic requirements on general graphs and on application-specific graphs. For example, avoiding edge crossings (R05) and bends (R06) had a significant effect on readability for general graphs, but were less significant for application-specific graphs. Additionally, it is difficult to directly connect syntactic requirements to understanding, since they are influenced by a reader's environment, culture, and education. In contrast, semantic requirements are directly linked to understanding, because the meaning of the data is directly connected to the layout of the graph. The studies conclude, that semantics are equally or even more important than syntactics when creating diagrams [25, 86]. In the following we list known *semantic requirements*:

R13 Node: Fixed position [52, 65]. It should be possible to assign nodes of special interest to a fixed position, along one or both axis in 2D.

R14 Node: Clustering and containment [25, 72, 84, 92]. Nodes with a semantic relation should be placed in close proximity to form a cluster. Clusters can be placed in containers. To model such containers, the SBGN provides *compartments*. They group biological entities and processes that occur in the same subcellular location. The nodes representing these entities and processes as well as their edges are placed inside their associated compartment. Compartments can also contain other compartments. Consequently, one compartment in the SBGN represents exactly one cluster. A process node cannot be placed in a compartment where it is not connected to at least one biological entity. There is also the possibility to place nodes on a compartment's boundary, to convey e.g., that a process is taking place in the cell membrane [84]. In our work, we assume that the clustering into compartments is predefined by domain-experts. The SBGN suggests that a different background coloring of compartments may support the clarity of the diagram. This requirement is based in symmetry and proximity [25].

R15 Link: Flow direction [25, 84, 92]. A directed graph tells a story with a defined flow. This is best represented by ensuring that the graph follows a consistent overall flow direction. Cycles should flow in a clockwise direction [92]. In the SBGN the flow direction represents the temporal flow of processes. It is suggested to keep the main flow direction vertically or horizontally, starting from the top left corner [84]. The requirement of flow direction is based on the perception of orientation [25].

R16 Node-Link-Connection: Maximize minimum angles [25, 86, 92]. Edges connected to a node should be distributed evenly around it. As side-effect high-degree nodes, which are most likely important nodes, are placed at the center of the graph. Such a centering of important nodes moves them to the center of attention. Thus, this requirement gets a semantic meaning. This requirement is based on the resolution limits of the human eye [25].

R17 Node-Link-Connection: Connection points [84, 92]. Processes are represented in an SBGN diagram as *process nodes*. The process node's glyph consists of a square and two handles on opposing sides of the square. Consumption and production arcs are only allowed to be connected to one of these handles. In contrast, modulatory arcs can only be connected to the square's two sides without handles. A process glyph and its allowed connectivity is demonstrated in Figure 2.3. This requirement is not derived from the underlying graph structure, it is a semantic restriction defined in SBGN.

R18 Diagram: Similarity [25, 65, 84, 91, 92]. Since elements with common features, such as color or shape, appear grouped together, similar subgraphs

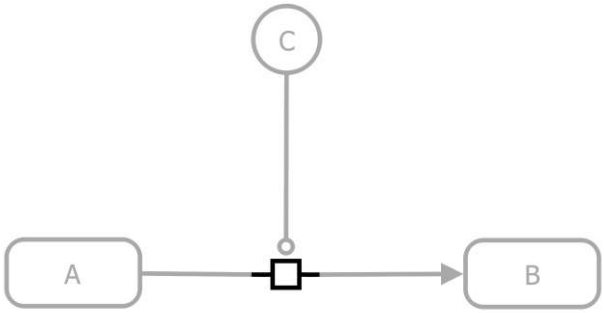


Figure 2.3: A process node and its required connectivity defined by other nodes. A process consists of a square (the center) and two handles, shown in black. Consumed and produced entities connect to these handles on opposing sides, while modulatory arcs connect to the center of the process perpendicular to the handles, shown in gray.

should be drawn in a similar manner. Such similar subgraphs, or reoccurring network structures, are also called *motifs* [47]. The SBGN also suggests using a standard layout for similar sub-components. Similarity itself is a perceptual principle [25].

R19 Diagram: Symmetry [25, 65, 68, 98, 86, 92]. To aid the reader in understanding the underlying graph structure, symmetric structures should be drawn symmetrically [65]. This way, they can be perceived as distinct figures. The symmetry should be maximized locally as well as globally [25]. The SBGN also suggests drawing sub-components involved in parallel and reversible processes symmetrically [92]. Symmetry itself is a perceptual principle [25].

R20 Diagram: Display structure of the data [52, 65, 98]. A graph's nodes should be structured or separated into different layers based on a semantic criterion, this helps to convey the underlying structure.

R21 Diagram: Preserving the mental map [92]. Traditionally, metabolic pathways are static layouts. Since they became dynamic, it is important to preserve the reader's mental map. If pathways are filtered, combined, or created, existing nodes and links need to stay at their previous positions, to prevent the reader from loosing orientation.

R22 Diagram: Node degree reduction through cloning [84, 92]. High-degree nodes could have a negative impact on overall readability. If edges connected to a high-degree node are distributed with an equal angle around it, i.e., maximizing the minimum angle (R16), this node moves into the center of the graph and, therefore, becomes focused. If this happens to a rather unimportant but highly connected node, e.g., a node representing ATP, it

distracts attention from more important nodes. To solve this problem the SBGN supports cloning of elements. Cloned nodes need to be marked with a black bar at the bottom. However, this should not make the diagram disjoint or more difficult to interpret.

R23 Diagram: Expand & collapse [84, 92]. To reduce complexity of the graph, the SBGN introduces the *submap* node. It hides its content and represents connections to elements inside the submap as *terminals*. In dynamic diagrams, submaps can be expanded and collapsed within the same diagram or be opened in another one.

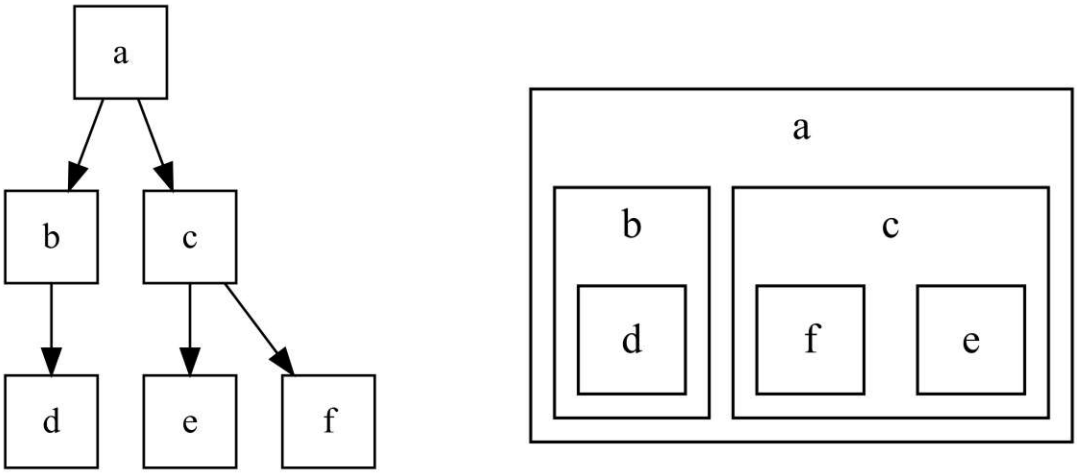
A diagram following the syntactic and semantic requirements may not always result in a better layout, it could also lead to an ambiguous and unintuitive graph [65]. In comparison, a human curator might choose to ignore certain requirements to achieve better readability [92]. Many of these requirements are also conflicting, e.g., node clustering is adverse to an even node distribution. Complying with one requirement means breaking another, therefore trying to satisfy them all is infeasible [25, 65]. Often it is less important that an individual requirement is satisfied, but that a combination and prioritization according to a task is fulfilled [86]. Finding an appropriate compromise between the requirements should be the goal [86, 98]. Usually, a trade-off between the requirements is made based on the semantics of the graph and its intended use [25]. The optimal way of selecting and balancing application-specific requirements to maximize understanding is a separate challenge that has not been solved yet [25, 65].

2.5 Graph Drawing Algorithms

The general goal of drawing diagrams is to convey relational information in a way that makes it more readable, understandable, and usable [86]. Most often, this goal is achieved by considering the graph drawing requirements during the layout progress [65].

Automatic drawings of biological networks were initially generated by common graph drawing algorithms. However, many of these algorithms are designed for simple, idealized graphs, and do not achieve similarly useful results for biological networks [52]. These approaches include hierarchical approaches, the topology-shape-metrics approach, algebraic approaches, force-directed, and constraint-based approaches [71, 91, 98]. Hierarchical and force-directed approaches are among the most popular ones [71, 98]. The automatic drawing process then evolved by extending these approaches to consider the specific requirements of biological networks [71].

In the following sections we review the most relevant approaches and introduce some of their domain-specific extensions. For a more detailed overview of general graph drawing techniques, we refer to the surveys of, e.g., Gibson et al. [65] and Tarawaneh et al. [98].



(a) (b)

Figure 2.4: Different drawing conventions of hierarchical graph structures. (a) shows the tip-over convention and (b) displays the inclusion convention.

2.5.1 Hierarchical Layout

The hierarchical drawing is also known in the literature as level or layered, as monotone drawing, or as *tip-over convention*. It is characterized by edges pointing in a common direction and nodes being separated into different layers based on predefined criteria [22, 71]. An alternative way to visualize a hierarchical data structure is by using the *inclusion convention*, where a parent-child relationship is mapped to a container holding another container [22]. The difference between the tip-over and the inclusion conventions is illustrated in Figure 2.4.

A fundamental approach to create hierarchical drawings is the Sugiyama algorithm, also known as the Sugiyama method or Sugiyama framework [96]. Since the diagram is solely build upon the hierarchy encoded in edge directions or, alternatively, in node-attributes, no initial layout is required. The Sugiyama algorithm consists of tree steps:

1. Cycles are removed, since the algorithm is designed for DAGs. This is commonly done by reversing the direction of some edges.

2. A multilayered digraph is constructed. This is done by partitioning the nodes into layers in such a way that a node is only connected to another node in the next layer. The layers are vertically arranged, i.e., every node on the same layer has the same y-coordinate. If a node has multiple parents on different layers, dummy nodes are added to form long edges over multiple layers.

3. Edge crossings are minimized by reordering nodes on the same layer, i.e., changing a node's x-coordinate.

Hence, the y-coordinate emphasizes the graph hierarchy, while the x-coordinate is responsible for satisfying the graph drawing requirements. Various algorithms exist for each of these steps [96].

The Sugiyama algorithm results in a hierarchical drawing with non-overlapping nodes (R02), minimal edge crossings (R05), and minimal edge length (R09). It does not inherently consider node sizes, but can be extended to do so [90]. The Sugiyama algorithm can be successfully applied for graphs of up to 500 nodes.

Most of the hierarchical approaches derive or evolved from the Sugiyama algorithm [71]. Schreiber [90] adapted the Sugiyama algorithm to fit the requirements of biological networks. He mapped the networks to an ordered bipartite, directed graph, extended the algorithm to consider variable node sizes, and grouped disjoint subgraphs into dummy nodes. Each subgraph can be arranged separately with an arbitrary graph drawing algorithm. The flow direction (R15) of the graph is considered during the layout process by forcing nodes of the same hierarchical layer into vertical alignment, and enforcing a horizontal ordering. Since the algorithm is at its core still the Sugiyama algorithm, it shares its characteristics. Cycles have to be removed in a separate step and it reduces edge crossings (R05). Additionally, Schreiber's variation of the Sugiyama algorithm can handle disjoint subgraphs and draws diagrams of biological networks with up to 300 nodes in a layered fashion, thereby emphasizing the chronological order of its processes (R15) [90].

Barsky et al. [19] developed a layered layout, called Cell Region-Based Rendering And Layout (Cerebral), for biological networks. It aims to visualize the nodes' subcellular locations, resulting in a cross-sectional view of a cell [19]. Thereby, domain-specific information is integrated into the layout, which furthers the possibility of gaining insights about the biological network. All nodes are placed on a grid, starting with a random placement, which is then improved via a stochastic sampling and a scoring function, while keeping the nodes in their designated subcellular layer. Edge-bundling and interaction techniques, like panning and zooming, intelligent labeling at multiple zoom levels, and highlighting, make their approach feasible for networks with a few hundred nodes. Pathway products, which share molecular functions within a subcellular location, are grouped visually by enclosing them in their convex hull. The approach is accessible as open source plugin for Cytoscape [58].

The discussed algorithms generate drawings in the tip-over convention. They do not consider nodes as containers of other nodes and can, therefore, not map the SBGN process description, which uses diagrams in the inclusion convention.

2.5.2 Topology-Shape-Metrics Layout

The common use-case for Topology-Shape-Metrics (TSM) approaches is the drawing of undirected graphs as orthogonal grid layouts, therefore satisfying uniform node distribution (R03) and orthogonality of edges (R08). The TSM approach consists of three steps:

1. The *planar step*, calculates a planar embedding of the graph. If a planar embedding is not possible, the graph can be temporarily made planar by inserting dummy nodes.

2. The *orthogonalization step* changes the shape of the edges without changing the planar embedding. Every edge is replaced by an orthogonal path consisting of vertical and horizontal lines. In this step, the number of edge bends can be minimized (R06).

3. Finally, the *compaction step* calculates the positions of all nodes and bends by placing them on a unit grid without producing overlaps. In this step, the graph drawing's area can be minimized (R12).

Each of these steps can be solved by many heuristics and approaches.

Siebenhaller et al. [92] demonstrated the use of the TSM approach to draw biological diagrams in the SBGN. They extended the TSM approach to support hierarchical clustering and the modeling of compartments (R14). However, disjoint subgraphs are not mentioned, the results only show graphs with maximum node degree of 4 and neither demonstrate nor discuss the impact of higher degree nodes on the orthogonality of the layout. The results also only show comparatively small diagrams, and the SBGN requirements are only partially satisfied, e.g., hyperedges as well as the correct connectivity and visualization of process glyphs are not supported [92]. The approach of Siebenhaller et al. [92] is available in the yEd Graph Editor [12].

2.5.3 Force-Directed Layout

The basic idea of force-directed algorithms is to model a physical system of attraction and repulsion to optimally place the graph's nodes [71]. An optimal placement is most often defined by graph drawing requirements [65, 86]. Force-directed algorithms developed into two main approaches, spring-embedded ones and energy-based ones [71].

The spring-embedded approach interprets nodes as steel rings and edges as springs connecting them. This spring system is then set into a random initial state and released [71]. When the spring-embedded approach approximates an equilibrium in this spring system, i.e., the forces influencing each node sum up to zero, then optimal placement is achieved [65, 71].

Dogrusoz et al. [43, 44] used a spring-embedded approach to design a compound layout for biological networks that visualizes the correct placement of processes and biological entities in their designated compounds. The authors introduce the relativity force that aligns processes into a certain flow direction. In the first step, a skeleton graph (without trees) is placed with a basic spring-embedding, but without relativity and gravitational forces. Then, the previously removed trees are re-added, level-by-level, while considering relativity and gravitational forces. In the last step, node positions and compartment bounds are optimized. Inter-graph edges are elongated proportionally to

their nesting depth. This approach addresses the following requirements: clustering and containment (R14) as well as flow direction (R15). It is implemented in the Pathway Analysis Tool for Integration and Knowledge Acquisition (PATIKA) editor [40] and demonstrated on small biological networks up to 50 nodes without SBGN. This approach is also versatile as it can be used in other domain applications, e.g., on brain network data as demonstrated by the work of Wißmann [83].

Energy-based approaches consider the layout problem as an optimization problem, where an *energy function* $F(\mathbf{X})$ (also known as cost or objective function) over the positions of nodes $\mathbf{X}$ is minimized [65, 91]. The energy function encodes the desired graph drawing requirements and measures the quality of the layout [91]. Its minimization leads to an optimal drawing [61]. In contrast to spring-embedded approaches, energy-based approaches need an initial layout to start their minimization process.

One of the fundamental energy-based approaches has been designed by Kamada et al. [68, 91] and later improved by Gansner et al. [61]. They interpret a graph as a spring system where nodes are connected by springs of a desirable target distance, i.e., the graph theoretical distance. A *balance* of the spring system can be reached, when all pairwise Euclidian distances $\|\mathbf{x}_i - \mathbf{x}_j\|$ of nodes i and j approach the corresponding target distances d_{ij} of nodes i and j [49, 61, 68]. This is formulated in an energy function, which defines the *imbalance* of the spring system, as the square summation of the real distances between all pairs of nodes minus their desirable target distance. Minimizing the imbalance of the spring system results in an optimal layout considering the graph drawing requirements of uniform edge length (R07), and symmetry (R19) [49, 61, 68]. This imbalance function, is more popularly known as the *stress function* in the Multi Dimensional Scaling (MDS) community [49, 61, 91]. The stress function is described in the following equation:

$$stress(\mathbf{X}) = \sum_{i<j} w_{ij}(\|\mathbf{x}_i - \mathbf{x}_j\| - d_{ij})^2, \tag{2.1}$$

where the matrix $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ contains the positions of the nodes. The normalization constant is defined as $w_{ij} = d_{ij}^{-\alpha}$. The value of α varies in the literature, it ranges from 0 to 2 [49]. d_{ij} is set to the graph theoretical distance [49], or as suggested by Cohen [36] to the *linear-network* distance, which reflects a graph's distance as well as its structure. Kamada et al. [68] place all nodes on a circle as initial layout, which can lead to edge crossings in the result layout. Gansner et al. [61] recommend instead to use multiscale techniques or subspace-restricted computation for the initial layout, both techniques also improve the speed of the subsequent stress computation. It is noted at this point that different initial layouts lead to structurally identical graphs with different layouts, i.e., *isomorphic graphs*. The minimum of the stress function can be calculated by a two-dimensional Newton-Raphson method, where in each iteration only one node is moved [68], or by gradient descent [49]. However, both methods could lead to local minima, i.e., while the resulting layouts are locally optimal, globally there is still room for improvement [91]. Gansner et al. [61] solved this issue by minimizing the stress function using stress majorization, which is an approach from the field of MDS. In contrast to

Newton-Raphson or gradient descent, it is a global approach and guarantees a monotonic decrease of the energy value, improved robustness against local minima, and shorter run times. Therefore, Gansner et al. [61] improves Kamada et al. [68] in terms of layout quality, monotonicity of convergence, and run time. Using the stress function is simple, intuitive, and works for connected, undirected, symmetric, asymmetric, or weighted graphs with straight-line edges. It shares strong similarities with MDS [61, 68].

Force-directed algorithms in general are intuitive and easy to understand because of their real-world metaphor. They are also easy to implement and extend with additional constraints [71]. They preserve proximity (R09), do a clear decomposition of clusters (R14), and display a graph's symmetry (R19) [49]. Basic force-directed approaches are for graphs of moderate size with fewer than 100 nodes and underlying graph structures of grids, trees, or sparse graphs [65, 98]. They generate an organic appearance, but that is not necessarily the desired outcome or application-specific goal [49]. The minimization of edge crossings is not taken into account, which can lead to confusing results for highly connected graphs. Preserving the mental map is not supported, since a rerun of the algorithm would lead to a different result [49]. They are considered expensive, in terms of run time with $O(n^3)$, where n is the number of nodes. For dense graphs, they perform poorly as they tend to generate *hair balls* [65, 98]. Force-directed algorithms are among the most frequently used and modified layout algorithms [65]. They were revised and optimized to improve the listed drawbacks many times [98]. Meanwhile, they can handle thousands of nodes and the results are suitable for various applications [65, 71]. A general overview and comparison of force-directed algorithms is provided by Gibson et al. [65].

2.5.4 Constraint-Based Layout

Constraint-based techniques extend force-directed approaches by incorporating additional constraints into the layout process. These constraints are usually derived from graph drawing requirements, recall Section 2.4, some user-defined placement criteria, or from an application domain's node attributes [65, 91]. Because biological data already have notations with associated placement rules that standard approaches cannot satisfy, the biological community has strongly advocated the development of constraint-based approaches [65]. The aim of these approaches is not only to support understanding of graph structures, but also to place them in a *biological meaningful* way [19]. Firstly, we discuss a number of general constrained-based approaches that were applied to biological networks, and then discuss approaches specifically tailored to them.

Dwyer et al. [49, 52, 53, 50, 48] developed multiple approaches, from general ones to approaches specifically targeting biological networks. They started with an approach named Constrained Layout of Digraphs (Dig-CoLa) [49], which was applied to gene-expression networks [49]. It is designed for general digraphs rather than biological networks and is based on the idea of the Sugiyama algorithm [96], but reformulated as force-directed approach with additional constraints to visualize the overall graph flow direction (R15). The algorithm requires prior knowledge of the data's relative hierarchy, which is encoded as node attribute and describes which node precedes another one. If

node i precedes node j, then i is drawn above j. This constraint is imposed onto the edges as weight. If a substructure does not have a clear hierarchy, e.g., a circle, the edge weight can also be zero and represent thereby undirected edges. Because constraints can be better integrated into the global stress majorization [61] than into the local one [68], the global version is used to solve the resulting constraint stress function [49]. Dig-CoLa differs from other approaches, which are based on the Sugiyama algorithm, by calculating the horizontal and vertical layout positions simultaneously. This is possible by using a force-directed instead of a hierarchical approach. Dig-CoLa emphasizes the hierarchical structure of a graph along a vertical or horizontal axis, while preserving the proximity relations of nodes (R09) and symmetries of substructures (R19). The algorithm can handle a substructure without a clear hierarchy, like a circle, with the same quality as an undirected layout approach would. Dig-CoLa is better suited to draw hierarchies than other force-directed approaches designed for undirected graphs and gives a good indication of the temporal flow (R15) of processes encoded in the graph. Dwyer et al. [49] state that Dig-CoLa is more suited for large digraphs than the Sugiyama algorithm, but is not optimized to reduce edge crossings. Dig-CoLa is implemented in the open source graph visualization software Graphviz [56].

Another approach by Dwyer et al. [52] is Incremental Procedure for Separation Constraint Layout of graphs (IPSep-CoLa). This time, stress majorization is extended with separation-constraints in each dimension, formulated as $u + d \leq v$, where u and v are horizontal and vertical positions, respectively, and d is the minimum separation to be achieved. With this approach, the following graph drawing requirements can be addressed: fixed position (R13), clustering and containment (R14), flow direction (R15), display of the structure (R20), and preservation of the mental map (R21). The stress function is solved iteratively by alternately minimizing quadratic functions in the horizontal and vertical direction. The minimization is done by a specialized gradient projection that performs at comparable speeds as unconstrained stress majorization [54]. By using diagonally scaled gradient projection in a follow-up paper, speed and convergence were improved [50]. IPSep-CoLa is demonstrated on gene-activation networks among others and can handle large graphs with thousands of nodes. Although the authors state that non-rectangular bounding boxes are possible, it is complicated and computationally expensive to implement. They also state that the constraints can become hard to satisfy for large graphs and, therefore, improvements are still necessary, e.g., in preprocessing or by using softer constraints [52].

Dwyer et al. [53] also extended their constraint stress majorization to integrate edge routing. They added an additional layout step and considered polyline edges with their separation constraint during the layout process. After the initial layout step, created with planarization-based methods, they added an additional edge routing step. This step performs edge routing, while considering node sizes, and reducing edge crossings. It adds polyline edges to the graph. Subsequently, the edge bending points are included in the following constraint layout step, considering the separation constraints of Dwyer et al. [48, 52] with the aim of straightening edge bends and reducing edge length. This approach

facilitates the integration of edge routing into a force-directed layout process and the support of polyline edges, while keeping crossings (R05) and bends minimal (R06) [53].

Schreiber et al. [91] together with Dwyer, demonstrated that separation constraints solved via gradient projection are applicable to biological networks. They showed that the separation constraints can be customized to emphasize pathways, draw cycles, and satisfy the drawing requirements of clustering and containment (R14) as well as flow direction (R15). Their approach gives a layout framework for different types of biological networks. It can also arrange parts of a large network differently and supports interactive exploration while preserving the mental map (R21) [91].

The literature suggests applying different approaches to different parts of a network rather than developing a general-purpose approach that meets all the requirements of biological networks [24]. The benefit lies in the possibility of using multiple automated layout techniques, which are specifically designed to handle certain substructures, or utilizing the human's intelligence to find an aesthetically pleasing layout [107]. The overall layout is the result of merging an initial layout with automatically- or user-defined layouts of substructures. This way, the layout can also meet domain-specific needs, which would otherwise not be possible with a holistic layout technique [107]. While Becker et al. [24] demonstrates this via a combination of customized spring-embedding approaches, Yuan et al. [107] formulates such a combined approach as an extension of the stress function called Laplacian Constrained Distance Embedding (LCDE) and solve it via a stress majorization technique. The predefined layouts of the subgraphs are used as constraints for the stress function. Opposed to Dig-CoLa and IPSep-CoLa, they do not aim for a hard constraint satisfaction, but for a softer way that allows each subgraph to be preserved as much as possible (R21). LCDE does this well through distance preservation even if more than one constraint influences a subgraph. It was demonstrated, among others, on a biochemical metabolic pathway, by merging multiple pathways with user-defined layouts into a single one.

2.5.5 Comparison and Discussion

The graph drawing approaches reviewed so far have different capabilities. In this section, we summarize and compare them as well as discuss their suitability for drawing biological networks in the SBGN.

Most hierarchical drawing algorithms are designed to draw diagrams in tip-over convention. This makes them unsuitable to visualize subcellular locations in SBGN, since such locations are displayed in the inclusion convention. While Barsky et al. [19] is able to visualize subcellular locations as cross-sectional, i.e., layered, views, their approach cannot visualize a subcellular location being contained in another one. Hierarchical algorithms could be used to highlight the temporal flow of SBGN process diagrams by aligning their execution trajectory with a predefined flow direction.

TSM approaches can be used to create orthogonal grid layouts. However, such layouts do not satisfy domain-specific requirements of the SBGN. The extension of Sieben-

haller et al. [92] successfully integrates compartments and, therefore, subcellular locations. However, the authors do not demonstrate scalability of their method to large and complex data sets.

Force-directed approaches are able to handle graphs with many thousands of nodes. If they include constraints, these approaches are referred to as constraint-based. The constraints usually model graph drawing or domain-specific requirements. Formulating and combining the needed constraints to completely satisfy the needs of biological networks is an ongoing challenge. Dwyer et al. [49, 52, 53] and Schreiber et al. [91] proposed several approaches, none of them was demonstrated in the SBGN, but each deal with different requirements of biological networks. Combining several approaches to meet all requirements in a single layout is usually not possible. The reason for this is that

Table 2.1: Graph drawing approaches, categorized into hierarchical, TSM, force-directed, and constraint-based ones. For each approach, we indicate whether it has been demonstrated on biological networks, can visualize subcellular locations, can handle hypergraphs, whether it supports disconnected components, and whether the SBGN has been used.

Approach	Hierarchical	Topology-Shape-Metrics	Force-Directed	Constraint-Based	Biological Networks	Subcellular Location	Hypergraph	Disconnected Components	SBGN
Sugiyama et.al [96]	✔								
Schreiber [90]	✔				✔			✔	
Barsky et al. [19]	✔				✔	✔			
Siebenhaller et al. [92]		✔			✔	✔			(✔)
Dogrusoz et al [43]			✔		✔	✔			
Kamada et al. [68]			✔						
Gansner et al. [61]			✔						
Dwyer et al. [49]				✔	✔				
Dwyer et al. [52]				✔	✔	✔			
Dwyer et al. [53]				✔					
Schreiber et al. [91]				✔	✔	✔			
Becker et al. [24]				✔	✔				
Yuan et al. [107]				✔	✔				

these approaches are often based on different theoretical foundations. Becker et al. [24] as well as Yuan et al. [107] attempted to combine the advantages of different approaches by applying them to subgraphs and combining their results into a single layout.

Table 2.1 shows an overview of the graph drawing algorithms we discussed, including their categorization and capabilities. In general, these approaches do not consider hyperedges and only Schreiber et al. [90] mention disjoint subgraphs in biological networks. While some approaches were demonstrated on biological networks, only Siebenhaller et al. [92] use data in the SBGN.

2.5.6 Quality Metrics

The literature defines a number of quality metrics to evaluate the quality of node-link diagrams, also called aesthetic or quality measures. Metrics that are commonly used include the number of edge crossings, the number of edge bends, and the symmetry [81, 86, 87]. There are also more specialized quality metrics like the clustering metric, which evaluates the visual clustering quality of a clustered graph [81], or the so-called *sprawlter metric* [79], which measures the space efficiency of a layout. Graph drawing approaches based on the stress function additionally evaluate the stress value and the number of iterations [61, 103]. Those metrics correlate with a user's ability to perform tasks on a graph and focus on the evaluation of syntactic graph drawing requirements. However, none of the above-mentioned metrics evaluate domain-specific characteristics of a graph drawing, e.g., those specific for the SBGN.

2.6 Visualization of Compound Structures

In addition to a network's connectivity, its compound structure and cluster information should be integrated into a network's layout or visualized in its drawing [37]. In metabolic networks, this information describes the parts of a cell where a reaction occurs. This is done in the SBGN by drawing compartments as bounding boxes [84]. Other common approaches that visualize cluster information, add visual attributes to identify cluster membership such as texture, color, symbols, or bounding outlines after all cluster members are moved into close proximity. If a close proximity is not possible, cluster discontinuities emerge [37]. In this section, we review advanced techniques to visualize cluster relations as overlays on a graph layout.

Gansner et al. [62] introduced an approach to represent cluster relations in a graph as a geographic-like, natural-looking map, called *GMap*. Since GMap requires a graph layout that emphasizes the underlying cluster information of the graph, their approach starts with creating a layout. Firstly, an initial embedding is calculated, e.g., using MDS, followed by a cluster analysis, e.g., with k-means. After an optional node overlap removal step, the initial embedding and clustering are subsequently used to generate the map. This is achieved by adding random points to the original embedding before creating a Voronoi diagram from the nodes' positions and random points on the bounding boxes

of their visual representation. The random points are added to create non-uniform but map-like Voronoi cells. The size of the bounding boxes is scaled by an application-specific weight derived from the data set. Voronoi cells belonging to the same cluster are then merged and assigned the same color. This results in a map with natural-looking outer and inner boundaries, and cluster sizes representing their importance. Adjacent clusters are colored with maximal color distance, which is achieved by solving a vertex labeling problem. GMap only supports disjoint clusters, but a cluster does not necessarily have to be represented as a contiguous area. This map-like approach visualizes structural information of the underlying data, such as clusters and neighborhoods. It is aesthetically pleasing and encourages users to interact with the map more than traditional graph drawings [62].

Another approach called *Bubble Set* overlays continuous bounding contours on existing graphs [37]. In contrast to GMap it does not require an explicit layout step and thus can be applied to any graph. It also creates contours that look hand-drawn instead of bounding areas. These continuous contours are created at interactive speeds using implicit surfaces, followed by marching squares. These contours include all cluster members, but exclude and navigate around non-cluster members. This is done by connecting all cluster members with virtual edges to the cluster's center of mass. The authors explicitly decided against using real edges to connect members, because a node-link relation does not implicitly define a cluster relation. If the virtual edges would intersect other clusters, edge routing steers these edges around the clusters. This results in blobby looking, aesthetically-pleasing shapes and reduces cognitive load of keeping track of spatially dispersed clusters. Bubble set handles cluster overlaps by adding transparency.

The *Line Set* approach draws a continuous curve with a distinct color through all members of a cluster. This curve should be smooth without self-crossings and minimize bends. The shape of the curve depends on the order in which the members of the clusters are visited. Finding a proper visiting sequence can be accomplished by solving the traveling salesman problem. Line set is stated to improve readability of cluster membership, because it introduces less clutter than traditional bubble techniques [15].

Jianu et al. [67] compared and evaluated GMap, Bubble Set, and Line Set through user studies in terms of their limitations and their effectiveness in conveying cluster information to a reader. These three approaches can be used on graphs containing spatially dispersed clusters. In GMap these dispersed clusters result in non-contiguous cluster areas, called islands, their cluster relation is only apparent through color association. In contrast, Bubble and Line Set handle these spatially dispersed clusters by connecting them visually. Bubble and Line Set achieved the best results in terms of accuracy at tasks to identify cluster memberships. For general network tasks like path tracing, GMap achieved the best results. The visual overlay with cluster information introduced a 25% accuracy penalty for all approaches, possibly caused by their induced visual clutter.

While these approaches are able to visualize cluster memberships, even of spatially dispersed clusters, none of them were designed for biological networks. Also none of

these approaches discuss clusters contained in other clusters or consider edges as part of clusters.

A cluster visualization technique that is demonstrated on biological pathways is described by Wu et al. [105]. It embeds a clustered graph into an area-balanced partitioning of the screen space through a top-down hierarchical Voronoi tessellation. While this approach can handle disjoint subgraphs, the visualization of multiple hierarchy layers of the clustered graph is considered future work. In summary, an approach that is compatible with the requirements of the SBGN has yet to be developed.

2.7 Interaction and Complexity Reduction

Nowadays, a common problem in visualization is the huge amount of data. Real-life networks are often too complex and too large to be understood or even displayed as a whole [13, 45]. One way to visualize such networks, which can consist of thousands of nodes, is to reduce their complexity using clustering, nesting, and interaction techniques.

Dogrusoz et al. [45] propose a method to reduce network complexity through an expand and collapse technique. Their underlying data model is a compound graph $C = (V, E, F)$, where V includes next to normal nodes also compound nodes, E is the set of adjacent edges, and F is the set of inclusion edges. Inclusion edges describe in which compound node a node is nested. The proposed method expands and collapses these compounds and adapts the existing layout through a specialized incremental force-directed layout method together with a heuristic for the placement of reappearing nodes in the expand operation. This keeps layout changes minimal during expand and collapse operations, and, therefore, preserves the mental map. The authors demonstrate their method on a biological network in SBGN by expanding and collapsing compartments. The method is available as extension for Cytoscape [58]. There are still many open issues related to such expand and collapse techniques, e.g., preserving the mental map, ensuring smooth transitions, and efficiency [13, 45].

2.8 Databases, File Formats, and Software Support

Since a wealth of biological information is collected in databases, it is important to know how to access, store, and analyze these data [17, 106]. Bader et al. [17] provide *pathguide*, a list of databases for biological pathway data. Among others, BioPath [31], Reactome [57], and KEGG [69] are listed. For another overview, we refer to Wu et al. [106]. An increasing number of pathways are drawn according to the SBGN and then stored and made available in these online databases.

There are different data formats used by the life science community. All these data formats aim to define standards, share information, and promote cooperation [66]. The Systems Biology Markup Language (SBML) is primarily used for simulation and analysis tools, because it allows users to include mathematical models of reactions and to represent

networks through different equations [66]. The basic version of SBML does not support storing the layout and visual representation of a network. However, it can be extended by a layout and render package [26, 64]. A C++ implementation of an SBML parser is provided by the software library LibSBML [30].

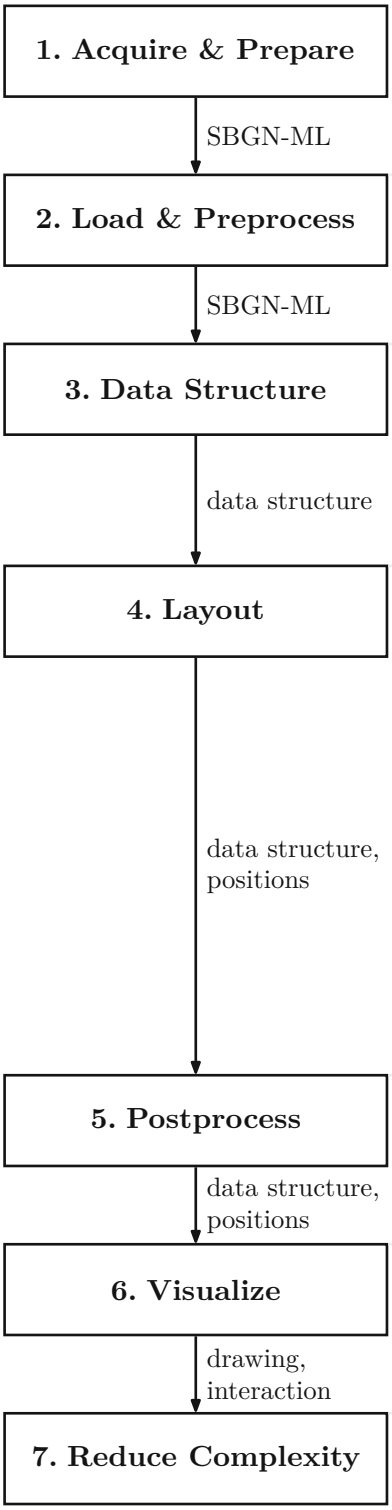
Since SBML does not faithfully store biological networks drawn in the SBGN, another file format called Systems Biology Graph Notation - Markup Language (SBGN-ML) was introduced. As the SBGN defines how the diagrams are visualized, SBGN-ML defines how these diagrams are stored. SBGN-ML stores the structure of a network and the SBGN glyphs as annotated nodes and arcs. The layout of a network is also stored by SBGN-ML for the following reasons: (a) good layouts should be preserved, (b) recomputing a layout is costly and time-consuming, and (c) manually-curated layouts usually communicate a specific message. To read, write, and modify SBGN-ML files, the software library LibSBGN was developed [101]. The library also supports the validation of diagrams according to the SBGN's definition. SBGN-ML and LibSBGN do not describe the appearance of glyphs. The goal of this file format is to increase the interoperability between SBGN-compatible software. For this purpose LibSBGN was written in several languages, e.g., in C++ and Java.

There are more than 170 specialized software tools for modelling, analyzing, and visualizing biological pathways [14, 71, 77, 101], e.g., PATIKA [40], Cytoscape [58], CellDesigner [59], and Vanted [88]. For a detailed overview please consult the work of Wang et al. [102]. There are also software solutions for network visualization of general graphs, but they usually do not support the established drawing conventions of biological networks [14], e.g., Graphviz [56].

CHAPTER 3

Methodology

In this chapter we define an approach to generate drawings of metabolic networks that satisfy the syntactic and semantic requirements (Section 2.4) of the SBGN. The created drawings need to faithfully represent the data, while supporting comprehension and readability. For this purpose we design a holistic pipeline **(C3)** that is divided into several steps, from data loading to complexity reduction. Each step addresses a set of requirements through selected methods. We discuss each step in detail, review alternatives, and provide examples. In the following we present our pipeline.



Firstly, we acquire metabolic pathway data and prepare them for our usage. We specifically obtain SBGN maps, saved in the SBGN-ML file format, from the Reactome pathway knowledgebase [57].

Afterwards, we load and preprocess the data to recalculate or remove unnecessary information, e.g., entity sizes and arc bends.

In the next step, we parse the loaded SBGN-ML file into a data structure. Since we plan to give special attention to the visualizations of subcellular locations, the data structure should allow us to access the nesting information of compartments. For this purpose, we chose a clustered graph structure.

Based on the literature, we chose a constraint-based layout algorithm, since they can be adjusted to model the requirements of SBGN. The layout consists of the following sub-steps:

(4.1) initial embedding,

(4.2) initial layout, and

(4.3) constraint layout.

For the first sub-step we chose the pivot MDS approach [32], for sub-steps 2 and 3, we chose the versatile, extendable, scalable, and constrainable vectorized form of the stress function introduced by Wang et al. [103]. It combines an unconstrained and constrained layout algorithm into one uniform framework.

In this refinement step, we possibly reroute edges to reduce remaining edge-edge or node-edge crossings, e.g., by introducing edge bends using an edge routing algorithm.

We visualize the now positioned nodes and edges according to the SBGN and visualize compartments in a motif like way.

Finally, we provide an adaptive version of expand and collapse interaction for clustered graphs to reduce the complexity of the graph and handle large data sets.

Table 3.1 gives an overview of our approach's capabilities, the detailed workflow, and the applied techniques. It recaps the requirements of graph drawings as well as the requirements of the SBGN and relates them to our approach. Requirements (rows) can be satisfied in one step or across multiple steps (columns) in our pipeline. The steps that are involved to satisfy a requirement are depicted by check marks. Black filled check marks indicate the steps we currently implemented in our prototype, whereas white filled check marks are considered future work. Unsupported requirements, i.e., the limitations of our approach, are crossed out. To convey this information in the text of this thesis as well, we highlight the implemented requirements in bold, e.g., **(R02)**, the requirements we defer to future work in regular font, e.g., (R11), and the limitations by crossing them out, e.g., ~~(R03)~~.

The table shows that most requirements need several consecutive steps to be fulfilled. For example, it takes preparation, the proper data structure, constraints, and a visualization technique, to satisfy the clustering requirement **(R14)**. Not all requirements can be handled independently, e.g., node sizes **(R01)** need to be determined first, to subsequently remove possible overlaps **(R02)** using a constraint. These dependencies cause a specific order in which the requirements need to be processed. Limitations are caused by the choice of the underlying techniques, e.g., a uniform node distribution ~~(R03)~~ is not an inherent result of the chosen layout technique. The table clarifies the overall goal of our entire process, as each step serves to faithfully represent the nested structure of the data **(R20)**.

In the following sections we elaborate on all steps of our approach and how they contribute to satisfy the requirements of graph drawing. In each step, we discuss the necessary implementations to fulfill the relevant requirements, mention possible implementation alternatives, and their limitations in creating an automated SBGN graph drawing.

3.1 Acquire & Prepare Metabolic Pathways

There are many publicly available biological databases for metabolic pathways, e.g., Reactome [5, 57], BioModels [1], UniProt [10]. We chose to acquire real-life data sets from the pathway knowledge-base *Reactome*. It provides manually-curated open source and open-data resources of human pathways and reactions [38, 57]. All data sets provide an existing layout that was hand-made by domain experts. Reactome supports data exchange with a variety of data formats like SBML [94] and SBGN-ML [77] and additionally provides the data sets in other common formats, such as JPEG, PNG, and PDF. Through Reactome's online visual pathway browser, different data sets can be selected by visual exploration, as shown in Figure 3.1. The data sets include biological pathways from different species, e.g., homo sapiens, gallus gallus and plasmodium falciparum. We selected different data sets with varying numbers of nodes and edges to demonstrate the possibilities of our method. We acquire the selected pathways' SBGN-ML files as well as their image representation, as a layout reference, and the descriptions of their function.

Table 3.1: The syntactic and semantic graph requirements and their relation to the individual steps of our approach. Steps involved in fulfilling a requirement are denoted by check marks. Black filled check marks indicate the steps we currently implemented in our prototype, whereas white filled check marks are considered future work. Limitations are crossed out.

Requirements	1. Aquire& prepare [57]	2. Load & Preprocess	3. Data Structure [18, 80]	4.1 Initial embedding [32]	4.2 Initial layout [103]	4.3 Constraints [103]	5. Postprocess	6. Visualize	7. Reduce Complexity
SYNTACTIC									
R01 Node: Size	☑	☑							
R02 Node: Non-overlap					☑	☑			
R03 ~~Node: Uniform distribution~~									
R04 Node: Orthogonality		☑							
R05 Link: Crossings				☑	☑	☑	☑		
R06 Link: Bends		☑					☑		
R07 Link: Uniform length					☑	☑			
R08 Link: Maximize orthogonality						☑			
R09 Node-Link-Connection: Minimize edge length					☑				
R10 Node-Link-Connection: Non-overlap					☑		☑		
R11 Diagram: Aspect ratio				☑					
R12 Diagram: Minimize area									
SEMANTIC									
R13 ~~Node: Fixed Position~~									
R14 Node: Clustering & containment	☑		☑			☑		☑	☑
R15 Link: Flow direction						☑	☑		
R16 Node-Link-Connection: Maximize minimum angles						☑			
R17 Node-Link-Connection: Connection points						☑			
R18 Diagram: Similarity	☑		☑			☑			
R19 Diagram: Symmetry					☑	☑			
R20 Diagram: Display structure of the data			☑	☑	☑	☑	☑	☑	☑
R21 Diagram: Preserving the mental map				☑		☑		☑	☑
R22 Diagram: Node degree reduction through cloning	☑								
R23 Diagram: Expand & collapse			☑						☑

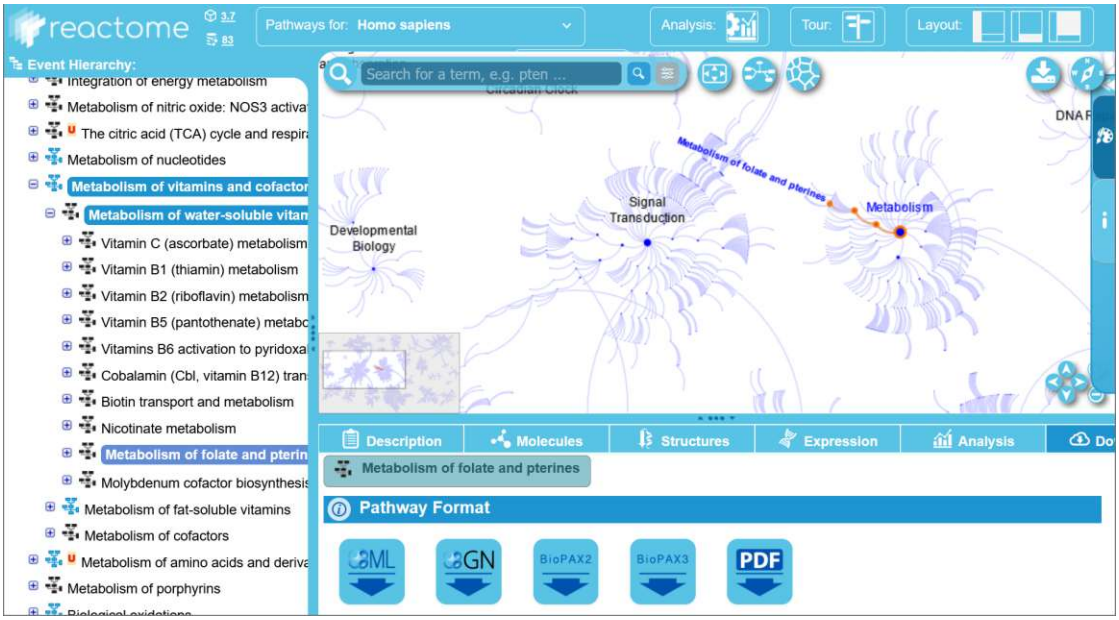


Figure 3.1: The Reactome pathway browser [5, 57].

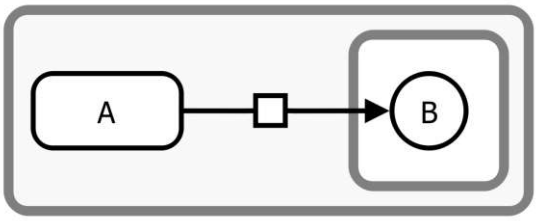


Figure 3.2: Example SBGN drawing to demonstrate its SBGN-ML structure in Listing 3.1.

An example SBGN-ML file representing the SBGN drawing from Figure 3.2 is given in Listing 3.1. An SBGN-ML file holds an SBGN map that contains different glyph and arc entities. These glyphs and arcs contain an identifier and the attribute *class*, which is their SBGN entity type. Glyphs contain a label, a position, its size **(R01)**, as well as its parent compartment as the attribute *compartmentRef* **(R14)**. The compartment reference holds the identifier of a compartment as value. It can also be empty if the node is not included in a compartment, or it can be entirely missing, because it is an optional attribute. The class *process* is a special glyph with two additional ports that have a position. We also refer to these ports as *handles*. An arc has a designated start and end, but could also consist of a sequence of points, i.e., a path. Arcs are not explicitly part of a compartment and, therefore, do not contain the attribute *compartmentRef*. All entities are connected via *id*s and the arc's attributes *source* and *target*.

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<sbgn xmlns="http://sbgn.org/libsbgn/0.2">
  <map language="process description">
    <glyph id="nwtN_1feb78a0" class="compartment">
      <label text="parent"/>
      <bbox x="281.68" y="280.15" w="223.19" h="94.5"/>
    </glyph>
    <glyph id="nwtN_c90b149a" class="macromolecule" compartmentRef="
        nwtN_1feb78a0">
      <label text="A"/>
      <bbox x="297.31" y="312.38" w="60" h="30"/>
    </glyph>
    <glyph id="nwtN_33aed61d" class="compartment" compartmentRef="
        nwtN_1feb78a0">
      <label text="child"/>
      <bbox x="427.00" y="296.77" w="61.25" h="61.25"/>
    </glyph>
    <glyph id="nwtN_dfc66e7a" class="simple chemical" compartmentRef="
        nwtN_33aed61d">
      <label text="B"/>
      <bbox x="442.63" y="312.40" w="30" h="30"/>
    </glyph>
    <glyph id="nwtN_9aec2f54" class="process" compartmentRef="nwtN_1feb78a0">
      <bbox x="386.40" y="320.30" w="14.28" h="14.28"/>
      <port id="nwtN_9aec2f54.1" x="383.54" y="327.44"/>
      <port id="nwtN_9aec2f54.2" x="403.54" y="327.44"/>
    </glyph>
    <arc id="nwtE_faa8664d" class="consumption" source="nwtN_c90b149a" target
        ="nwtN_9aec2f54.1">
      <start x="357.93" y="327.41"/>
      <end x="382.91" y="327.44"/>
    </arc>
    <arc id="nwtE_70bb781f" class="production" source="nwtN_9aec2f54.2"
        target="nwtN_dfc66e7a">
      <start x="404.16" y="327.44"/>
      <end x="438.88" y="327.41"/>
    </arc>
  </map>
</sbgn>
```

Listing 3.1: The SBGN-ML data representing the SBGN drawing shown in Figure 3.2. Identifiers and decimals are shortend for better readability.

Unfortunately, SBGN-ML files acquired from Reactome do not provide compartment references for its nodes as this information is only an optional attribute of SBGN-ML. Because this information exists semantically in the layout made by domain experts, we can add it manually to the SBGN-ML. We wrote a small editing option for our tool that allows users to load an SBGN map from its SBGN-ML file, multi-select entities, and insert them via a context menu into their corresponding compartments, see Figure 3.3. Finally, the SBGN map with the additional compartment references can be saved as SBGN-ML file.

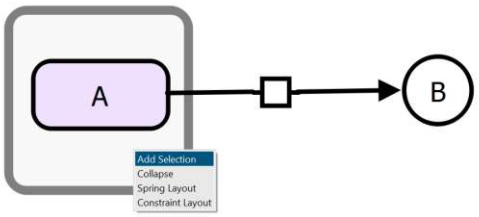


Figure 3.3: Our solution to add compartment information to entities.

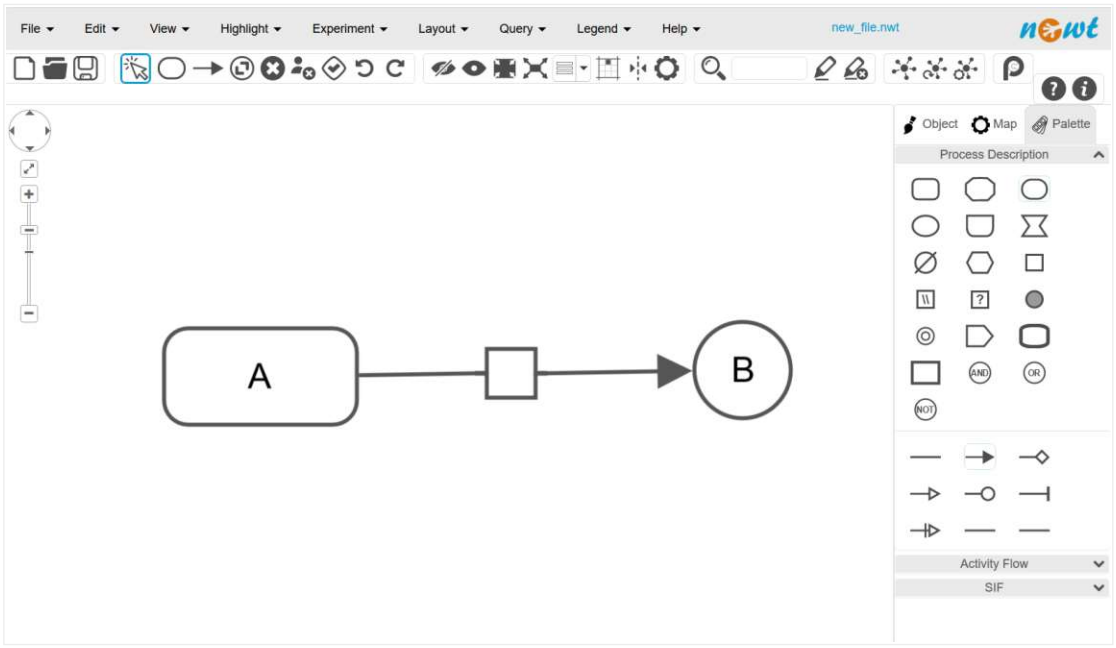
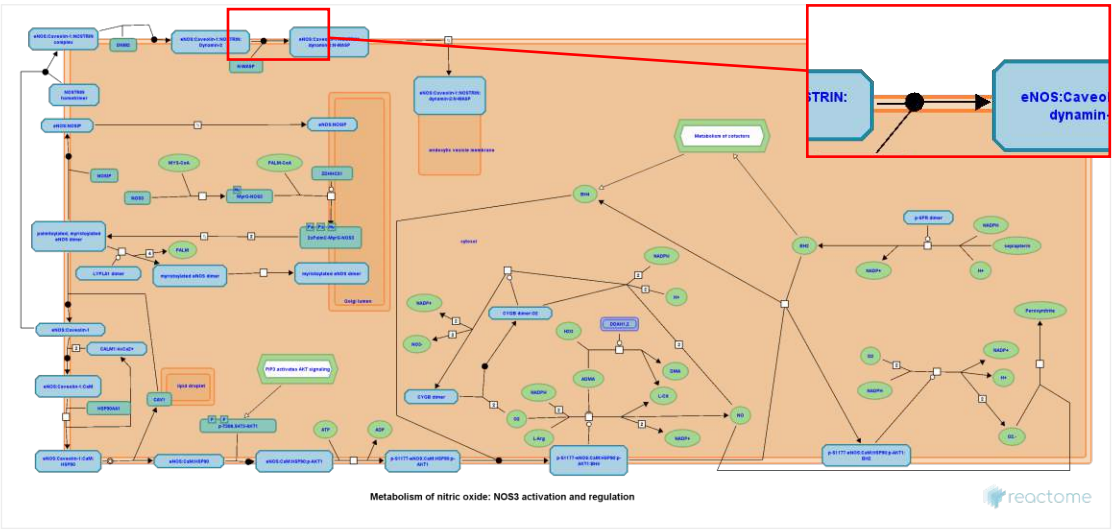


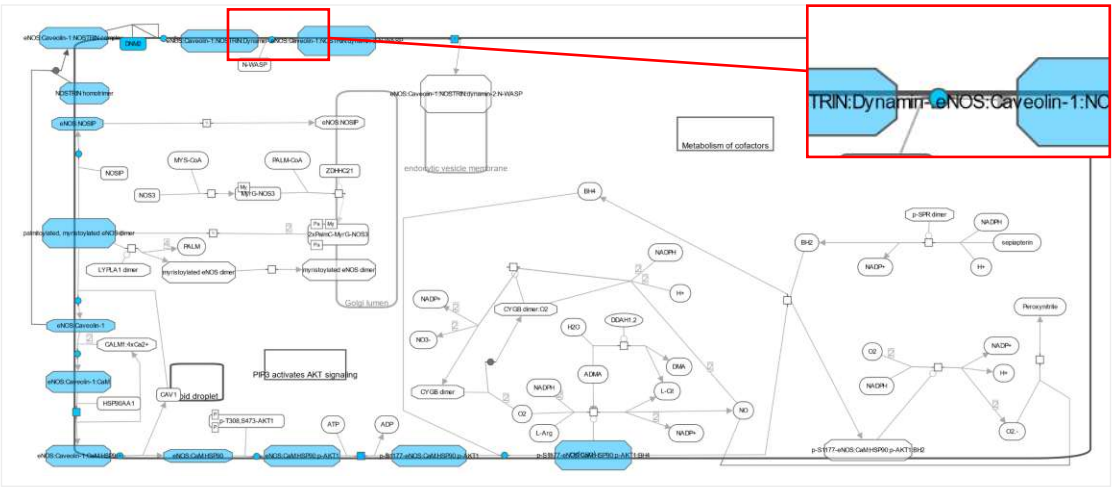
Figure 3.4: Newt online editor [4, 89].

Compartment references could also be added with existing tools, e.g., the *Newt online editor* [4, 89], shown in Figure 3.4. Newt is an SBGN editor, implemented as a web application that allows users to create and edit existing pathways, and export them in different formats, e.g., SBML as well as SBGN-ML. In Newt, one can load data sets and add entities to existing compartments as follows: Entities are first selected, then a targeting cursor is activated by clicking on the selection again while pressing *ctrl*, the cursor is then dragged over the target compartment, where it is released. Subsequent saving includes the newly added compartment references to the SBGN-ML file. Since this interaction is hidden and unintuitive, we preferred our own editing tool.

With this minimal preparation using visual editors, we were able to supplement the data from Reactome with the missing compartment information. The recovering of entity and compartment relationships could also be done automatically. The nodes'



(a)



(b)

Figure 3.5: Illustration of the community convention placing entities on double compartment boundaries. (a) shows a hand-curated SBGN map where the author placed entities on the compartment's double boundaries (the two orange lines in the zoom-in placed in the top-right corner) to convey that processes occur within the membrane. (b) shows the SBGN map loaded from the SBGN-ML file associated with the hand-curated map from (a). The blue nodes originally placed on the compartment's double boundary are now placed on the single boundary or within the compartment. The original compartment affiliation is lost, as shown in the detailed zoom-in for comparison. The reason for this is that the community convention cannot be represented in SBGN-ML.

positions and sizes as well as the compartments' positions and bounding boxes are known through the SBGN-ML, and a simple point-inside-bounds-test would suffice to compute the missing information. One obstacle to this procedure is a community convention that has evolved to represent a process occurring within a cell membrane in an SBGN map. In this convention two compartments are drawn in close proximity to create a double boundary, representing a cell membrane. Within this double boundary, processes and entities are placed, as shown in Figure 3.5a. This convention has currently no real mapping to an SBGN-ML description, as SBGN-ML only maps an inclusion relationship between one entity and one compartment. Hence, the double boundary is resolved as one compartment, this is shown in Figure 3.5. While this is a known limitation between the SBGN definition and its mapping to SBGN-ML or other file formats [84], it is not discussed in the literature **(C3)**. We exclude the community convention for simplicity and only choose metabolic pathways where this convention is not used or not necessary. From now on, we assume to have the compartment glyph relation to its nodes available as compartment references in the SBGN-ML files.

Another aspect of real-life data sets is the cloning of nodes **(R22)**. Choosing the nodes to be cloned depends on a domain-specific consensus by the life science community, e.g., a commonly cloned node is ATP. If this is performed, the number of high degree nodes and edge intersections can be reduced, and the overall readability of the graph is improved. We assume that all cloned nodes are already included in the data, hence, we do not have to take special care when loading the data.

Additionally to the real-life data sets from Reactome, we manually generated several phantom datasets with the Newt online editor. These phantoms were deliberately designed to depict various graph structures with different numbers of nodes, node sizes, different connectivities, and various compartment nestings. They are used to develop, test, verify, and identify limitations of different aspects in a controlled way.

To summarize, as first step of our approach we acquired real-life data of biological pathways in SBGN from the pathway knowledge database Reactome in the file format SBGN-ML. This data was edited to reconstruct the missing compartment entity relation via a simple visual editing tool. We also created phantom data sets using the Newt online editor.

3.2 Load & Preprocess of SBGN-ML Files

We used the C++ software library LibSBGN to load the SBGN-ML files [39]. After loading we preprocess the data.

Complex glyphs are a special kind of container that hold only glyphs but not arcs, shown in Figure 3.6. Arcs from other entities do not connect to entities contained inside the complex but to the complex itself. As Siebenhaller et al. [92] suggest, a complex is represented as one node in the graph, and we consider the contained nodes only during visualization **(R02)**.

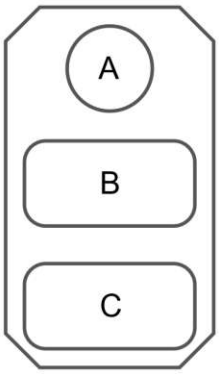


Figure 3.6: A special container entity, a complex glyph. Since there are no arcs connecting to the entities inside the complex, it can be represented as one node in a graph.

In an SBGN-ML file arcs are depicted as start and end points, but can also consist of a sequence of points, i.e., a path. Hence, an SBGN-ML file supports polyline edges. These polyline edges usually represent edge bends created in the loaded layout. Neither of the chosen layout algorithms support graphs with polyline edges, neither the pivot MDS for the initial embedding [32], nor the vectorized stress majorization for the constraint layout [103]. Since our approach generates a new layout, the existing polyline edges are ignored. We discard all paths loaded from the data and replace them with a straight edge from the start to the end points of the polyline edge. This removes all edge bends **(R06)** from the loaded data.

Entities contain their predefined sizes in the SBGN-ML file in pixels. For entities with labels, their sizes are not always defined to fully encapsulate the node labels. To ensure labels are included in the nodes, we chose to resize entities with a user-defined maximum width and a user-defined padding. Entities containing their labels may result in larger nodes, but the overall layout might be less cluttered. All entities have now a fixed size and contain their labels before starting the layout process **(R01)**.

Neither the node labels nor the nodes themselves contain rotation information in SBGN-ML. Hence, we consider them axis-aligned. Since no rotation is introduced later, nodes and labels stay orthogonal **(R03)**.

The original positions of nodes and edges are not required for our approach, so we could have discarded them too. Instead, we load them to visualize the data after the initial loading step for convenience. To summarize, we load the SBGN-ML via a software library, merge complexes and their contained entities to one node, remove edge bends, and resize nodes to contain their labels.

3.3 Data Structure

The data loaded from the SBGN-ML files need to be transformed into a data structure, specifically, a graph representation that incorporates all information required during the layout process [49, 90, 107]. This graph representation needs to support the multivariate nature of biological pathways [71], and to facilitate the creation and visualization of the layout.

Firstly, we recap the properties of the SBGN process notation to select a suitable data structure. The process notation describes processes which consume biological entities with a possible involvement of a catalyst or an inhibitor to become other biological entities, as demonstrated in Figure 3.7. This gives the data an inherent direction, which needs to be modelled in our data structure, making the graph a directed graph. Biological entities and processes are both mapped to vertices. Since biological entities are only allowed to connect to processes, the data translates to a bipartite graph. This also means the graph contains no self loops, which makes it a simple graph.

A biological network can consist of disconnected components and can contain cycles. This needs to be considered when using graph algorithms. One or multiple metabolites are connected to a handle of a process glyph. This translates to a one-to-many and many-to-one relationship in the graph representation, which can be mapped to hyperedges. Hence, a hypergraph is an accurate representation of the SBGN process notation [80], as shown in Figure 3.8.

Maniadi et al. [80] investigated the suitability of hypergraphs as representation of metabolic pathways. In their data model, every node represents a biological entity and every edge is a reaction. Therefore, process nodes do not exist and hyperedges represent many-to-many relations. To use conventional graph drawing techniques and visualize such hypergraphs, they transform them into common graphs. Hyperedges are simplified by inserting dummy nodes at the hyperedge's branching points. One dummy node q_u connects all source nodes u_i, while the other one q_v connects all target nodes v_j. This is only done when the hyperedge has multiple source or target nodes. If the hyperedge has

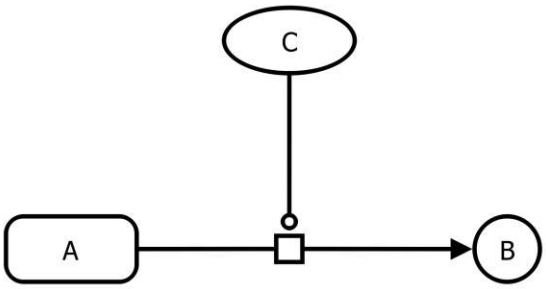


Figure 3.7: An example biological process in SBGN process notation. The macromolecule A becomes the simple chemical B under the influence of the catalyst C. The graph flows from A over the process node to B as well as from C over the process node to B.

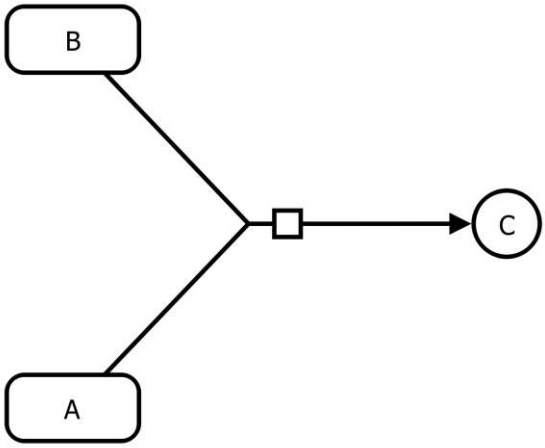


Figure 3.8: Example SBGN drawing showing a hyperedge. The macromolecules A and B are connected to the process node from the left through a hyperedge.

one source and one target node, it is treated as common edge and no dummy nodes are inserted. Consequently, at most two dummy nodes are inserted per hyperedge. With this approach, the problem of drawing a hypergraph (recall Definition 2.1.12), can be reduced to the problem of drawing a graph $G = (V, E)$, where $e = \{u_i, q_u\}$ are edges from the source to its dummy node, $e = \{q_v, v_i\}$ are edges from the other dummy node to the target node, and $e = \{q_u, q_v\}$ is a dummy edge if $q_u \neq q_v$. Conventional techniques can be used now at the only expense of an increased network size caused by the insertion of the dummy nodes [80]. The approach of Maniadi et al. [80] is used in other relevant works, e.g., Becker et al. [24].

We adapt the approach of Maniadi et al. [80] to the SBGN process notation. To recall, this notation requires to explicitly model process glyphs as (real) nodes in the graph. For each handle of every process glyph, we now insert a dummy node and make no exceptions for hyperedges with only one source or target node. As shown in Figure 3.9, consumed and produced entities are only connected to these dummy nodes (blue dots), while nodes connected through regulatory arcs are connected to the real node (red dot).

With the insertion of dummy nodes we introduce two different edge types: real edges and dummy edges. The dummy edges connect the handles' dummy nodes to the center node of the process glyph and have a shorter edge length than real edges. The edge type is important for the layout process and, therefore, is imposed upon the edges as attribute, making our data structure a multivariate network. Every entity node has a label that is fully enclosed in its node's boundary, resulting in different node sizes. The node sizes are added as node attributes to our data structure.

So far, our data structure is able to model a metabolite's transformation from one entity into another one, but we have not yet considered the location of the transformation within a cell. The location is displayed as compartment glyph in SBGN, and represented

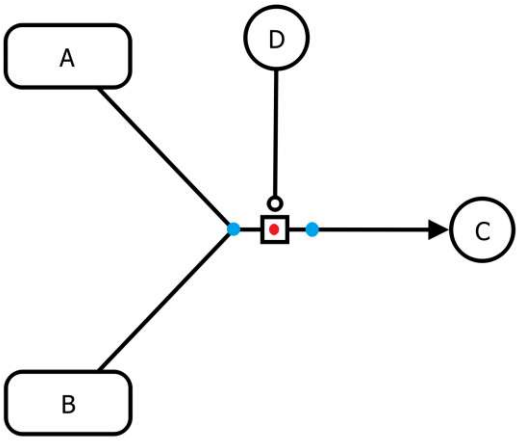


Figure 3.9: Mapping of a process glyph to dummy nodes. The two blue dots represent the inserted dummy nodes, while the red dot represents the real process node.

as compartments and compartment references in SBGN-ML. Compartments can hold entity nodes, process nodes including their dummy nodes, and other compartments. To represent this hierarchical structure, we use a clustered graph [71]. Alternatively, we could encode the relationship between a node and a compartment as a node attribute, where every node holds a list of compartment references. In such a case, selecting all nodes of a compartment would not be efficient, since every compartment list of every node would need to be queried.

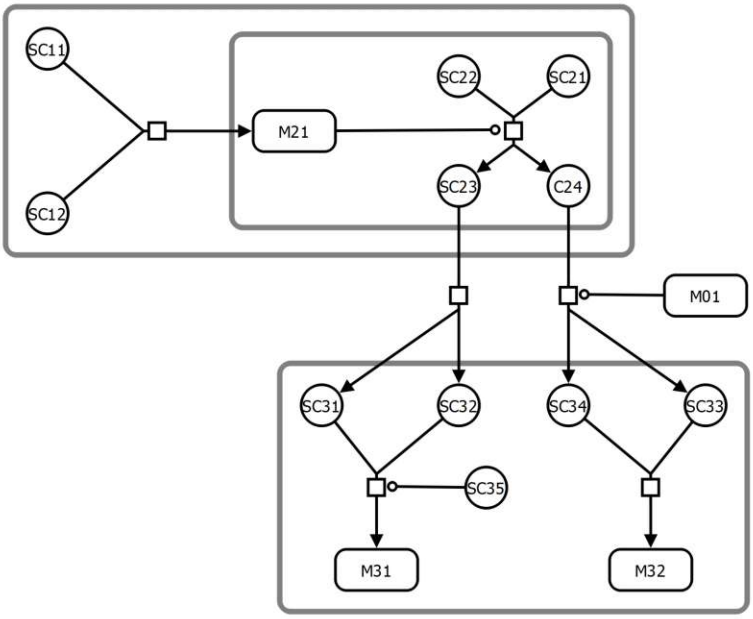
For that reason, we construct a clustered graph $C = (G, T)$, where G is the input graph without cluster information and T is a rooted tree, referred to as *cluster tree* and every node of T is a cluster c. From now on, we refer to nodes of T simply as clusters. The cluster tree T describes an inclusion relationship between clusters [13, 18], i.e., clusters can contain child clusters, recall the tip-over convention shown in Figure 2.4a. Every cluster relates to a subgraph G' of G, chosen according to a cluster information or grouping criterion. A child cluster relates to a subgraph G'' of its parent cluster's subgraph G' of G. As the depth and width of the cluster tree increases, the number of vertices that a cluster or subgraph can have decreases. The root cluster of T contains all vertices of G. The leaves of T are exactly the vertices of G. The nodes of T at depth or level i represent the subgraph $G_i = (V_i, E_i)$ of G. These levels can be visualized as a multilevel view of the clustered graph, where every cluster level is drawn on a different plane and represents a different abstraction level of the graph [13].

When loading an SBGN-ML file, we generate its representing clustered graph C_1 recursively by using compartment affiliations as clustering information. A directed graph G is constructed from the SBGN-ML file, where every glyph's compartment reference is added as attribute to every vertex $v \in V(G)$. Then, we construct the cluster tree T. We add G to the tree's root node and refer to it as dummy cluster, as the root has

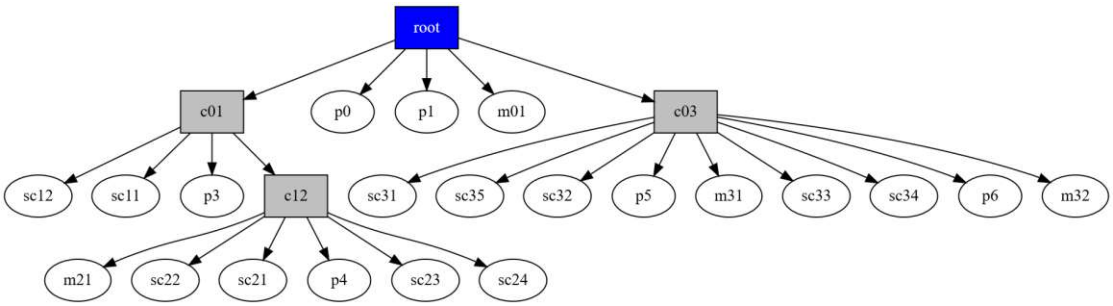
no actual representation in the SBGN-ML file. Subsequently, we recursively add child nodes by, firstly, identifying clusters in the current node of T, and, secondly, adding the corresponding subgraph G' to this node. Then, we continue the procedure at the child node level. As SBGN-ML already contains the cluster information in the form of compartments and compartment references for every glyph, there is no need to apply a clustering algorithm to G to identify clusters. An actual compartment is translated to a cluster c by adding a node to T with the compartment reference as node attribute. The subgraph G' of G contained in cluster c includes a vertex v, if v has the same compartment reference as c, or if v is included in a child cluster of c. Edges are implicitly part of a cluster, if its start and end vertices are part of the same cluster. The result of this procedure is a graph that is clustered by compartment affiliation and represents an SBGN-ML file. This graph allows us to access individual subgraphs included in a compartment as well as all subgraphs on a specific viewing level. In Figure 3.10a we see an example SBGN network with three compartments and one of them being nested. Figure 3.10b shows its corresponding cluster tree, where the rectangular nodes represent clusters and the elliptic leaf nodes represent actual vertices in G. The corresponding multilevel view is depicted in Figure 3.11.

For the purpose of a more granular control and flexibility in the upcoming layout step, we decided to subdivide the clustered graph further by additionally including graph connectivity as clustering information. Again, we start to construct a clustered graph C_2 from the root node of T. Firstly, we add G to the root node of T and then we subdivide G into clusters. As described in the previous paragraph, the clustering is initially performed according to compartment affiliation. For each compartment, we add a new child node to the root node of T. Additionally, we now determine all disconnected components or disjoint subgraphs of G that are not part of any compartment. For each of these components, we add a new child node to the root node of T and, therefore, represent them as separate clusters. These component clusters are depicted as magenta boxes in Figure 3.12b. Since these components are originally not represented in the SBGN-ML file, we refer to their nodes in T as dummy nodes. Again, every cluster is a subgraph G' of G. This subdivision procedure (see Algorithm 1) is then repeated recursively on every subgraph contained in a compartment cluster, until there are no more child compartments remaining. Figure 3.12b presents an example SBGN network with its corresponding cluster tree. The multilevel views of this network are presented in Figure 3.13.

We compare the clustered graph C_1 constructed by compartment clustering with the clustered graph C_2 constructed by compartment and connected component clustering on the basis of their multilevel views. The root viewing level $Level(0, C_1)$ (Figure 3.11a) as well as $Level(0, C_2)$ (Figure 3.13a) show the whole graph. The next viewing level $Level(1, C_1)$ has two clusters (Figure 3.11b), while $Level(1, C_2)$ has four (Figure 3.13b). $Level(2, C_1)$ has one cluster (Figure 3.11c), compared to four clusters in $Level(2, C_2)$ (Figure 3.13c). Every cluster in C_1 contains a subgraph of G that includes all vertices of the cluster's compartment and disregards their connectivity. In contrast, every cluster in C_2 contains a connected subgraph of G only. C_2 allows us to address connected



(a)



(b)

Figure 3.10: An SBGN drawing and the visualization of its graph's cluster tree. (a) shows an SBGN drawing and (b) presents the cluster tree of its clustered graph, performed with compartment clustering. The gray nodes represent clusters associated with the compartments depicted with gray rectangles in (a), the blue node is a dummy cluster.

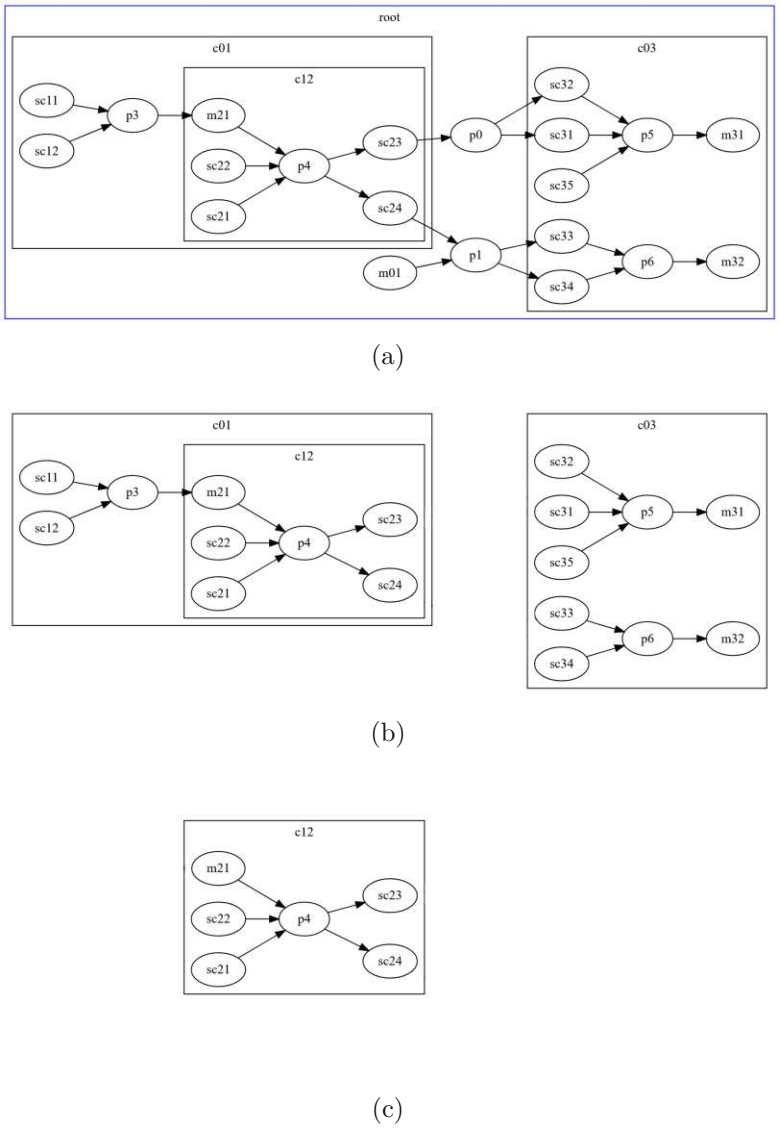


Figure 3.11: The multilevel view of the clustered graph of the SBGN drawing shown in Figure 3.10a. The clustering was performed with compartment clustering. The rectangular outlines depict the cluster tree's clusters, the blue outline represents the root node's dummy cluster, while the black outlines are clusters made from compartments. (a) displays the cluster level zero, the root level, while (b) shows cluster level one, and (c) cluster level two.

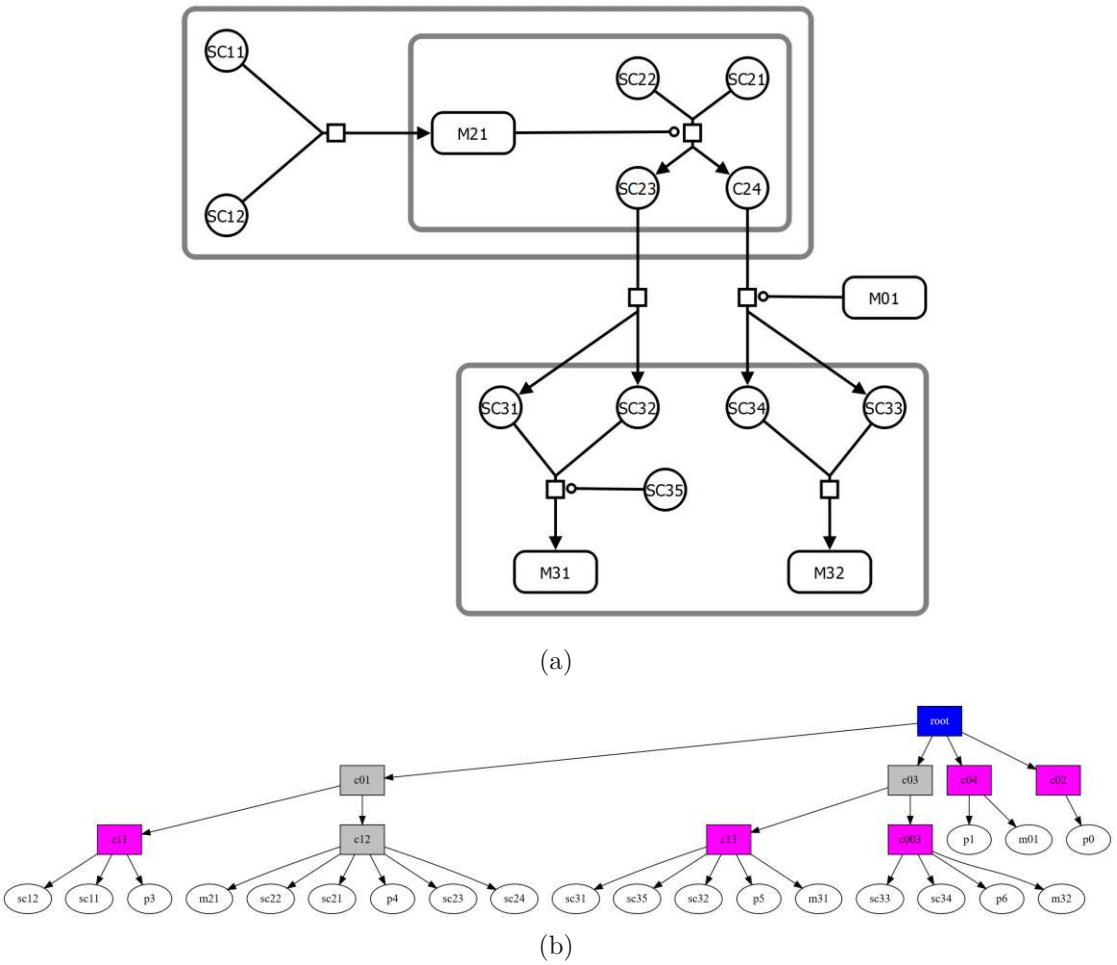
Algorithm 1 Cluster tree algorithm with compartment and disconnected component clustering for an SBGN-ML file.

Input: $G = (V, E)$ *// directed graph built from SBGN-ML*
Output: T *// cluster tree*

```
1: T ← tree with single root node
2: subgraph(root(T)) ← G          // assign the whole graph G as subgraph to the root
3: attr(G, 'CompartmentRef') ← ∅   // G has no compartment affiliation
4: CreateTreeLevel(root(T))       // start building the cluster tree from the root
5: return T

6: procedure CreateTreeLevel(N)                    // N is a node in T
7:    clusters ← Cluster(subgraph(N))              // find clusters in the subgraph of N
8:    for each c ∈ clusters do
9:       N_child ← new child of N                  // add new child node to N
10:      subgraph(N_child) ← c                    // assign the cluster to the child node
11:      if ∄ attr(c, 'Dummy') then               // if the cluster is not a dummy...
12:         CreateTreeLevel(N_child)              // ...recursively continue building T
13:      end if
14:   end for
15: end procedure

16: // find clusters in the subgraph G with clustering criteria:
17: // compartment affiliation, disconnected components, disjoint subgraphs
18: function Cluster(G)
19:   // get compartment affiliation of G
20:   k ← attr(G, 'CompartmentRef')
21:   // get list of subgraphs assigned to child compartments of compartment k
22:   compartments ← compartments p ∈ G where attr(p, 'CompartmentRef') = k
23:   // get list of remaining subgraphs consisting of components that are children of k
24:   components ← disconnected components of G \ compartments
25:   components ← components ∪ disjoint subgraphs of G \ compartments
26:   for each c ∈ components do
27:      attr(c, 'Dummy') ← true                  // mark components as dummy clusters
28:   end for
29:   // return a list of subgraphs
30:   return compartments ∪ components
31: end function
```



(a)

(b)

Figure 3.12: An SBGN drawing and the visualization of its clustered graph's cluster tree. (a) shows an SBGN drawing and (b) presents the cluster tree of its clustered graph, performed with compartment and connected component clustering. The gray nodes represent clusters associated with compartments shown with gray rectangles in (a), the magenta nodes are clusters derived from compartment-based connectivity, the blue node is a dummy cluster.

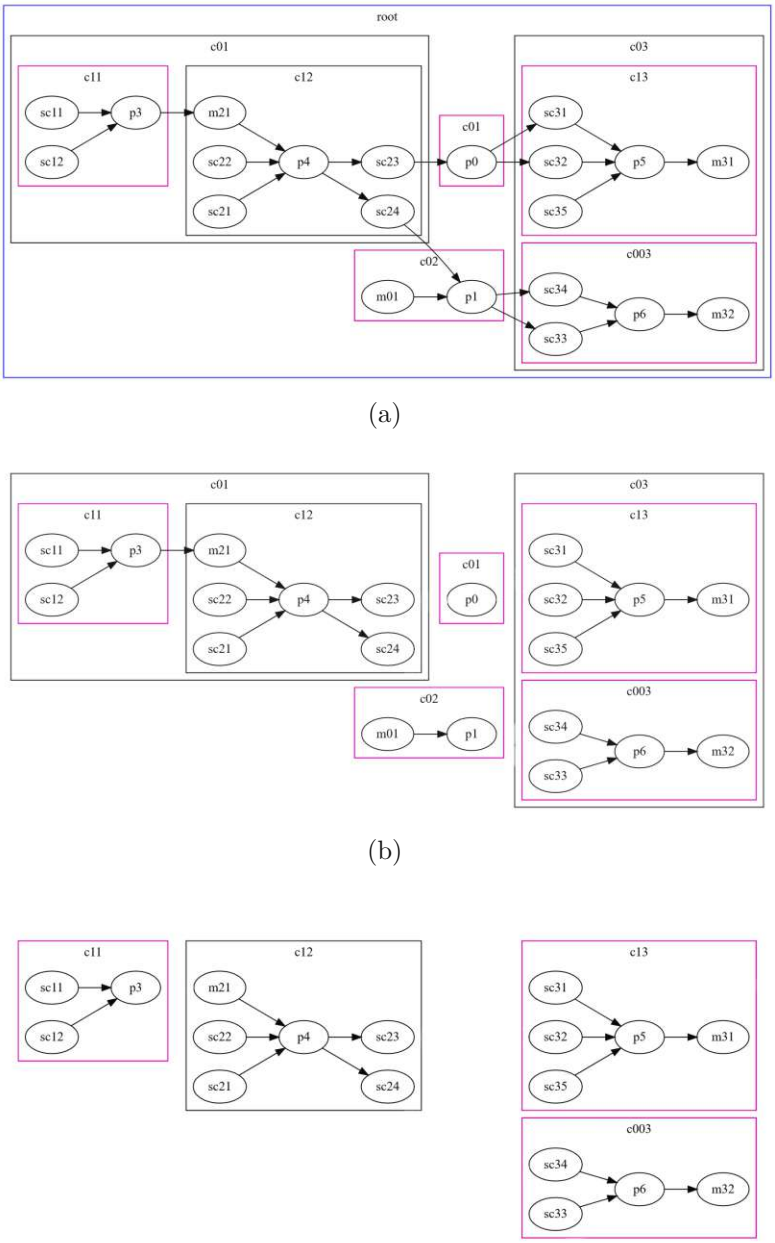


Figure 3.13: The multilevel view of the clustered graph of the SBGN drawing in Figure 3.12a. The clustering was performed with compartment and connected component clustering. The rectangular outlines depict the cluster tree's clusters, the blue outline represent the root's dummy cluster, while the black outlines are clusters made from compartments and the magenta outlines are clusters made from connected components. (a) displays the cluster level zero, the root level, while (b) shows cluster level one, and (c) cluster level two.

components in a compartment separately and, therefore, consider them separately in the following layout step, for details go to Subsection 3.4.4.

To summarize **(C3)**, we model the underlying graph structure of an SBGN drawing as a multivariate clustered graph $C = (G, T)$ with clustering information from compartments and connected components, where each subgraph is a simple directed graph G. All hyperedges are removed through the insertion of dummy nodes. The SBGN glyph's node sizes are incorporated as vertex attributes in G. The different edge lengths are added as edge attributes in G. The compartment affiliation is encoded as vertex attributes in G, but also as clusters of T. The multivariate clustered graph forms the basis for considering the clustering of nodes **(R14)** and for applying expand and collapse **(R23)** techniques to reduce complexity in the following layout steps. This data structure serves as the input for the following layout steps.

3.4 Layout

Decisive for the choice of the layout approach are the domain-specific requirements of biological networks. The approach has to be customizable as well as scalable to large networks. For these reasons and based on recommendations of the related work, recall Section 2.5, we decided to use a constraint-based layout approach. Concretely, we choose the vectorized stress majorization of Wang et al. [103] as layout framework. Their approach combines unconstrained and constrained layouts into one and is able to model a variety of constraints proposed by earlier works on layouts for biological networks. For example it can mimic the separation constraint of IPSep-CoLa [52, 103] and models circular shapes with fewer constraints than Dwyer's approach [48, 103]. In contrast to IPSep-CoLa, the approach models soft constraints and, therefore, is able to merge conflicting constraints with weights [52, 103]. Like the approach of Yuan et al. [107] it is able to merge multiple subgraphs into one layout while preserving each subgraph's mental map. It is said that stress majorization creates layouts with an overall smaller stress value than approaches based on gradient descent [61]. Wang et al. [103] state that the vectorized stress majorization can handle any constraint that can be modeled using vectors. While the vectorized stress majorization has yet to be demonstrated on metabolic pathways, it was shown to be versatile and extendable. Therefore, we consider it a suitable framework to automatically create drawings of biological networks in the SBGN process notation.

Firstly, we recap the vectorized stress majorization in detail [103]. The basic concept is the stress function, recall Equation 2.1 [61, 68]:

$$stress(\mathbf{X}) = \sum_{i<j} w_{ij}(\|\mathbf{x}_i - \mathbf{x}_j\| - d_{ij})^2.$$

This original stress function aims to create a graph where a predefined target edge length d_{ij} is reached between vertices i and j. $\mathbf{X}$ describes the positions of all vertices. Wang et al. [103] extended this function to achieve not only a target edge length but also

a target direction by replacing all distances and positions with vectors:

$$stress(\mathbf{X}) = \sum_{i<j} w_{ij}(\|\mathbf{x}_i - \mathbf{x}_j\| - \mathbf{d}_{ij})^2. \tag{3.1}$$

This is achieved be reformulating the original function through the use of an auxiliary variable $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}^T$ and the Couchy-Schwarz inequality $\|\mathbf{x}\|\|\mathbf{y}\| > \mathbf{x}^T\mathbf{y}$ leading to the target edge vector $\mathbf{d}_{ij}$:

$$\mathbf{d}_{ij} = d_{ij}\frac{\mathbf{z}_i - \mathbf{z}_j}{\|\mathbf{z}_i - \mathbf{z}_j\|}. \tag{3.2}$$

In this equation the target edge length d_{ij} is used as a scale factor and $\frac{\mathbf{z}_i - \mathbf{z}_j}{\|\mathbf{z}_i - \mathbf{z}_j\|}$ is a normalized vector, i.e., a direction. The target edge vector $\mathbf{d}_{ij}$ therefore encodes a target length as well as a target direction. By replacing the target edge length d_{ij} with the target edge vectors $\mathbf{d}_{ij}$ in Equation 2.1, we obtain the vectorized form of the stress function, shown in Equation 3.1. The weight is defined as $w_{ij} = (d_{ij})^{-2}$ and is used as normalization factor. The vectorized stress function considers the complete graph G, where actually existing edges between i and j are termed *real edges*, all others *virtual edges*. G is an undirected graph. Equation 3.1 minimizes the sum of squared differences between the edge vectors $\mathbf{d}_{ij}$ instead of the difference between a real and a target edge length, as it was done in the original Equation 2.1. The equation is solved by bringing it into a matrix form, differentiating it with respect to $\mathbf{X}$, and setting the result to zero. The outcome is the following linear system:

$$\mathbf{LX} = \mathbf{JD}, \tag{3.3}$$

where $\mathbf{L}$ is the weighted Laplacian matrix, $\mathbf{J}$ is the matrix of weights of the edge vectors, and $\mathbf{D}$ is a matrix holding all $\mathbf{d}_{ij}$ vectors of all real and virtual edges in G, referred to as the vector of directions. This final equation is solved by a block-coordinate descent method, by alternately updating the vector directions $\mathbf{D}$ with the current configuration of $\mathbf{X}$, called D-step, and calculating the new vertex positions by solving the linear system from Equation 3.3, called P-step.

Constraints can be directly integrated into the vectorized stress function by extending it as follows:

$$stress(\mathbf{X}) = \underbrace{\sum_{i<j} w_{ij}(\|\mathbf{x}_i - \mathbf{x}_j\| - \mathbf{d}_{ij})^2}_{\text{considers real and virtual edges in } G} + \underbrace{\sum_{(i,j)\in E'} v_{ij}(\|\mathbf{x}'_i - \mathbf{x}'_j\| - \mathbf{d}'_{ij})^2}_{\text{considers real and virtual edges in } G'}, \tag{3.4}$$

where $G'(V', E')$ is a constraint graph with $V' \subseteq V$ and $E' \subseteq E$. E' includes real as well as virtual edges and G' is a directed graph. Equation 3.3 is extended to

$$(\mathbf{L} + \mathbf{L}')\mathbf{X} = \mathbf{JD} + \mathbf{J}'\mathbf{D}', \tag{3.5}$$

where $\mathbf{L}'$, $\mathbf{D}'$, and $\mathbf{J}'$ are the matrices corresponding to $\mathbf{L}$, $\mathbf{D}$ and $\mathbf{J}$, but only hold data from G'.

The constraints are derived from user input, or, as in our case, from requirements of an application domain, and are defined through the target edge vectors $\mathbf{d}'_{ij}$. These vectors give an edge a specific length and direction, imposing a constraint on the edge. The target edge vectors $\mathbf{d}'_{ij}$ are weighted by the application-specific weight v_{ij}. This way each constraint is weighted individually, which allows conflicting conditions to be satisfied, making them to soft constraints. Solving Equation 3.5 only differs from the unconstrained version in the D-step, where $\mathbf{d}'_{ij}$ and, consequently, $\mathbf{D}'$ may need to be recalculated in every iteration, depending on the constraints. The stress majorization stops at a user-defined number of iterations. With this integration of constraints into the vectorized stress function, Wang et al. [103] created a unified framework for constrained as well as unconstrained graph visualizations.

Based on the literature, we perform the following sub-steps to create a constraint layout: initial embedding, initial layout, and constraint layout. Since we chose an energy-based layout approach, an initial embedding is necessary and it influences the basic properties of the resulting drawing, like orientation. The initial layout step *untangles* the graph. The choice of the approach also influences which characteristics of the graph are highlighted in the final drawing. The constraint layout modifies the initial layout according to the application-specific requirements.

In the following sections, we discuss how to find a suitable initial embedding, adjust the initial layout to fit the needs of the application domain, formulate the necessary constraints to enforce the rules of the SBGN, and how to define the order of the constraint's application.

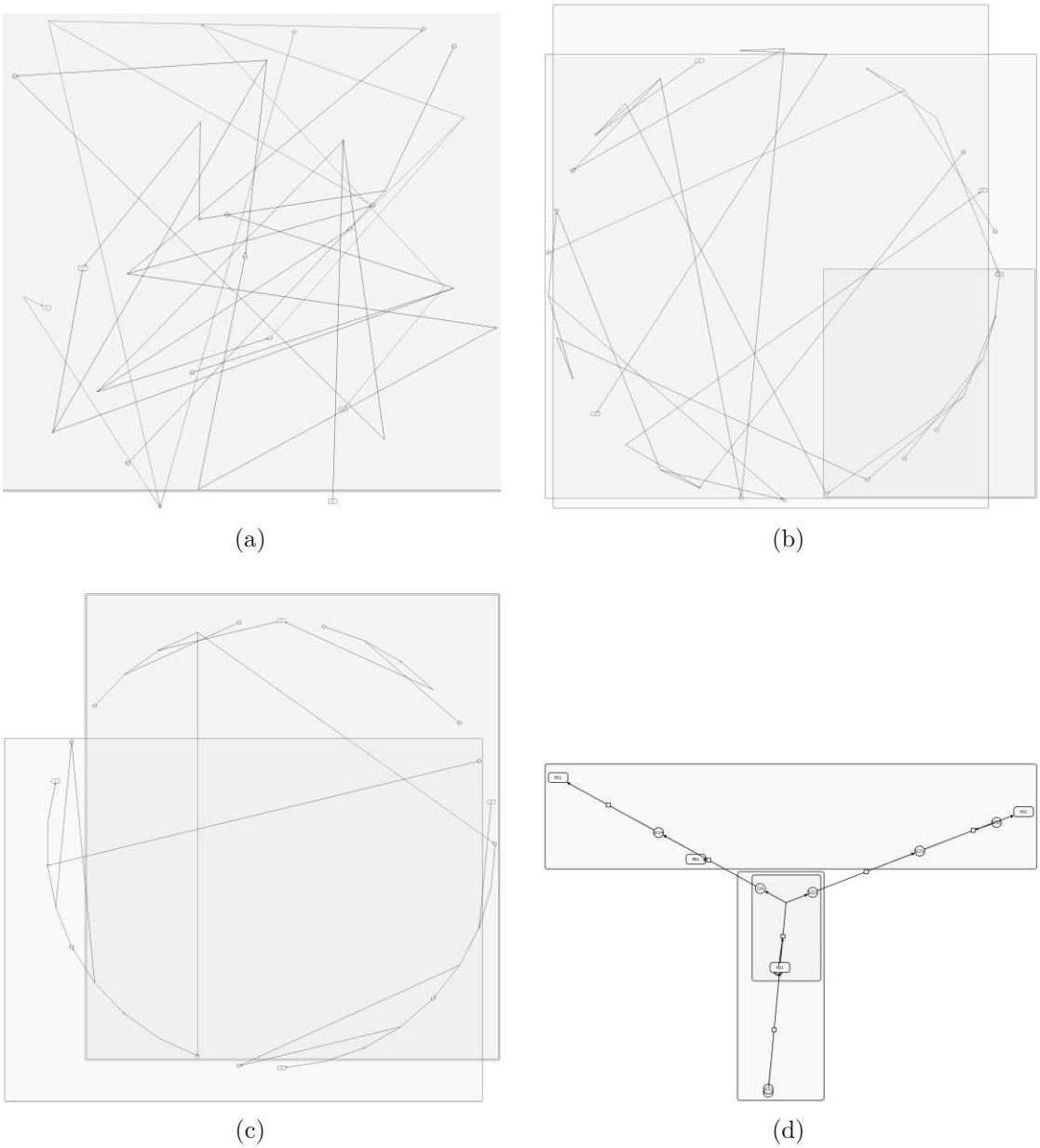
3.4.1 Initial Embedding

The first step in creating a constraint layout, or generally an energy-based layout, is the creation of an initial embedding. The initial embedding defines the starting positions of the graph's vertices in a two-dimensional space and is used as input for the actual layout process. The initial embedding influences the subsequent layout steps by contributing to the orientation of the graph, the number of edge intersections **(R05)**, and the convergence speed of the layout algorithm, among others. Choosing an appropriate method for generating an initial embedding facilitates the creation of a favorable layout. Subsequently, we discuss several initial embeddings that we explored throughout the course of our work.

A straightforward initial embedding is to use the one stored in the SBGN-ML file. Since our goal is an automated layout without user interaction, we ignore the stored glyph positions and replace them with a newly calculated initial embedding.

Another simple initial embedding is a random initialization of the vertex positions. This approach was used as an initial embedding in the work of Wang et al. [103]. An exemplar random embedding is shown in Figure 3.14a.

Kamada et al. [68] and Yuan et al. [107] define the initial embedding by placing all vertices on a circle. This circle's diameter is the screen width and all vertices are placed



(a) (b)

(c) (d)

Figure 3.14: Different initial embeddings created for the example SBGN drawing shown in Figure 3.10a. (a) shows a random embedding. (b) presents a circle embedding with the vertices being placed in the same order as the nodes are stored in the graph. (c) shows an ordered circular embedding following a depth-first traversal of the nodes. (d) displays the pivot MDS embedding with the MinMax strategy and four pivots.

equidistantly on it [68]. Having a fixed diameter is suitable for cases with a small number of vertices, but does not scale well as vertices might be placed too close to each other. Initially, we place the vertices on the circle in the same order as the nodes are stored in the graph. An example of a circle embedding is shown in Figure 3.14b. The circular placement order of the vertices strongly affects the subsequent layout step. For example, the number of edge crossings can be reduced, if the order is chosen properly. For that reason, we order the vertices based on a depth-first traversal of G. The first node stored in the graph is selected as start vertex for this traversal. An example of this ordered circular embedding is shown in Figure 3.14c. The subsequent layout step depends now on the chosen start vertex for the depth-first traversal, but finding a favorable root vertex of a graph is a non-trivial problem.

Multi Dimensional Scaling (MDS) is another technique suggested by the literature to create initial embeddings. Combined with layout techniques based on stress majorization, favorable final layouts are created since these two techniques are related in a mathematical sense [61]. MDS is a dimensionality reduction technique to represent high-dimensional data in a low-dimensional space. In general, MDS can be used to visualize the structure of a graph. The first version of MDS, known as *classical* MDS, has a global optimum and can be directly computed by spectral decomposition [100]. It has a high computational complexity, i.e., quadric with respect to the input data, and, thus, is not suitable for large graphs [32].

Classical MDS was improved by *distance scaling* [75]. Distance scaling tries to match the Euclidean distance to the graph theoretical distance by minimizing a stress measure on the graph. Stress is modeled with retracting and repelling spring forces [32]. This idea inspired the layout approach of Kamada et al. [68] and makes the mathematical relation to the stress function apparent.

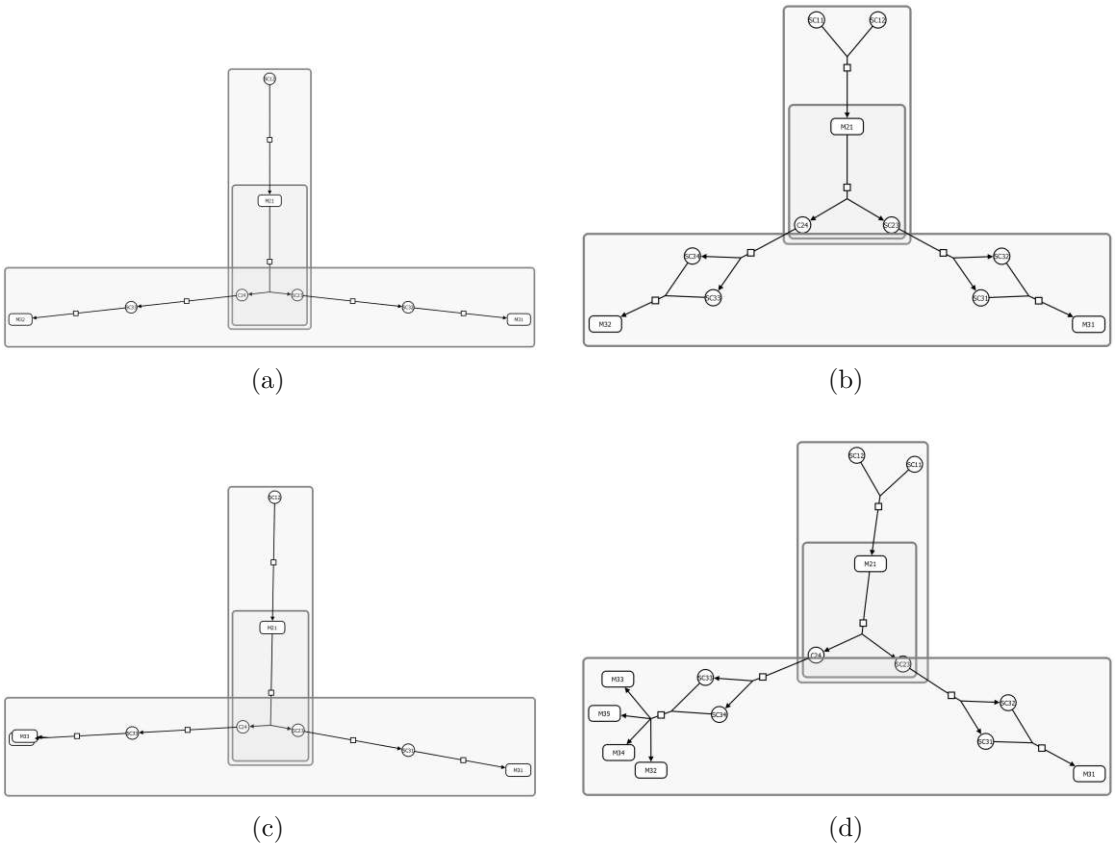
Another evolution of classical MDS is *landmark* MDS [93], an efficient approximation of classical MDS. It selects a set of pivot nodes, also called landmarks. A matrix of squared distances, referred to as distance matrix, is obtained by computing the shortest path only from each landmark to all other nodes. The initial embedding is then calculated through eigen decomposition. Landmark MDS has a smaller time and space complexity than classical MDS.

Another improved version of classical MDS is *pivot* MDS [32]. It is based on the same idea as landmark MDS and also approximates the classical MDS through a set of pivot nodes. But instead of using only the distance information of these nodes in the distance matrix, it uses the distances from all nodes to these pivots, and, therefore, includes the complete graph's distance information. Moreover, instead of squared distances it uses rectangular distances to construct the distance matrix. Pivot MDS is faster than the classical MDS and makes it thereby suitable for large graphs. It facilitates progressive computation as well as iterative refinement. Compared to landmark MDS it needs fewer pivots to reach the same stability. While landmark MDS is faster, pivot MDS results in more informative layouts [32]. An example of the pivot MDS embedding is shown in Figure 3.14d. Pivot MDS is recommended for the efficient generation of an initial

embedding, which is then used as input for subsequent algorithms [32]. Consequently, we are also using pivot MDS to generate our initial embedding.

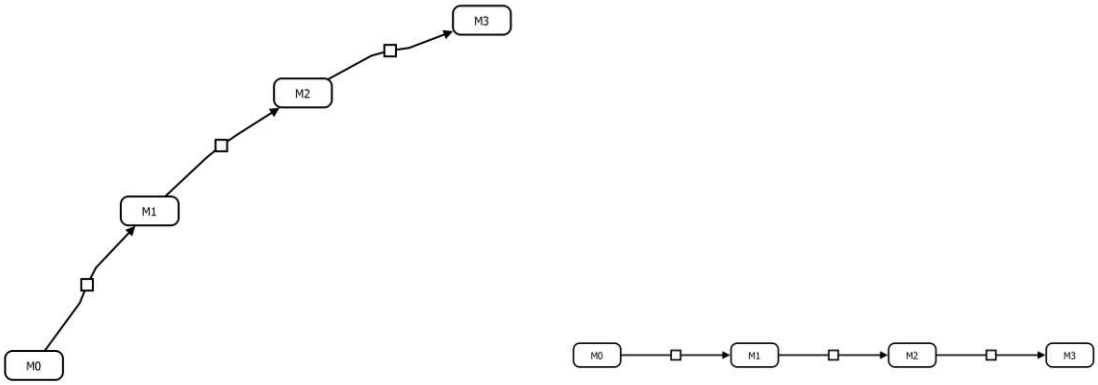
There are different strategies to select the pivots. One of them, which reportedly gives good results, is MinMax sampling [73]. The first pivot k_0 is chosen randomly from the vertex set V, and the next pivot k_i is chosen from the $i-1$ vertices such that it has the longest path possible from k_0. This minimizes the longest distance from V to its pivots. The minimum number of pivots is $d+1$, where d is the dimensionality of the embedding, but a larger number than $d+1$ is recommended [93]. The optimal selection of pivot points as well as their optimal number are still open problems [73].

We decided to select all nodes of G with node degree ≥ 4 as pivots. We consider them strongly connected and, therefore, important to approximate G. In case this selection strategy leads to less than three selected pivots, which is the absolute minimum for



(a) (b) (c) (d)

Figure 3.15: The pivot MDS embedding of an asymmetric graph introduces a slight rotation in comparison to its symmetric counterpart with a vertical symmetry axes. (a) shows a pivot MDS embedding of a symmetric graph with its unconstrained stress layout presented in (b). (c) shows a pivot MDS embedding of an asymmetric graph with its unconstrained stress layout presented in (d).



(a) (b)

Figure 3.16: Comparison of circle embedding and pivot MDS embedding and an unconstrained stress layout, applied on a path-like graph. (a) shows an unconstrained stress layout made from a circle embedding, while (b) uses pivot MDS.

a two-dimensional embedding [93], we use a fallback selection strategy. It defines the number of pivots as 10% of the vertices in G, but if there are less than three, MinMax is used. This fallback strategy can become necessary for small and simple graphs.

We found that the orientation of the pivot MDS embedding depends on the structure of the graph. A symmetric graph has a uniform weight distribution, in terms of graph theoretical distance, and the graph is aligned with the canvas. If we unbalance the same graph by adding vertices on one side of its symmetry axis, the initial embedding will start to rotate due to the unequal weight distribution. This orientation is then propagated to the initial layout, as demonstrated in Figure 3.15.

In a direct comparison between a circle embedding and the pivot MDS embedding, we can see the limitations of the circle embedding for straight path segments [61], as shown in Figure 3.16. The circle embedding in combination with the unconstrained stress layout cannot straighten line segments. In contrast, pivot MDS followed by an unconstrained stress layout can achieve this.

In summary, we evaluated several methods to find an initial embedding: random, circle, circle with depth-first traversal, and pivot MDS. Finally, we decided to use pivot MDS with highly-connected vertices as pivots, because it leads to good final layouts in combination with stress majorization [61]. The initial embedding is used as input for the subsequent step to calculate the initial layout.

3.4.2 Initial Layout

The purpose of the initial layout step is to unravel the graph. This is accomplished by computing a force-directed layout, but without considering any constraints [48]. An

initial layout can be calculated by solving the unconstrained stress function, recall Equation 3.1 and Section 3.4, using any undirected, connected, weighted graph with an initial embedding and a desired edge length as input. This initial layout step does not enforce specific edge directions. This is accomplished by updating the direction vectors in the D-step, recall Equation 3.6, with the vertex positions of the previous iteration, as seen in [103]:

$$\mathbf{d}_{ij} = d_{ij} \frac{\mathbf{x}_i(t) - \mathbf{x}_j(t)}{\|\mathbf{x}_i(t) - \mathbf{x}_j(t)\|}, \tag{3.6}$$

where the distance d_{ij} between two vertices i and j is usually defined as $d_{ij} = s_{ij}$, with s_{ij} being the graph theoretical distance or shortest path distance [61, 68]. The stress majorization is finished, if a user-defined maximum number of iterations is reached or if the stress value drops under a specified minimum. The result of this process is an organic-looking layout with a uniform edge length **(R07)**, preserved proximities between related nodes by minimizing the edge length **(R09),** and with an emphasis on the graph's symmetry **(R19)** [49, 61, 68]. This layout approach can lead to *hair balls* in dense graphs, and is limited to connected graphs [61, 65, 98]. It does not inherently reduce edge crossings or enforce a uniform node distribution on a grid (R03). To support uniform node distribution, other layout approaches would be more suitable, e.g., one derived from TSM [92]. TSM-based approaches support orthogonal edges (R08) and consider uniform node distributions (R03). Since an orthogonal layout and a uniform edge distribution is not strictly required by the SBGN, we accept at least a uniform node distribution ~~(R03)~~ as a limitation of our approach.

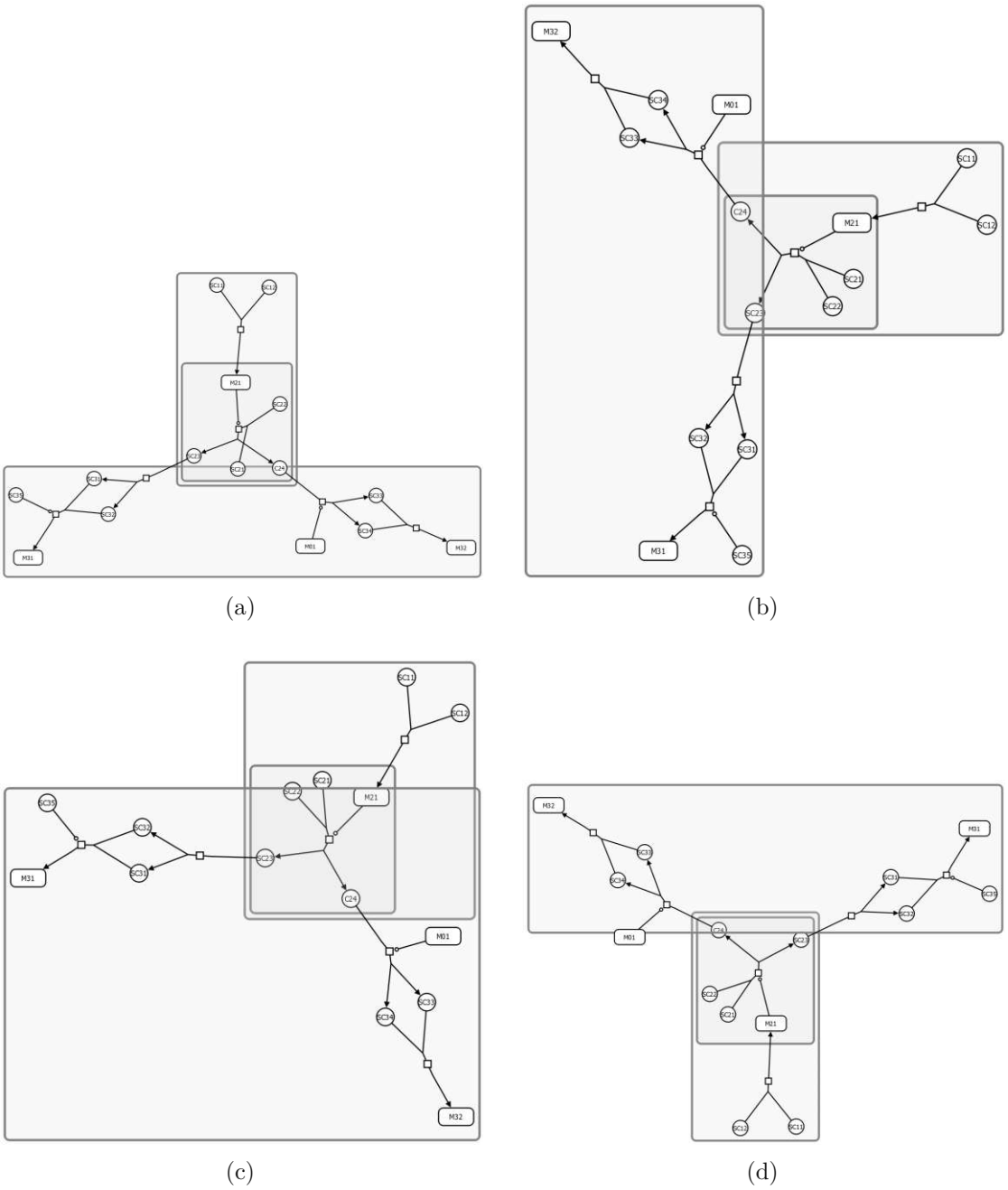
We modulate the vectorized stress majorization to fit our application domain, the visualization of biological pathways. Since the graph obtained from the SBGN data is a directed graph, but the unconstrained stress function only considers undirected graphs, we ignore edge directions for the initial layout. We further redefine the distance d_{ij} as follows:

$$d_{ij} = s_{ij} \cdot l \cdot n_{ij}, \tag{3.7}$$

where s_{ij} is the shortest path distance calculated with the Dijkstra algorithm [41], l is the predefined length of its edge type, and n_{ij} is the neighborhood weight. Instead of a uniform edge length as in general graphs, the SBGN requires different edge lengths for process handles and arcs. We accomplish this by scaling the shortest path distance with length l that depends on the edge type, as shown in Equation 3.8. We need two lengths, one for the real edges as well as virtual edges, referred to as l_1 and another one for dummy edges, referred to as l_2:

$$l = \begin{cases} l_1 & \textbf{if } e_{ij} \in (\textit{real edges} \cup \textit{virtual edges}), \\ l_2 & \textbf{if } e_{ij} \in \textit{dummy edges}. \end{cases} \tag{3.8}$$

The lengths of the edges are user-defined in pixels and are encoded as edge attributes in the graph. Using pixels instead of the graph theoretical distance in the layout process, has the advantage of having, both, the edge length and node size in the same unit (pixels). This is necessary to calculate the constraints in the subsequent step. Alternatively,



(a) (b) (c) (d)

Figure 3.17: Different initial layouts without neighborhood weights of the SBGN drawing from Figure 3.10a with the initial embeddings shown in Figure 3.14. (a) shows an initial layout using random embedding. (b) displays an initial layout derived from circle embedding. (c) shows an initial layout obtained from an ordered circular embedding. (d) shows an initial layout using a pivot MDS embedding.

to integrating the edge length into the initial layout, we could enforced different edge lengths via constraints, but this would potentially add more conflict between constraints and introduce unnecessary stress to the system. In real life graphs as well as SBGN networks, the distribution of node degrees is often highly non-uniform [61]. A uniform edge length causes the neighborhood of high-degree nodes to become dense, resulting in visual *hairballs* in the graph [61]. In SBGN data sets, nodes with the highest degrees are usually process glyphs or submap glyphs, leading to dense and cluttered areas around them. This can be improved by weighting edges by their nodes' neighborhood sizes [61]. We calculate the neighborhood weight n_{ij} as follows:

$$\begin{aligned} n_{ij} &= n_w \cdot |N_i \cup N_j| - |N_i \cap N_j| \\ &= n_w \cdot |N_i| + |N_j| - 2, \end{aligned} \tag{3.9}$$

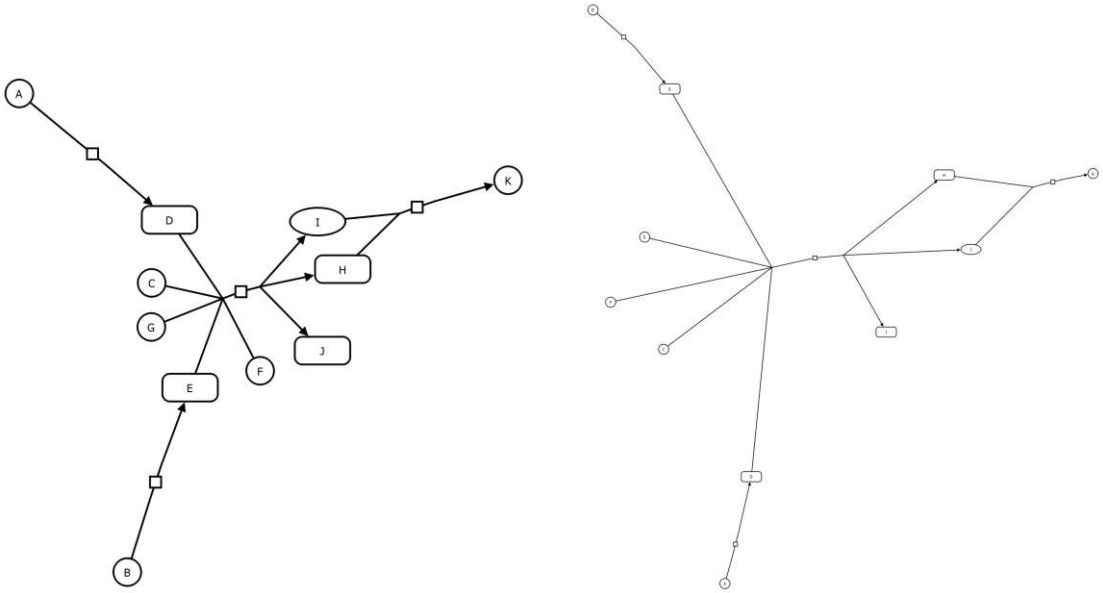
where N_i is the neighborhood of vertex i defined as $N_i = \{j|(i,j) \in E\}$ and n_w is an additional weight based on the edge type. The introduction of n_w allows us to follow the SBGN process description recommendation by placing process ports near the process symbol [80]. While the neighborhoods of real and virtual edges are not further weighted, we weight the neighborhood of dummy edges by 0.25. The reason for this weight is to give a pair of dummy edges half the weight of real and virtual edges:

$$n_w = \begin{cases} 1 & \textbf{if } e_{ij} \in (\textit{real edges} \cup \textit{virtual edges}), \\ 0.25 & \textbf{if } e_{ij} \in \textit{dummy edges}. \end{cases} \tag{3.10}$$

Including the neighborhood weight in the calculation of the distance d_{ij}, generates more space around high-degree nodes and, thereby, reduces node overlaps **(R02)**, edge crossings **(R05)**, and node-edge overlaps **(R10)**. This results in a more informative layout [61]. For our experiments, we chose a maximum of 50 iterations to solve the vectorized stress function. Larger graphs may require more iterations, although their number increases only moderately [61]. An example calculated from different initial embeddings, but without neighborhood weighting can be seen in Figure 3.17.

An example of an initial layout including neighborhood weights can be seen in Figure 3.18b. Compared to its equivalent drawing without neighborhood weights (see Figure 3.18a), the resulting layout is more spread out and less cluttered.

To summarize, we calculate the initial layout based on the unconstrained vectorized stress function of Wang et. al. [103], but modified it to integrate requirements of the SBGN process notation, without relying on constraints and introducing additional stress. We can scale edges based on their edge type to their final size, resolve dense graph areas, and reduce edge crossings as well as node overlaps.



(a) (b)

Figure 3.18: Effect of neighborhood weights on the initial layout created from a pivot MDS embedding. (a) shows the initial layout without neighborhood weights and (b) displays the initial layout of the same graph with neighborhood weights. These weights lead to extended edges and a graph that takes up more area. This can be seen from the zoom out and the resulting smaller displayed nodes.

3.4.3 Constraint-Based Layout

At this point, we are able to generate force-directed layouts, which enhance symmetry, inherently reduce edge crossings and overlaps, and aim at the desired edge lengths. To further support the requirements and layout guidelines of the SBGN, we need to customize the initial layout further by applying constraints.

The vectorized stress majorization supports three types of constraints: direct, metrics-based, and shape-based constraints. Direct constraints define target edge vectors $\mathbf{d}'_{ij}$ with a certain length and/or direction. Metrics-based constraints calculate target edge vectors from a set of nodes through a quality metric, while shape-based constraints define a reference shape to which the associated nodes are fitted [103]. We already derived syntactic as well as semantic graph drawing requirements from the literature as well as the SBGN definition [77] in Section 2.4. In the following paragraphs, we discuss requirements that we can formulate through edge vectors as constraints in the framework of vectorized stress majorization, recall Table 3.1. Firstly, we review requirements that are already implicitly addressed in the previous layout steps, then we discuss the requirements we consider future work, followed by requirements we consider limitations, and finally we present requirements that we formulate as constraints.

We already addressed the reduction of edge crossings while creating the initial embedding **(R05)**, but it is possible to consider it in this layout step as well. In their work Wang et al. [103] demonstrate a metric constraint to minimize edge crossings for small graphs or subgraphs. However, they mention, that the reduction of edge crossings is still an open problem for large graphs and cannot be solved with their constraint [103]. For this reason, we do not minimize edge crossings (R05) with a constraint in our work.

The initial layout step optimizes the graph to have a uniform edge length **(R07)**, but does not consider the nodes sizes. Two large nodes connected by an edge can visually appear closer together than two small nodes connected by an edge of the same length. This could be remedied by a constraint, which augments the edge length through the node sizes, but we consider this an improvement and therefor future work.

Dwyer et al. [49] demonstrated that the flow direction (R15) can be incorporated into a constraint layout with Dig-CoLa. An equivalent constraint can be formulated as direct constraint in the vectorized stress majorization. Since we could not estimate the interference of this constraint with other ones and our main focus is on an SBGN-compliant layout, we decided to leave this constraint for future work.

The initial stress function inherently enhances global symmetry **(R19)**, so we already addressed this requirement in previous layout steps. Wang et al. [103] demonstrate an additional symmetry constraint for user-defined subgraphs to increase local symmetry. In our work, we consider only global symmetry and see local symmetry as future work.

Similar subcomponents and substructures should be drawn in the same way (R18). Possible relevant information are graph substructures like circles or the relative hierarchy of biological reactions [49], and also repeating processes. By finding topologies of substructures and semantic matching, a shape catalogue can be build. This shape catalogue can then be used to formulate shape constraints. These can then be applied on similar topologic or semantic substructures to achieve similar sub-layouts. Since satisfying this requirement is beyond the scope of this thesis, we consider it a future endeavor.

The requirement to allow fixed positions for nodes of special interest ~~(R13)~~ is difficult to realize in the context of vectorized stress majorization. Since the framework defines constraints through vectors they enforce relative layout rules. While it is possible to remove the influence of all other nodes to a node of special interest by modeling an inverse weight function, this does not fit into the concept of the approach. If only one node would be fixed in position the layout would revolve around this point, only changing its reference point and not its shape. Therefore we consider this requirements as limitation.

The initial layout does not inherently produce orthogonal edges (R08). This behavior can be accomplished with a constraint, namely the *general orthogonal constraint*, as shown in Figure 3.19. The aesthetic of orthogonal edges in combination with a uniform node distribution (R03) could be enforced with alternative approaches [92]. Since the SBGN does not expect edge orthogonality on all edges, we do not use the general orthogonal

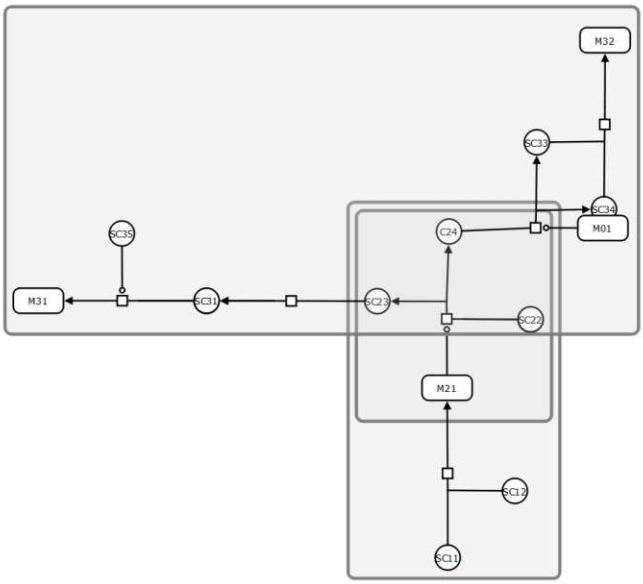


Figure 3.19: Result of the general orthogonal constraint, which forces edges to become orthogonal.

constraint. For the sake of understanding and completeness, we have nevertheless formulated and implemented it.

Some SBGN glyphs, e.g., complexes or multimers, are specifically designed to represent the nesting or stacking of nodes. The standard glyph representation of a single node forbids the overlapping of nodes. The stress function does not inherently consider node sizes, it models nodes only as sizeless points. Therefore, a constraint is required to prevent nodes from overlapping **(R02)**. We formulate this constraint as the *non-node overlap constraint*.

Disjoint compartments should should not overlap **(R14)**. The initial layout has no knowledge of clusters or containers and, therefore, cannot take them into consideration. The knowledge of compartments can be integrated into the layout via a constraint, to avoid compartment overlaps and to repulse nodes unrelated to a compartment. We name this constraint *compartment constraint*.

The initial layout does not inherently maximize the minimum angles between edges. While Wang et al. [103] demonstrate a star constraint for this purpose, it is not customized to the special layout definitions of process glyphs. Instead of spreading out evenly around a process handle, incoming edges (consumption arcs) and outgoing edges (production arcs) must stay in one hemisphere. Modulatory arcs are placed ideally perpendicularly to the process handles or evenly spread on their respective side. To satisfy the maximizing minimum angles **(R16)** and the connection point requirement **(R17)** in a manner consistent with the SBGN definitions, we define the *process node constraint* as well as the *modulatory arc constraint*. While we rejected a global orthogonal edge behavior, we apply

a *localized orthogonal constraint* **(R8)** on process handles only to achieve the defined visual representation of process glyphs.

The constraints are formulated by defining $\mathbf{d}'_{ij}$ of Equation 3.4 and are subsequently applied on the initial layout. As suggested by Dwyer [48], the constraints are applied in separate refinement steps after an initial layout was computed. This improves the layout iteratively. Every refinement step has a maximum number of user-defined iterations. In the following, we formulate a direct and an iterative non-node overlap constraint, a compartment constraint, a general orthogonal constraint, a process node constraint consisting of a process orthogonal constraint as well as a process equal angle constraint, and a modulatory arc constraint. With these newly formulated constraints we aim to satisfy the SBGN **(C1)**.

Direct Non-Node Overlap Constraint

Node overlaps can be caused by large nodes with long node labels or high connectivity of nodes. Reducing overlap, improves readability. We achieve this by adapting the non-overlap constraint of Wang et al. [103] for clusters in such a way that it can be applied on nodes. The *direct non-node overlap constraint* is formulated as metric constraint, which pushes overlapping nodes apart. The distance required for two nodes i and j to no longer overlap is referred to as Minimal Penetration Depth (MPD) and is calculated with the help of the Minkowski Difference as follows [48]. Firstly, we calculate the two nodes' individual convex boundaries, depicted as shapes A and B, as shown in Figure 3.20a. Then, we reflect shape B at one of its vertices, which we choose as the origin of the shape, and denote the reflection with $-B$. This is equivalent with a 180° 2D rotation around the shape's origin, as shown in Figure 3.20b. We then build the Minkowski Sum $A + (-B)$, which results in the Minkowski Difference $A - B$ [48]. Geometrically, this means $-B$ is placed with its origin at every vertex of A and the convex hull of this new shape is $A - B$, see Figure 3.20c. The minimum distance of B's origin to the boundary of the shape $A - B$ is the MPD [48], as shown in Figure 3.20d. The MPD vector $\mathbf{m}$ is the minimum displacement of node j required to no longer overlap with node i, as demonstrated in Figure 3.20e.

Instead of using the nodes' convex hulls for shapes A and B, we could have defined these shapes as rectangular bounding boxes. However, the convex hull provides a versatile non-overlap metric for convex shapes in general. We included a user-defined node margin when calculating A and B to account for space between nodes. The MPD vector $\mathbf{m}$ is then used to calculate a target edge vector $\mathbf{d}'_{ij}$ between i and j, as shown in Figure 3.20e and Figure 3.20f:

$$\mathbf{d}'_{ij} = (\mathbf{x}_i - \mathbf{x}_j) + \mathbf{m}, \tag{3.11}$$

We apply $\mathbf{d}'_{ij}$ on a virtual or real edge between the nodes i and j to force them apart, if they overlap in the initial layout. $\mathbf{d}'_{ij}$ is only calculated once before the refinement step, a node displacement can cause a new overlap with another node. This problem is

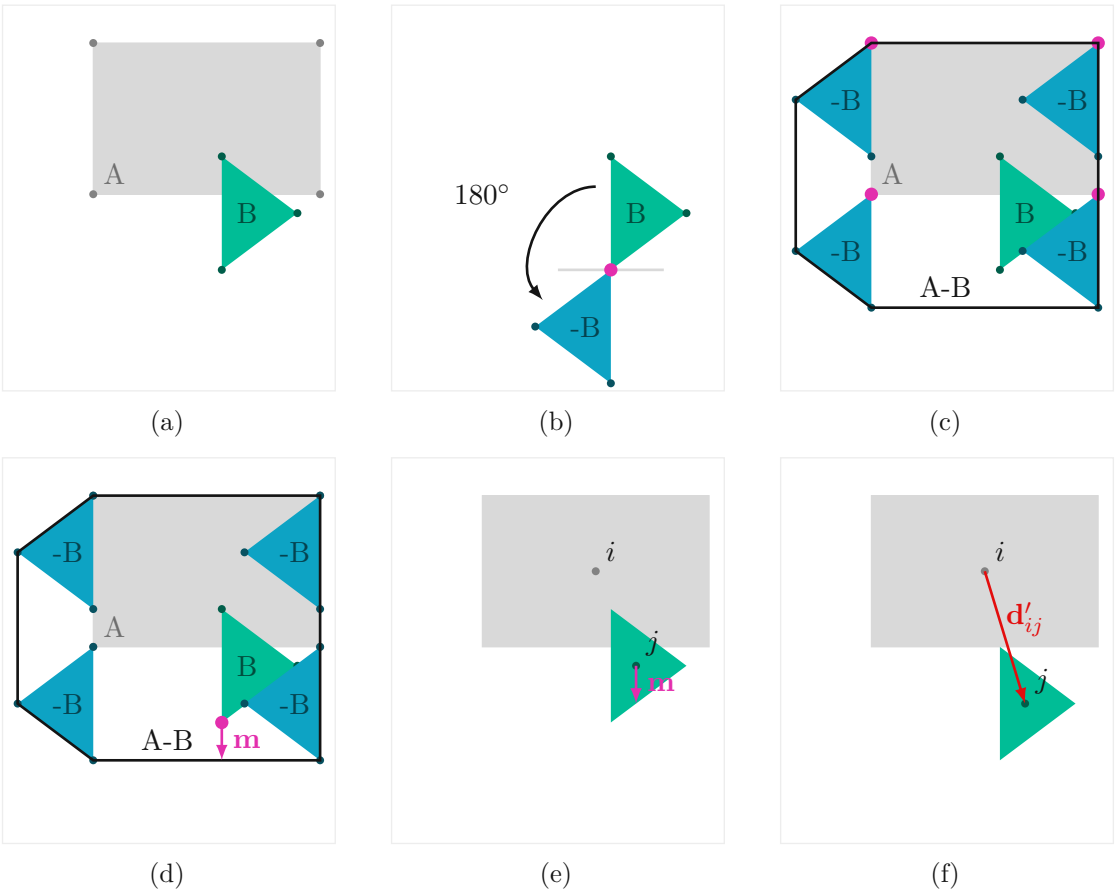


Figure 3.20: The non-node overlap constraint is calculated using the vector Minimal Penetration Depth (MPD) depicted as $\mathbf{m}$. It is geometrically defined as follows: (a) shows the convex boundaries of example shapes A and B. (b) reflects shape B by rotating it by 180° around a point, the origin visualized as the magenta dot, of the convex boundary of B, denoted as $-B$. (c) presents the Minkowski Difference $A - B$ by placing $-B$ with its origin at every point (gray dots) of the convex boundary of A and by calculating the convex boundary of this new shape. (d) shows the MPD vector $\mathbf{m}$, calculated as the minimum distance between B's origin and the boundary of $A - B$. In (e) the vector $\mathbf{m}$ is added to the node j's center position. This is the minimum displacement required that A and B no longer overlap. (f) The vector between node j's new center and the node i's center is the target edge vector $\mathbf{d}'_{ij}$, which is used as constraint.

addressed by repeating the refinement step while still including the non-overlap constraint from the previous refinement step to prevent oscillating behavior. The non-node-overlap refinement is repeated until all overlaps are resolved, or as user-defined number of iterations is reached.

Iterative Non-Node Overlap Constraint

This constraint is an alternative constraint to the direct non-node overlap constraint and also aims to remove node overlaps. We apply this constraint on every real and virtual edge of the graph. We calculate the MPD and $\mathbf{d}'_{ij}$ analogously to the direct non-node overlap constraint, but instead of calculating $\mathbf{d}'_{ij}$ once, we calculate it for every iteration of the stress minimization, as long as node i and j overlap:

$$\mathbf{d}'_{ij} = \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|} \cdot \begin{pmatrix} (w_i + w_j) \\ (h_i + h_j) \end{pmatrix} \cdot 0.5, \tag{3.12}$$

where w_i and h_i are the width and height of node i, and w_j and h_j are the width and height of node j. The width and height of a process glyph exclude the handles. The extent of the glyphs of all other objects includes the rectangular boundaries of their labeled boxes. Instead of moving one node the entire length along direction $\mathbf{d}'_{ij}$ both nodes i and j are moved half the way in opposite directions. This leads to a diagonal displacement. Alternatively, we could move them only along the shorter axis, leading to an axis-aligned displacement [48]. With the iterative non-node overlap constraint no additional refinement steps are necessary apart from the user-defined number of maximum iterations.

Compartment Constraint

Compartment glyphs represent subcellular locations and contain, for example, macromolecules, simple chemicals, and process glyphs as well as other compartments. They form a nested graph structure. Since the initial layout does not consider compartment boundaries, nodes could be placed inside a compartment without having an associated parent child relationship in the data structure, as shown in Figure 3.21b, we refer to these nodes as *misplaced*. This is a visual falsification of the original information and needs to be rectified. We define a *compartment constraint* based on the non-overlap constraint of Wang et al. [103] and our direct non-node overlap constraint to place a compartment's children only inside it. After the initial layout, we use the positions and boundaries of the compartment's child nodes as well as user-defined node margins to determine the convex hull of the compartment. Subsequently, we determine the MPD of overlapping compartments on the same nesting level, as described concerning our direct non-node overlap constraint. This is done for real and dummy compartments, recall Section 3.3, and allows us to move the entire content of a compartment in a specific direction, as presented in Figure 3.21c and Figure 3.21d. If we would individually consider the nodes of a dummy compartment, they could be moved out of the overlapping compartment into different directions, increasing edge length and preventing a consistent flow direction, as

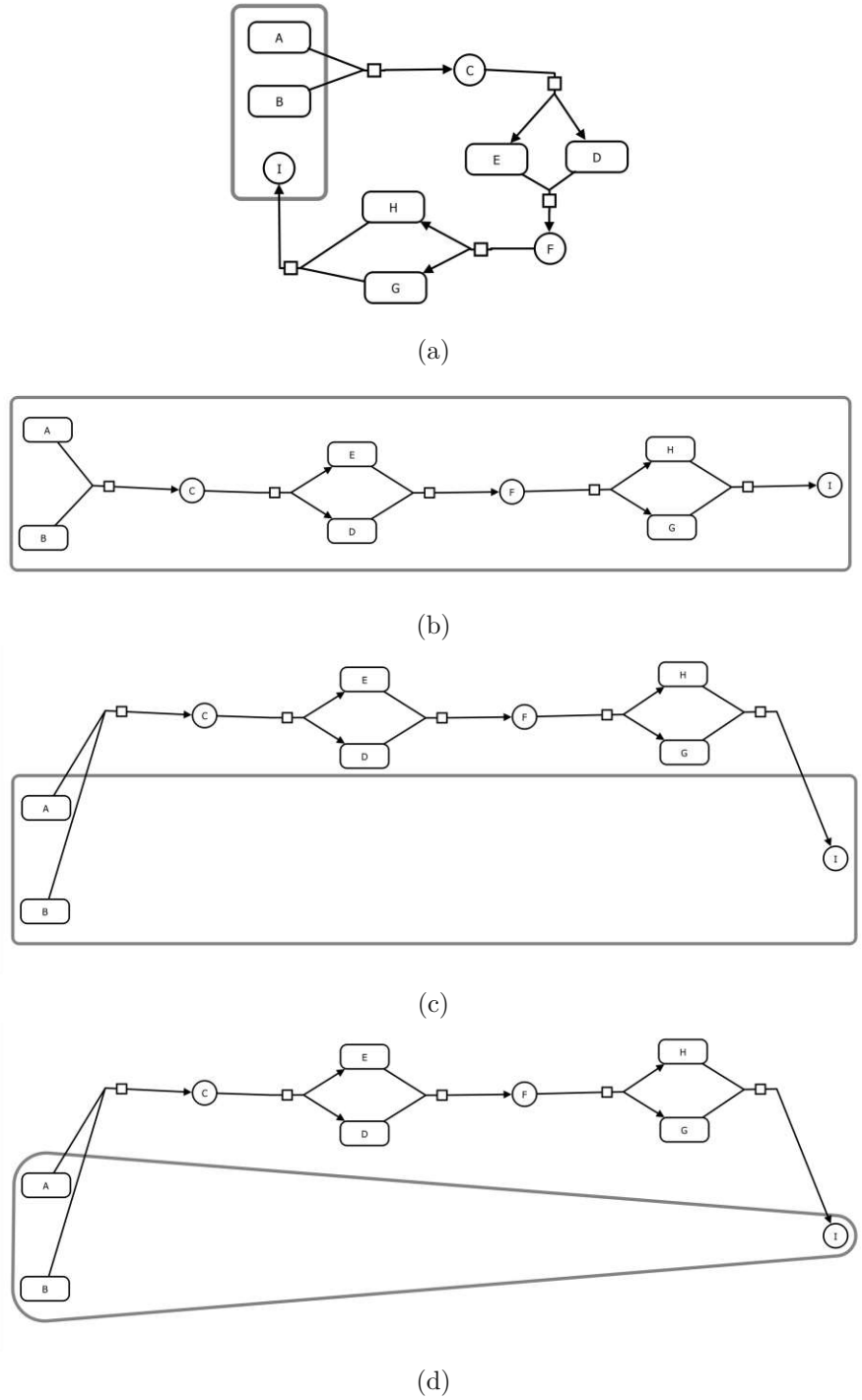
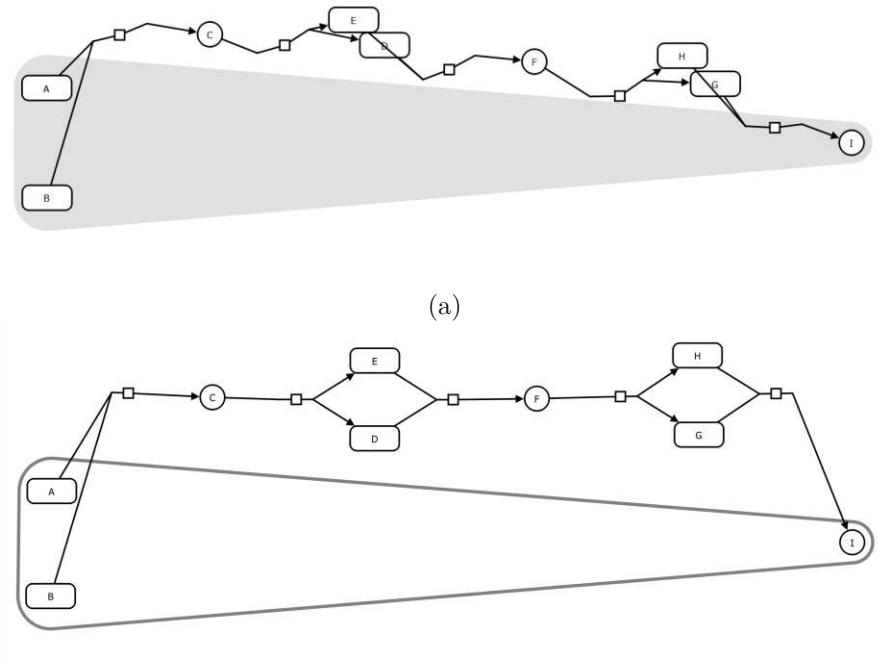


Figure 3.21: Applying an initial pivot MDS embedding to the graph of the manually-arranged drawing in (a), followed by an unconstrained stress layout, produces the drawing in (b). As the compartment affiliation of nodes is not depicted correctly, it is necessary to apply the compartment constraint. (c) shows the result of the compartment constraint, where the compartment is visualized as a box and (d) shows the same result as (c), but visualizes the compartment as convex hull.



(a)

(b)

Figure 3.22: Comparison between applying the compartment constraint on individual nodes and a cluster of nodes contained in a dummy compartment. (a) shows the result of the compartment constraint applied to individual nodes, the layout appears cluttered and the flow direction is obscured (real compartment shown as gray filled area). (b) shows the result of the compartment constraint applied on a whole dummy compartment, the flow direction of its nodes remains straight and the layout looks clear (real compartment shown as gray bounding box).

shown in Figure 3.22a. We exclude compartments with a parent-child relationship from the compartment constraint, because in such cases an overlap is intentional. Analogously to the direct non-node overlap constraint, a displacement of one compartment can cause an overlap with another compartment, which then demands a further refinement step. Since we use a compartment's convex boundary to remove overlaps, the compartment's visualization must also become a convex shape instead of a box, as shown in Figure 3.21d.

General Orthogonal Constraints

The general orthogonal constraint is designed to align all edges of the graph to be as orthogonal as possible. The constraint is inspired by the edge direction constraint of Wang et al. [103], which is defined as follows:

$$\mathbf{d}'_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\| \cdot \mathbf{u}_{ij}, \tag{3.13}$$

where $\mathbf{u}_{ij}$ is a direction vector in unit length, defined by a user or application [103]. We adapt this general definition to form a general orthogonal constraint. The direction $\mathbf{u}_{ij}$ is limited to adopt the directions of the x and y-axis, both in positive as well as negative direction, depending on the coordinate system:

$$\mathbf{u}_{ij} = \begin{cases} (1,0)^T & \text{right}, \\ (0,1)^T & \text{up}, \\ (-1,0)^T & \text{left}, \\ (0,-1)^T & \text{down}. \end{cases} \tag{3.14}$$

For every node pair i and j, their individual $\mathbf{u}_{ij}$ is automatically chosen by a heuristic, that uses the initial layout or a previous constraint step as input. For every input edge (i,j), we calculate its direction and determine its corresponding sector on the unit circle: $C_0 = 315° - 45°$, $C_1 = 45° - 135°$, $C_2 = 135° - 225°$, $C_3 = 225° - 315°$. The direction vector $\mathbf{u}_{ij}$ is then selected accordingly: for C_0 *right*, for C_1 *up*, for C_2 *left* and for C_3 *down*, as illustrated in Figure 3.23. This heuristic leads to an alignment of edge (i,j) that changes the previous layout as little as possible. $\mathbf{u}_{ij}$ is then scaled by the original distance between the nodes i and j, thereby defining the general orthogonal constraint $\mathbf{d}'_{ij}$. If applied to an edge, the edge is forced to align with $\mathbf{u}_{ij}$ while preserving its length. To generate an orthogonal graph this constraint is applied to every edge of the graph. For completeness, we have described this constraint, but we do not use it for any of our results. This constraint would result in orthogonal layouts that do not conform to the SBGN, as shown in Figure 3.19.

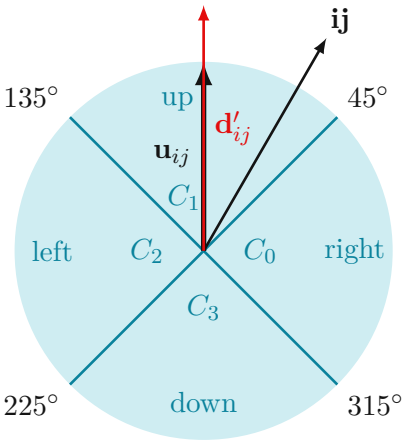


Figure 3.23: The direction vector $\mathbf{u}_{ij}$ of the orthogonal constraint is defined by considering the direction of the edge between the nodes i and j regarding its placement on a unit circle. Depending on the associated circle sector C_n with $n \in \{0,\ldots,3\}$, $\mathbf{u}_{ij}$ is chosen as right, up, left or down vector. The resulting vector $\mathbf{d}'_{ij}$ is axis-aligned and of length $\|\mathbf{x}_i - \mathbf{x}_j\|$.

Process Node Constraints

The SBGN defines the handles of a process glyph as a continuous straight line in horizontal or vertical direction. We represent process glyphs with three dummy nodes, as shown in Figure 3.24a. This allows the handles to move individually during the layout process, resulting in handles that are bent at the center of the process node, seen in Figure 3.24b, and, thereby, are not compliant with the layout rules of the SBGN.

To ensure compliance, a constraint is required. For this reason we introduce the *process orthogonal constraint* with the following two objectives:

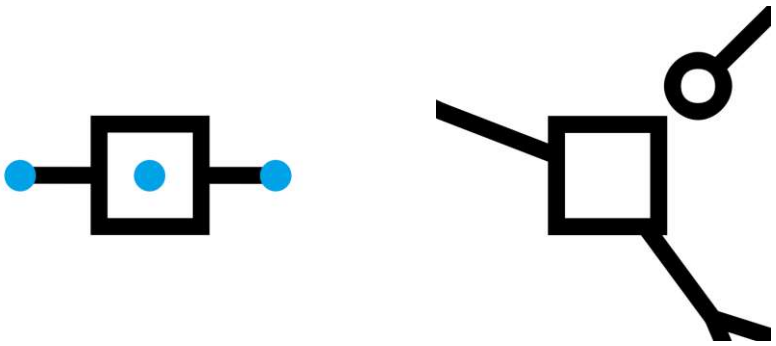
1. Both handles should be collinear, i.e., point in the same direction, to appear as one line.
2. Both handles should either be horizontally or vertically aligned.

We meet both objectives by adjusting the general orthogonal constraint to consider the three dummy nodes of process glyphs. The input for this constraint is the initial layout or the layout of a previous refinement step. For every process glyph we automatically select its three dummy vertices. In the following we refer to the node of the incoming handle as i, to the center node as j, and to the outgoing handle node as k. For these three vertices we define two constraints:

$$\mathbf{d}'_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\| \cdot \mathbf{u}_{ik}, \tag{3.15}$$

$$\mathbf{d}'_{jk} = \|\mathbf{x}_j - \mathbf{x}_k\| \cdot \mathbf{u}_{ik}. \tag{3.16}$$

To determine the process direction $\mathbf{u}_{ik}$ we calculate the handle direction from handle node i to handle node k. For this handle direction, we determine its unit circle sector



(a) (b)

Figure 3.24: The structure of a process node and its impact on the layout. (a) shows the SBGN-compliant appearance of a process glyph, with collinear handles on the left and right sides, and the blue dots representing the underlying dummy nodes. (b) shows the result of an unconstrained layout, with bent handles that are not compliant to the SBGN.

analogously to the general orthogonal constraint (Figure 3.23). Since the direction $\mathbf{u}_{ik}$ is shared by the constraints $\mathbf{d}'_{ij}$ and $\mathbf{d}'_{jk}$, imposing them on their associated edges (i, j) and (j, k) will force them into the same direction. Similar to the general orthogonal constraint, the edge length is not changed. Applying these two constraints on every process nodes' handles, aligns them in the same direction, either horizontally or vertically, while ensuring collinearity of the handles' edges. This way, both objectives are fulfilled.

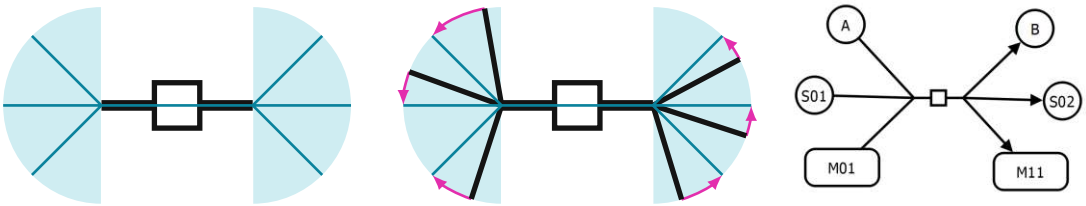
In addition to the layout of the process glyphs' handles, the SBGN defines how incoming and outgoing edges connect to the handles **(R19)**. The incoming edges represent consumption arcs and connect to one handle, the outgoing edges represent production arcs and connect to the other handle. This connectivity information is already defined through our data sets. The incoming and outgoing edges should be evenly distributed in the half-spaces before and after the process glyph in process direction. To align and distribute the edges properly, we use a shape-based constraint, referred to as *process angle constraint*. Wang et al. [103] formulate shape-based constraints as follows: A shape-based constraint defines a reference shape on which a local subgraph is fit. Once a local subgraph is selected by a user or application, the reference shape is discretized to fit the number of nodes of the selected subgraph. The nodes of the selected subgraph $\{\mathbf{p}_i\}$ are matched to the nodes of the discretized reference shape $\{\mathbf{q}_i\}$ using a modified version of the Iterative Closest Point (ICP) algorithm, which is widely used in the area of point set registration [103]. In every iteration of the ICP, the affine transformation $\mathbf{M}$ is calculated to minimize the following equation:

$$\sum \omega_i \|\mathbf{M}\mathbf{p}_i - \mathbf{q}_i\|^2. \tag{3.17}$$

The matrix $\mathbf{M}$ describes the minimal transformation of $\{\mathbf{p}_i\}$ to approximate $\{\mathbf{q}_i\}$. The weight ω_i is the vertex degree of node i at $\mathbf{p}_i$. The weight favors highly-connected nodes and highlights the reference shape within the layout. The matrix $\mathbf{M}$ is then used to define the target edge vector $\mathbf{d}'_{ij}$ and, therefore, defines the shape-based constraint:

$$\mathbf{d}'_{ij} = \mathbf{M}(\mathbf{p}_i - \mathbf{p}_j), \tag{3.18}$$

where pairs of $\mathbf{p}_i$ and $\mathbf{p}_j$ describe edges in the selected subgraph. By using the ICP approach for shape matching, the initial layout changes as little as possible if applying the shape-based constraint. We now formulate our process angle constraint on the basis of the star constraint of Wang et al. [103], which distributes edges evenly around a center node and generates star-like shapes. For our constraint we limit the edge distribution of incoming and outgoing edges to two separated semicircles relative to a reference direction, defined by the process handles. The two semicircles of the process angle constraint are visualized in Figure 3.25a in blue. For each process glyph we define an individual reference shape consisting of the nodes $\{\mathbf{q}_i\}$. It consists of the following vertices: three vertices on a horizontal line representing the three dummy vertices of a process glyph, for each incoming edge a vertex in equal distribution on the left semicircle of the reference shape, and for each outgoing edge a vertex in equal distribution on the right semicircle of the reference shape. An example reference shape for a process glyph with three incoming



(a) (b) (c)

Figure 3.25: The process angle constraint is defined for each process glyph individually to model the needed number of incoming and outgoing edges. (a) shows a reference shape for a process glyph with three incoming and three outgoing edges (dark blue). The edges are evenly distributed in there respective semi circle. (b) visualizes the edge fitting process between a process glyph and its reference shape. Each edge is fit to the nearest vertex in the reference shape and moved during the layout procedure accordingly, visualized as the magenta arrows. (c) presents a layout result after applying the constraint.

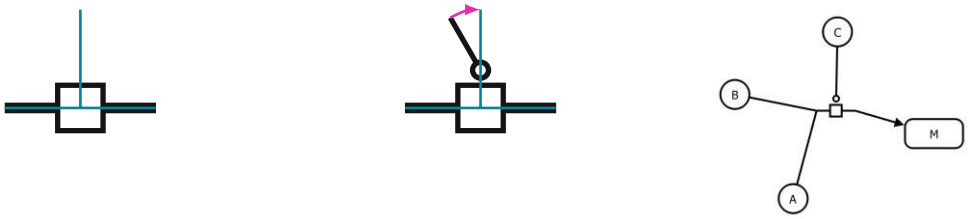
and three outgoing edges is shown in Figure 3.25a with dark blue lines. During the layout process the edges of a process glyph are fit to this reference shape by identifying the closest vertices of its reference shape and calculating the transformation **M**. Since **M** considers translation, rotation, and scale, our reference shape was defined in arbitrary orientation. The closest point matching process is visualized in Figure 3.25b. A small example of an SBGN map modified according to our process angle constraint is shown in Figure 3.25b. The three incoming and three outgoing edges are constrained to their respective semicircles and evenly distributed.

Alternatively to the ICP shape matching approach, the route and shape matching approach from Batik et al. [21] could have been used to match the process glyph to its reference shape. While the shape would have been more stable, because the approach is edge and routing aware, we would have lost the rotation invariance of the reference shape.

By combining the process orthogonal constraint and the process angle constraint, we can achieve SBGN compliance for process glyphs and their connections.

Modulatory Arc Constraint

So far, we described how to connect consumption and production arcs to process glyphs while complying with the SBGN. Additionally, modulatory arcs can be connected to process glyphs. They are connected to the center node of the process glyph and should be spread out in the semicircle perpendicular to the process handles while maximizing the minimal angle to other modulatory arcs **(R17)**. To satisfy these requirements we define a *modulatory arc constraint*. We apply a similar procedure as for the process angle constraint. Firstly, we define a reference shape consisting of the following vertices: three vertices on a horizontal line representing the three dummy nodes of a process glyph and one extra vertex placed perpendicularly to this line for the modulatory arc. The result is



(a) (b) (c)

Figure 3.26: The modulatory arc constraint reference shape is visualized in (a) as dark blue T-shape. It consists of four vertices, three for the process glyph and one for a modulatory arc. (b) shows an example modulatory arc being matched to its reference shape and moved during the layout procedure accordingly, visualized as the magenta arrow. In (c) we see the result of an applied modulatory arc constraint.

a T-shape, visualized in Figure 3.26a with dark blue lines. In contrast to the process angle constraint we do not create individual shapes for each process glyph. Although multiple modulatory arcs can be connected to a process glyph, we constrain every modulatory arc with the same shape, since we assume that already existing virtual edges will force them apart. The shape matching between a process glyph and its modulatory arc with the reference shape is visualized in Figure 3.26b. A result example of an applied modulatory arc constraint is shown in Figure 3.26c.

Constraints Combination

Selecting and combining a set of constraints is still an open problem [25, 65]. Not all constraints work well together, some work even against each other. The vectorized stress majorization enables the combination of constraints in a soft way by weighting the constraints, recall Equation 3.4. Defining favorable weights that highlight the preferred graph and layout properties is also still an open problem [86, 98].

The order in which the constraints are applied influences the result in our case. The reason for this is that some of our constraints use the current layout to calculate their metrics. Therefore, we define the constraints' order as follows:

1. Iterative Non-Node Overlap Constraint with weight v_{ij}^1
2. Compartment Constraint with weight v_{ij}^2
3. Process Orthogonal Constraint with weight v_{ij}^3
4. Process Angle Constraint with weight v_{ij}^4
5. Modulatory Arc Constraint with weight v_{ij}^5

The application-specific weights v_{ij} are user-defined. If we apply a new constraint, the previous constraints remain active to enforce them at the next refinement step. We choose

the weights according to $v_{ij}^1 < v_{ij}^2 < v_{ij}^3 < v_{ij}^4 < v_{ij}^5$, assuming that it becomes more difficult to enforce layout changes with an increasing number of active constraints. The behavior of these weights is demonstrated and evaluated in the results, see Chapter 4.

3.4.4 Multilevel Layout

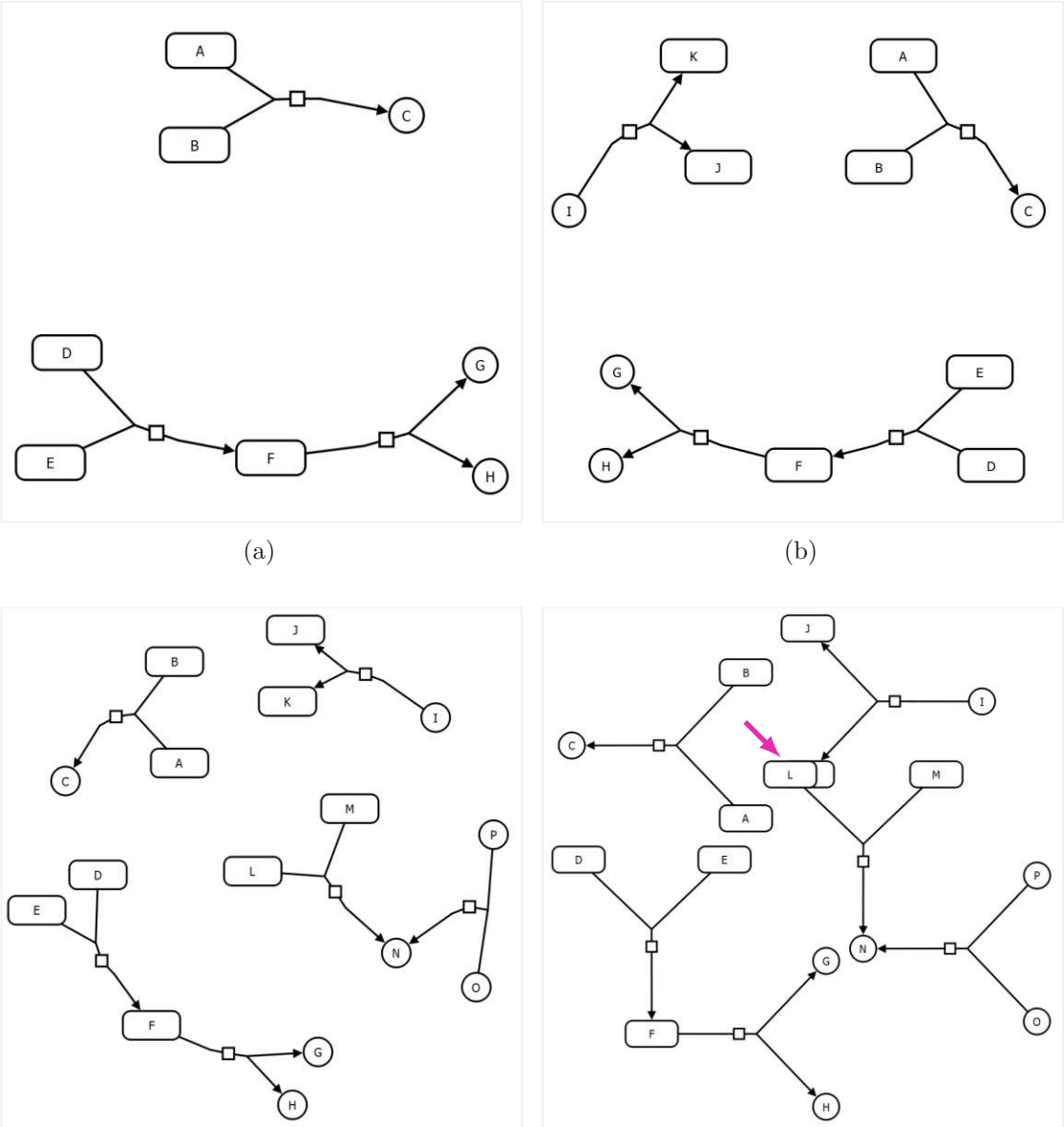
With the workflow consisting of an initial embedding, an initial layout, and the application of constraints, we can achieve layouts of metabolic networks. Such layouts emphasize symmetry, have a uniform process edge length, and a uniform edge length for production, consumption, and modulatory arcs. We can modulate these edge lengths to reduce the occurrence of hairballs around highly-connected nodes. We can reduce overlap of nodes and compartments and support the special layout of process glyphs with their connections. While this workflow allows us to produce reasonable results, we discovered two limitations **(C3)**:

1. In metabolic networks, not every entity is connected to other entities by a path, the underlying graph may rather consist of *disconnected components*. Such graphs are not supported in the layout procedure so far. The literature does not discuss this topic in the context of biological networks.

2. If a compartment consists of *disjoint subgraphs*, the compartment will not minimize its area.

The first limitation is caused by the choice of the techniques: the pivot MDS for the initial embedding and the vectorized stress majorization for the initial and constraint layout. Both techniques do not support graphs consisting of disconnected components [103]. To address this limitation we modify these approaches. We calculate the initial embedding with pivot MDS for each disconnected component individually, instead of calculating it for the whole graph. This results in an initial embedding where every component is placed at the origin, i.e., they are placed on top of each other. For our purpose this behavior is sufficient, since the subsequent initial layout takes responsibility for the component's arrangement. The vectorized stress majorization, does not support disconnected components since it uses the graph theoretical distance between two nodes to calculate d_{ij}. If two nodes are not connected, as is the case for two nodes in different components, the graph theoretical distance is defined as $+\infty$. But d_{ij} has to be a finite and displayable value. A rather straightforward solution is to connect the disconnected components with dummy edges to make the layout calculation possible. By selecting two nodes randomly and connecting them with an edge, we could introduce a wrongly perceived connection through the optical proximity relationship which is not present in the data set. This can cause a misinterpretation of the graph. Therefore, we decided on a redefinition of d_{ij} in the case that nodes i and j are in different components.

Depending on how d_{ij} is defined between two nodes in different components, the initial layout will arrange them differently. One option is to specify d_{ij} as a user-defined or



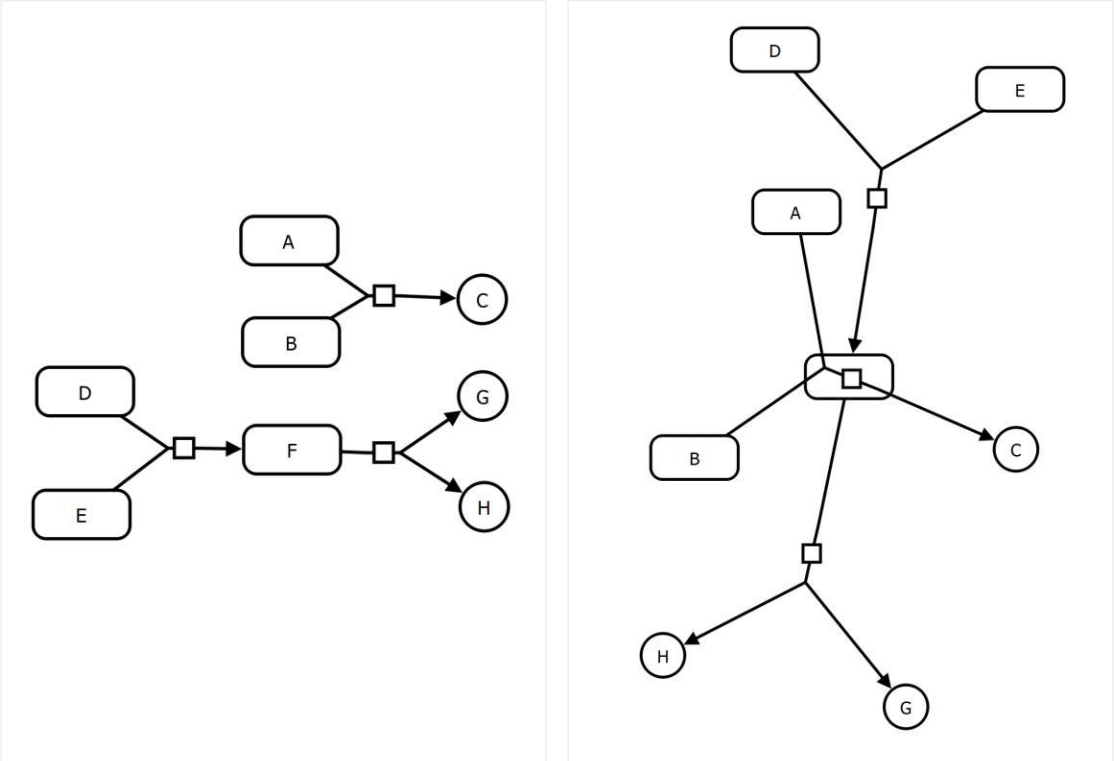
(a) (b)

(c) (d)

Figure 3.27: Redefining the graph theoretical distance between nodes of disconnected components as a multiple of the target edge length instead of $+\infty$ leads to a circular subgraph arrangement in the initial layout. The initial layout of two, three, and four disconnected components is shown in (a), (b), and (c), respectively. (d) shows the layout of four disconnected components with process constraints applied, which lead to overlaps, as indicated by the magenta arrow.

application-specific constant distance, e.g., a multiple of the target edge length. This sets the disconnected components into a relation to each other, while enforcing the selected distance between them. Setting d_{ij} to a fixed distance means that each node in one component is now connected to all nodes of the other component at a constant distance and vice versa. This leads to a highly interconnected graph which hinders the unfolding of the layout. This is demonstrated in Figure 3.27. The constant distance forces the components into a circular arrangement and introduces a curvature to the layout of each component. While our constraints can remove this curvature, as shown in Figure 3.27d, they can introduce a component overlap. This would make an additional non-component overlap constraint necessary.

Alternatively, we can redefine d_{ij} as the largest value a signed 32-bit integer field can hold, mimicking $+\infty$. This leads to a very small w_{ij}, thereby removing the relation between



(a) (b)

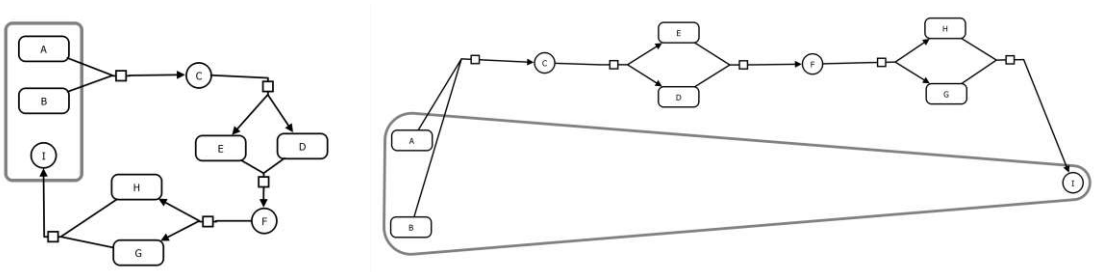
Figure 3.28: A manually-arranged graph with two disconnected components is shown in (a). The same graph is presented in (b) using the unconstrained vectorized stress majorization. The graph theoretical distance of disconnected nodes is set to the largest 32-bit signed integer value to mimic $+\infty$. Consequently, the disconnected components are untangled without exhibiting any influence between each other and are placed at the same position (origin). As a result, the two components are plotted above each other.

nodes without direct connection. This results in a separate layout for each component, all placed at the origin and stacked on top of each other, as seen in Figure 3.28. In contrast to setting d_{ij} to a constant distance, no curvature is introduced into the separate layouts. They are placed freely without external dependencies. To make the graph readable, we could again remove the overlap by an additional non-component overlap constraint, or by a subsequent arrangement step. An arrangement could be performed by a bin packing algorithm [27]. While both, setting d_{ij} to a constant distance and setting d_{ij} to a value mimicking $+\infty$ enables the vectorized stress majorization to run the layout computation, the results require further refinement.

The second limitation is caused by the inherent behavior of stress majorization, which distributes the nodes of a graph according to their graph-theoretical distance, together with its unawareness of compartment relations. Figure 3.29 shows nodes inside a compartment that are arranged according to their graph-theoretical distance. Since they are not connected with each other in their compartment, e.g., they are disjoint subgraphs, a large region of empty space is created between them. The requirement of minimizing the area (R12) is not fulfilled within compartments.

Connecting disjoint subgraphs with one dummy edge, or by connecting them with virtual edges of a constant distance, is a possible solution for this behavior. These solutions introduce the same problems as for disconnected components, i.e., either unwanted relations or circular arrangements. In the literature, only Schreiber et al. [90] discussed disjoint subgraphs in layout arrangements. They composed the subgraphs into one node to integrate them into one layout. The routing of edges into these assembled nodes is a difficult and unsolved problem.

We chose to further address both limitations, the disconnected components and the disjoint subgraphs, through a multilevel approach. The idea is to improve a coarse layout recursively into a detailed one, while traversing through a hierarchical graph [16], like



(a) (b)

Figure 3.29: The vectorized stress majorization does not inherently minimize the used area. It also has no inherent knowledge of compartment affiliation and does not place nodes that belong to the same compartment in proximity if they are not connected by an edge. (a) shows a compact layout made by hand. (b) shows the same graph after placing it with our approach including all constraints.

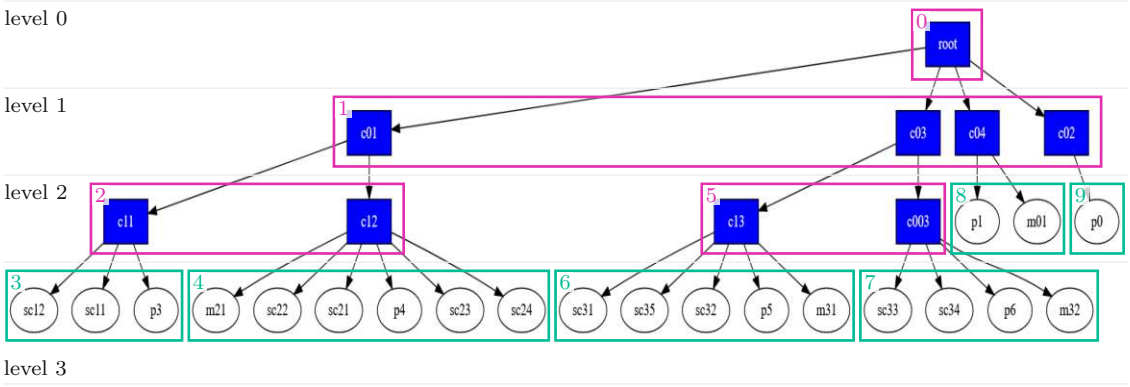


Figure 3.30: Illustration of the cluster tree traversal for the multilevel layout of Figure 3.31. The magenta rectangles depict cluster nodes and the green rectangle depict leaf nodes. The numbers in the top-left corners indicate the order of traversal (depth-first).

our clustered graph structure. We follow a top-down approach by starting the layout at the root level of the cluster tree and by finishing it at the tree's leaves. Firstly, we create a global reference layout by applying the initial embedding and the initial layout on the whole graph at the root node. The cluster tree's nodes are then visited through depth-first traversal. This is visualized in Figure 3.30 with the numbers in the top-left corners of the colored rectangles indicating the order of traversal. If a visited node is not a leaf, we choose between two layout strategies. Disjoint subgraphs are arranged either horizontally or vertically, while connected subgraphs are placed by unconstrained stress majorization. If a visited node is a leaf, we can apply our layout approach consisting of an initial embedding, initial layout, and a constraint layout. The global reference layout is now used as initial embedding for the leaf nodes. This influences the starting orientation of the local layouts of the leaves' subgraphs and facilitates a consistent integration into the final global layout. Before visiting the next node in the cluster tree, we apply the local layout changes to the global layout of the graph. To provide sufficient space when integrating the local layout into the global one, we use the bounding rectangle of the local layout to move its surrounding nodes aside. Figure 3.31 shows a result layout of our multilevel approach. The two disjoint subgraphs at the bottom of the SBGN map are arranged compactly next to each other. Alternative multilevel approaches [20] could also be integrated into our pipeline.

To summarize, in this section we discussed how we could extend pivot MDS and the vectorized stress majorization to support disconnected components in metabolic networks **(C1)**. We also presented a multilevel layout approach using our clustered graph data structure to handle disconnected components and disjoint subgraphs of a metabolic network **(C1)**. It improves the diagram's aspect ratio **(R11)** and minimizes the used area **(R12)**.

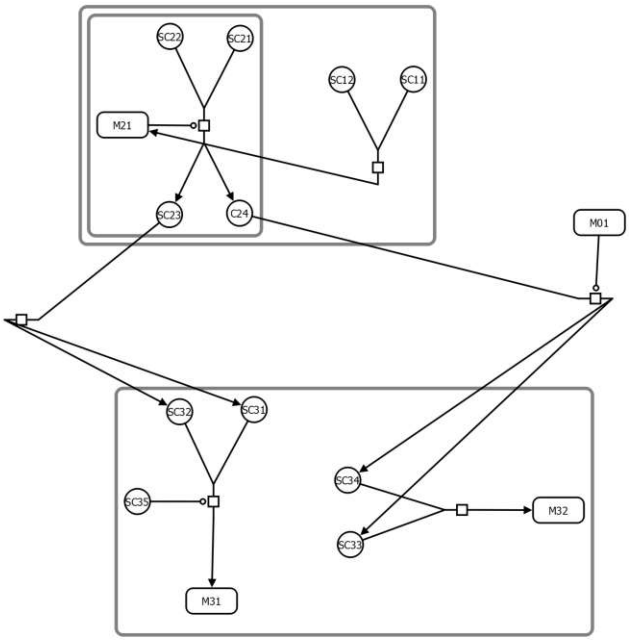
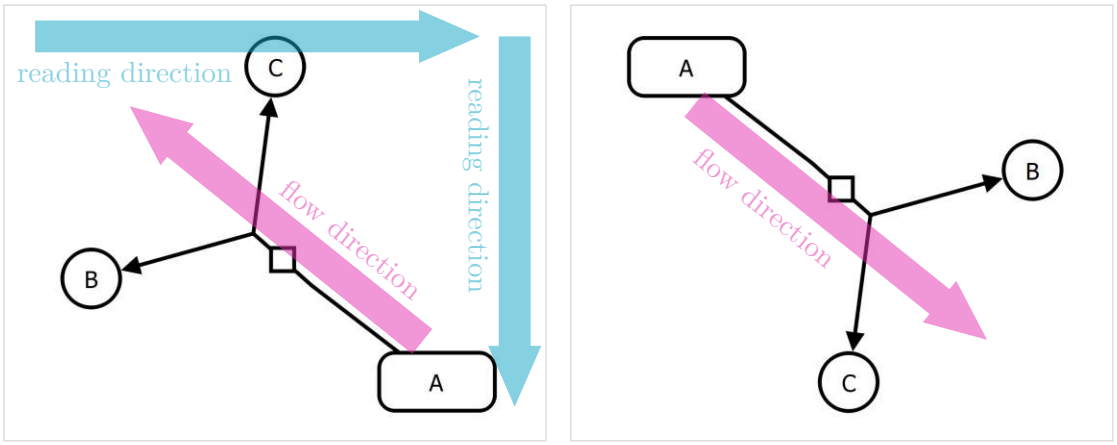


Figure 3.31: Result of out multilevel approach. The disjoint subgraphs in the lower compartment are moved together to minimize the used area in the compartment.

3.5 Postprocess

Although the preceding layout step has already considered the most important requirements, there is still room for improvement to the final drawing. These improvements can be accomplished as postprocessing operations. One aspect that can be improved upon is the alignment of the graph's flow direction **(R15)** with the reading direction. In western cultures the typical reading directions would be from left-to-right and top-to-bottom. The flow direction is not automatically aligned with the reading direction during the layout process. Therefore, we adjust it during postprocessing as follows. Firstly, we determine a flow direction $\mathbf{f}$ by selecting the visually most prominent arcs responsible for visualizing the direction, i.e., the production arcs drawn as arrows. We calculate the average direction of all production arcs, resulting in a vector representing the average flow direction of the graph. Then, we define the left-to-right reading direction as $\mathbf{r}_1 = (1, 0)^T$ and the top-to-bottom direction as $\mathbf{r}_2 = (0, -1)^T$, assuming the positive y-axis pointing upwards. To improve the alignment between flow and reading direction, the layout can be flipped horizontally and vertically. A vertical flip of graph G is done if $\hat{\mathbf{f}} \cdot \hat{\mathbf{r}}_1 < 0$. Analogously, the graph G is horizontally flipped if $\hat{\mathbf{f}} \cdot \hat{\mathbf{r}}_2 < 0$. This approach results in an approximated alignment of the flow direction with the reading direction. An example of this approach is shown in Figure 3.32. Alternatively, we could have rotated the layout from its flow direction to the vertical or horizontal axis, but then we would have lost the alignment of the process node handles to the axis, so we decided against rotation.



(a) (b)

Figure 3.32: The layout can be improved if the graph's overall flow direction is aligned with the reading direction. This is achieved in a postprocessing step. (a) shows an example biological process (without a previous layout) with a flow direction of $(-1, 1)$. (b) shows the biological process from (a) after applying a horizontal and vertical flip operation. The flow direction is now oriented in the same direction as the reading direction.

Another way to improve the layout, is to further remove edge crossings (R05) and edge-node overlaps (R10). Ideally nodes and edges should not cross. Even with the measures taken in the previous pipeline steps, crossings can still happen for highly connected graphs or big node sizes. This issue can be resolved in the postprocessing step through the use of edge routing algorithms. They usually resolve crossings by routing edges around nodes. This prevents nodes from being obscured by edges crossing on top of them, or misunderstandings about the connections between nodes and edges if edges cross under nodes [51]. The drawback of reducing crossings with edge routing is the introduction of edge bends into the layout. Since there are various existing approaches [42, 51, 60], we leave the integration of an existing algorithm into our pipeline as future work.

There are many possibilities to improve the layout with postprocessing. For example Siebenhaller et al. [92] proposed to flip subgraphs out of enclosed graph parts like circles, but discussing all of these possibilities is out of scope of this work.

3.6 Visualize Compound Structures

Entities of a biological network and their reactions are located in nested structures of a biological system. A metabolite lives in a cell, or more precisely in a sub-compartment of a cell, called the subcellular location. Examples are the cytosol, the nucleus, or the chloroplasts in plants. A metabolite's location could be described even more accurately by providing its affiliation to molecular complexes. This information yields additional

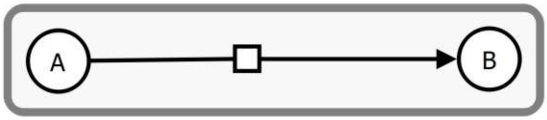


Figure 3.33: An example process encapsulated by a compartment that is visualized as a bounding box.

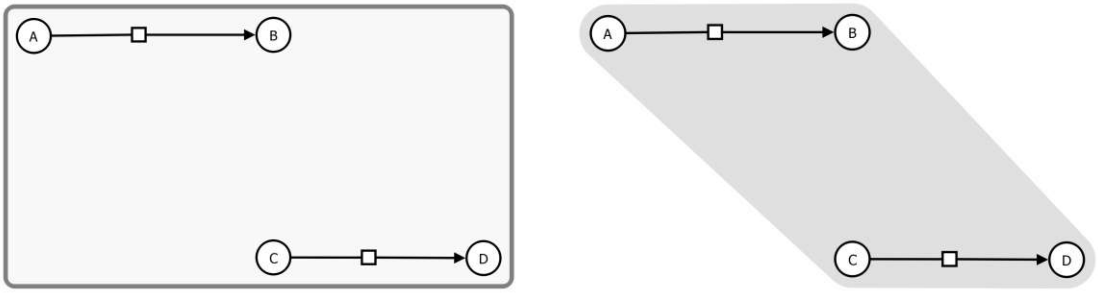
insight into the reactions, because metabolites can only interact directly with others in the same compartment. Therefore, it is beneficial to integrate an entity's location into the layout process [14, 45] and to visualize the nested structure.

The layout process aims to move entities of the same compartment into proximity of each other. Entities that are not part of the compartment, should be kept at a distance from it. Compartments can be visually represented by boundaries that should only encapsulate affiliated entities.

Barsky et al. [19] invented *Cerebral*, where entities of the same subcellular location are placed on bands of *cell slices*. Their approach does not appear to be compatible with the data described in the SBGN, as the cell hierarchy is represented in a tip-over convention instead of the inclusion convention used by the SBGN. The methods of Schreiber et al. [91] and Dwyer et al. [52] use a separation constraint to move compartments apart, and therefore, prevent subcellular locations from overlapping. Their approaches only consider rectangular bounding boxes of compartments, potentially resulting in significant empty space between nodes (depending on the network). They do not consider compartment nesting. Our layout approach, recall Section 3.4, is designed for layouts in inclusion convention and is able to handle convex compartment bounds, overlaps, and disconnected components. To visually enclose compartments, we have deliberately decided to develop our own approach, which is introduced in this section **(C2)**. This way, we are more flexible and can design custom representations **(R18)**. Ultimately, we aim to generate distinguishable shapes, i.e., motifs, for compartments to facilitate the mental map during browsing biological networks.

After the layout process is finished, there are different options to visualize a compartment's cluster information. Compartments are usually visualized by drawing their bounding box, shown in Figure 3.33. While the bounding box is suggested as glyph for compartments in the SBGN, alternative encapsulations are also valid [84].

Our layout process considers the non-overlap constraint for compartments **(R14)** to avoid placing nodes and edges in compartments where they do not belong. For nested subgraphs with disjoint subgraphs this can potentially result in drawings with compartment bounding boxes enclosing large empty areas, long edges connecting to outside nodes, and with an overall sparse as well as scattered looking appearance. An example is shown in Figure 3.34a. The usage of screen space could be improved by enclosing the compartments with a convex hull instead of rectangles, shown in Figure 3.34b. The convex hull is calculated from points sampled along the shape of each node. Convex



(a) (b)

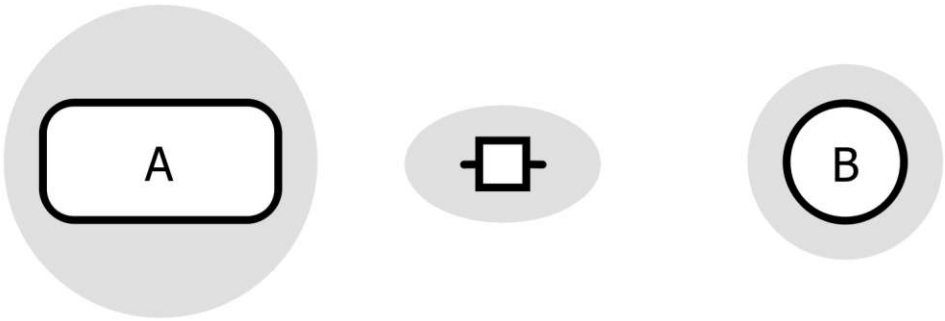
Figure 3.34: Comparison of compartment visualizations of two disconnected processes within one compartment with a layout that does not optimize the area. (a) displays the encapsulation of both processes with a standard bounding box. (b) depicts the encapsulation of both processes in the convex hull. Both compartment visualizations encapsulate a large empty area additionally to the contained processes, (b) less than (a).

hulls can also be used during the layout process. Nevertheless, neither the bounding box nor the convex hull create distinct figures [28], that would aid the identification and recognition of compartments.

There are other options to visualize cluster relations in graphs, such as the area-based GMap [62], the contour-based approaches Line Set [15] as well as Bubble Set [37]. While all of them can handle spatially dispersed clusters, GMap cannot handle cluster overlaps. Line Set and Bubble Set do not handle overlaps appropriately for our application domain. Also none of them consider edges to be part of the clusters, but this is a requirement of the SBGN. Since Line Set and Bubble Set visualizations lead to more accuracy in cluster identification tasks compared to GMap [67], and, perceiving shapes through boundaries is faster than using areas [55], we designed a new boundary-based approach to visualize compartments.

Boundaries communicate shape information, and a boundary is perceived by processing local convexities and concavities. Convexities are perceived as parts, nodes in our case, and concavities are perceived as boundaries between them, edges in our case. Concavities also improve the speed of visual target search [28]. For this reason, we further improve the compartment boundary by introducing concavities. For performance reasons, concavities are omitted during the layout process.

Subsequently, we describe our approach for creating compartment boundaries. We enclose nodes with circles whose diameter is the longer side of the node. Process nodes are enclosed by ellipses, which are placed at the center of a process node and are wide enough to contain its two handles. These shapes are sampled and the resulting points are added to the set of hull points. By computing the concave hull [2, 85] of these points, we obtain the compartment boundary. Figure 3.35 shows different compartment boundaries (filled gray shapes) containing a single entity. The shape of the concave hull using only the



(a) (b) (c)

Figure 3.35: Example entities encapsulated by gray circular and elliptic shapes representing compartments. (a) shows a macromolecule encapsulated by a circular shape that depicts the compartment boundary. (b) displays a process node enclosed by an elliptic shape. (c) visualizes a simple chemical encapsulated by a circular shape representing a compartment.

nodes' hull points does not always contain all the edges of the compartment, and its shape might be indistinguishable from other compartments. To prevent this, we include edges, whose both nodes are contained in the compartment, in the boundary calculation.

Edges need to be enclosed in the compartment boundary because they are part of chemical reactions, which take place inside the compartment. For this reason, we create shapes for the edges and include these shapes in the calculation of the compartment's concave hull. Although the vectorized stress majorization aims for a layout with a uniform target edge length, this might not always be feasible due to the applied constraints. This could lead to shorter as well as longer edges. Since an edge depicts the relation between two nodes, its length conveys the strength of this relation. Nodes connected with a longer edge might be perceived as less related to each other than nodes connected with a shorter edge. To compensate for a perceived weak relation between two distant nodes, we introduce an edge shape that follows the visual metaphor of an elastic band. If the distance between two nodes, the *real edge length*, is less than or equal to the desired *target edge length*, the elastic band is under no *tension*. If the real edge length exceeds the target edge length, the tension of the elastic band successively increases, as illustrated in Figure 3.36. We define this tension as follows:

$$tension = 1 - \text{clamp}(\frac{targetEdgeLength}{realEdgeLength}, 0, 1). \tag{3.19}$$

The tension becomes zero if the real edge length is shorter or equal to the target edge length. It approximates one if the real edge length is longer than the target edge length. We map this tension to the shape of the edge to visually emphasizes the actual relation between its two nodes. This is accomplished by first introducing a mapping function:

$$\lambda(t) = t^2 \cdot tension + 1 - tension, \tag{3.20}$$

with $t \in [-1, 1]$ moving along the edge, i.e., $t = -1$ at the start of the edge and $t = 1$ at the end. So far, $\lambda(t)$ only models the tension of the edge, but does not consider the shapes of its two nodes. By scaling $\lambda(t)$ with the linearly interpolated radii of the edge's source and target nodes, we compute the thickness of the edge's shape:

$$\text{thickness}(s) = \lambda(2s - 1)((1 - s) \cdot r_{source} + s \cdot r_{target}), \tag{3.21}$$

with $s \in [0, 1]$ moving along the edge, r_{source} being the radius of the source node, and r_{target} being the radius of the target node. The thickness describes how the shape of the edge behaves under tension and, therefore, mimics the behavior of an elastic band. Figure 3.36 illustrates the influence of the tension on the shape of an edge with different lengths.

Under high tension, the edge's thickness (blue lines in Figure 3.36) can be smaller than its source or target nodes' shapes (gray circles in Figure 3.36). This is demonstrated in Figure 3.36b and Figure 3.36c. To prevent discontinuities when combining the edge's thickness with the nodes' shapes (see zoom-in of Figure 3.36b), we smoothly blend them along the edge using the smooth maximum function (of degree two) for modeling implicit geometric shapes [78]. Therefore, we model the source and target nodes' shapes in terms of distances along the edge, $b_{source}(s)$ and $b_{target}(s)$, respectively. Now we can blend the shapes of the two nodes with the edge. This is achieved by, firstly, blending the shapes of the two nodes, $\text{blend}_1(s, \delta)$, and, subsequently, blending the result with the edge's thickness, $\text{blend}_2(s, \delta)$, as follows:

$$\text{blend}_1(s, \delta) = \frac{1}{2} \cdot (b_{source}(s) + b_{target}(s) + |b_{source}(s) - b_{target}(s)|_{2,\delta}), \tag{3.22}$$

$$\text{blend}_2(s, \delta) = \frac{1}{2} \cdot (\text{thickness}(s) + \text{blend}_1(s, \delta) + |\text{thickness}(t) - \text{blend}_1(s, \delta)|_{2,\delta}), \tag{3.23}$$

where $\delta = 10$ defines the blending range and $|\cdot|_{2,\delta}$ represents the smooth absolute function [78]. We calculate $\text{blend}_2(s, \delta)$ on both sides of the edge to create the final edge shape, depicted by the magenta lines in Figure 3.36. The hull points of the edge and its two nodes are computed by sampling the shape of the edge and the shapes of its source and target nodes (dark gray lines in Figure 3.36). To finally obtain the compartment's shape, we calculate the concave hull [85] of these points. The compartment's shape now contains the edge as well. Figure 3.37 shows three processes with production arcs of different lengths and, therefore, different tension. The processes are contained in compartments that are displayed with our so-called *elastic band visualization* **(C2)**.

If a compartment contains another compartment, the parent compartment's hull is computed as follows. Instead of using the concave hull of the child compartment to calculate the parent's hull, we calculate the child's convex hull. To create a margin between the child compartment and the parent compartment, we dilate the convex hull of the child by a user-defined number of pixels (10 px in our experiments). This margin emphasizes the inclusion of the child in the parent compartment. The child's dilated hull is then used together with the hull points of the parent's nodes and edges to calculate

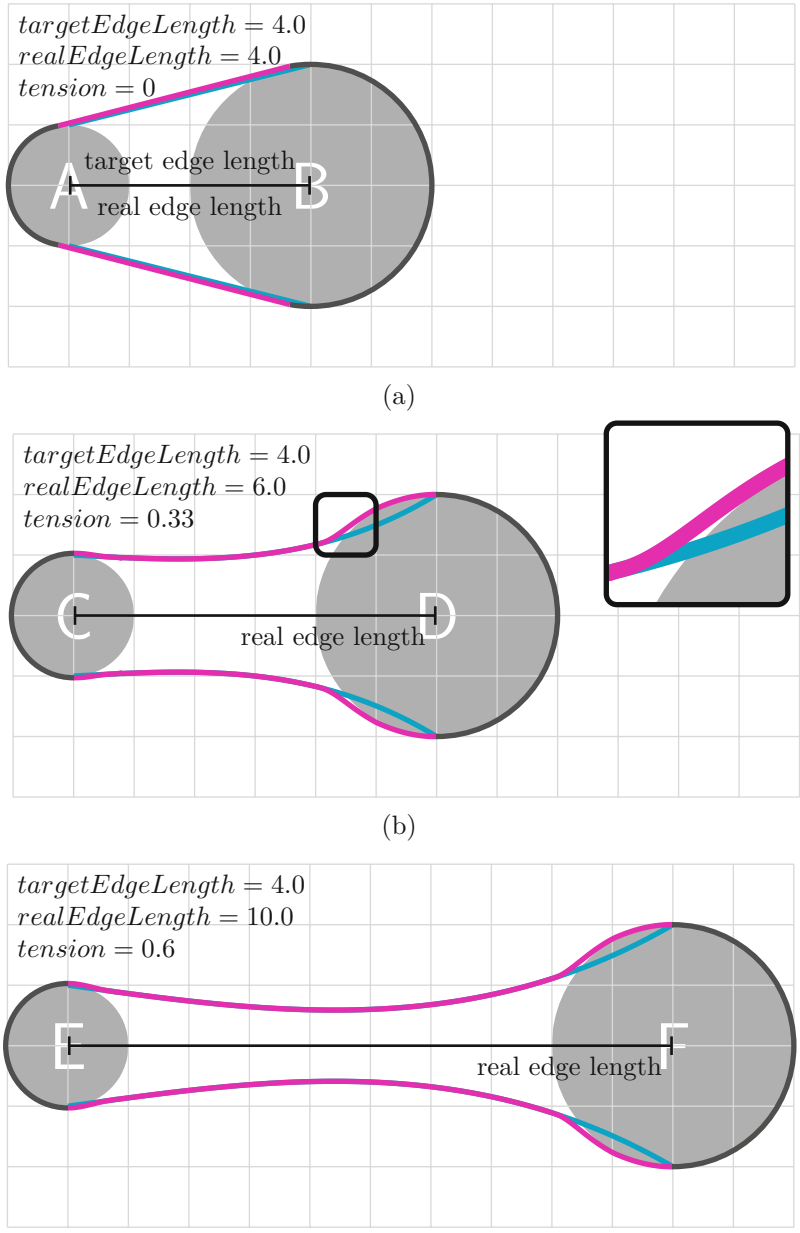


Figure 3.36: Illustration of our elastic band visualization approach. Figures (a), (b), and (c) demonstrate the elastic band visualization under different tension. This tension is introduced if the real edge length exceeds the layout's target edge length. The gray circles depict the shapes of the nodes A to F. The blue lines depict the thickness of the edge's shape. The magenta lines show the result when smoothly blending the nodes's shapes with the edge's thickness, see zoom-in in (b). The magenta lines and the dark gray lines (node shape boundaries) are then sampled to obtain the concave hull points. These points are used to calculate the concave hull, which is the final shape of the compartment boundary and is referred to as elastic band visualization.

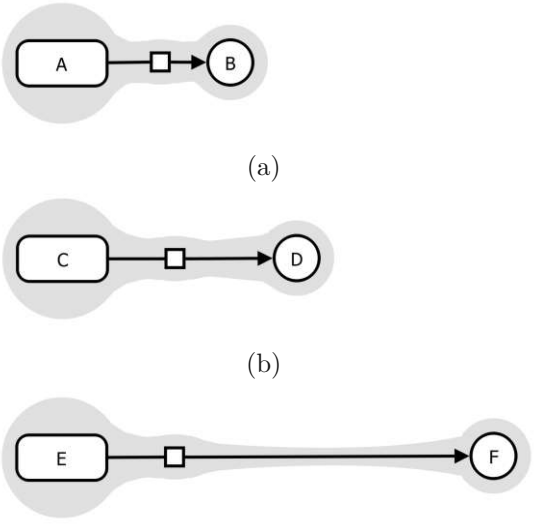


Figure 3.37: Processes contained in three compartments (gray filled shapes) visualized with our elastic band technique. The production arcs of the processes have different lengths, resulting in their shapes exhibiting different tension. Tension is caused by a deviation between the length of the production arc and the target edge length, which is enforced by the stress majorization during the layout process. (a) shows a production arc with a length smaller than the target edge length, resulting in $tension = 0$. In (b) the length of the production arc equals the target edge length, which still leads to $tension = 0$. If the length of the production arc starts to exceed the target edge length, the tension of the edge increases. (c) shows a production arc with $tension = 0.65$.

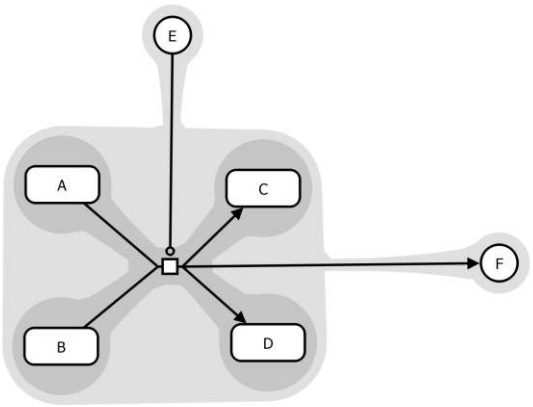


Figure 3.38: Visualization of a child compartment in a parent compartment. The child compartment consists of nodes A, B, C, D, and one process node. The parent compartment includes all nodes of the child compartment and, additionally, nodes E and F. The child compartment's shape (dark gray) is calculated from its concave hull. The parent's shape (light gray) consists of the child's dilated convex hull together with the concave hulls of the nodes E and F as well as their edges.

the hull of the parent compartment, as shown in Figure 3.38. This is a design choice made to reduce visual clutter, since visual clutter hinders data reading [67].

A compartment can contain several independent reactions, whose only relationship is that they occur in the same part of the cell. This leads to disjoint subgraphs or disconnected components inside a compartment. Since the concave hull algorithm [85] is not designed to create a joint hull of disconnected objects, applying it to the hull points of disjoint subgraphs or disconnected components would lead to artifacts. Such artifacts are, e.g., thin connecting lines between components, or large connected areas (depending on node placement). To solve this problem, we connect the disconnected components with new edges between their nodes. To model the strong relationship between disconnected components that belong to the same compartment, the shortest edges are chosen. Since these edges do not exist in the graph, we refer to them as *dummy edges*. This results in a complete graph of components connected with dummy edges. To simplify this graph, we compute the minimum spanning tree using Kruskal's algorithm with the lengths of the dummy edges as weights [74]. The remaining dummy edges of the minimum spanning tree define the connections between the disconnected components. This procedure is shown in Figure 3.39. Whenever the layout changes, the dummy edges and the minimum spanning tree are recalculated. This ensures that the components are always connected to their nearest neighbor. The dummy edges are then included in the calculation of the compartments boundary.

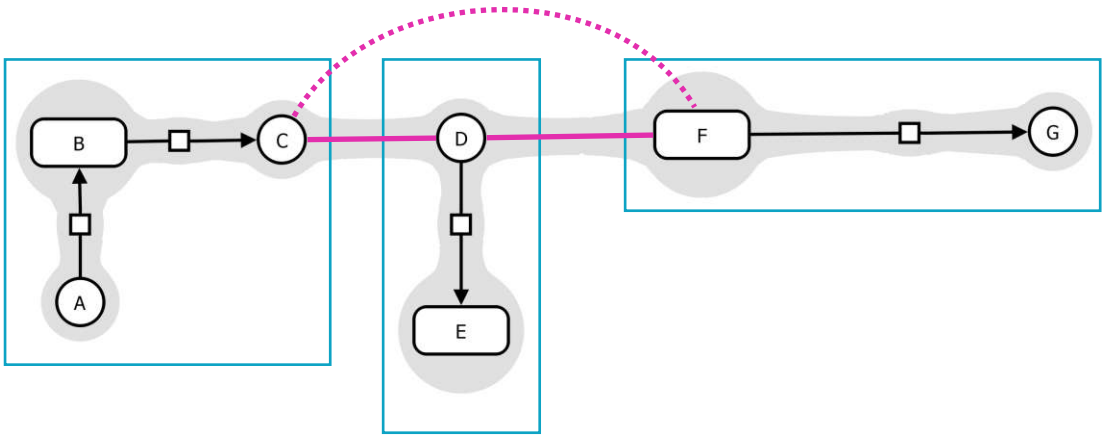
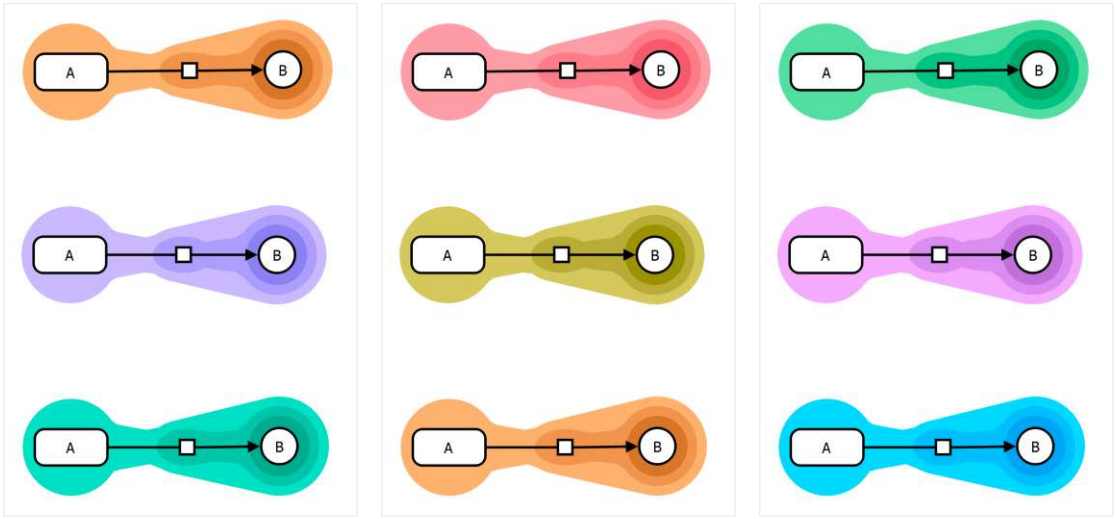


Figure 3.39: Visualization of a compartment (gray) consisting of three disconnected components (blue rectangles). Each pair of components is connected with the shortest dummy edge (magenta). The dotted dummy edge (bent only for illustration purposes) is removed by a subsequent minimum spanning tree operation. As a result, nodes C and D as well as nodes D and F are connected by dummy edges. These dummy edges are then used in our elastic band visualization to visually connect the disconnected components in a compartment.

In summary, compartments can now be visualized in the following way. The elastic band visualization includes edges in the compartment's shape while encoding node proximity. Nested compartments are emphasized by using a different boundary shape of their children. Disconnected components and disjoint subgraphs in the same compartment are visually connected by dummy edges. The combination of these facets leads to distinguishable shapes of compartments composed of concavities and convexities.

Since colors support cluster distinction [67], we decided to give each subcellular location a distinct color **(C2)**. The selected colors need to be of sufficient distance to each other to avoid misinterpretation of clusters [67]. We decided to generate the colors following the *tree colors* approach [99], which creates color schemes for tree structured data. The color scheme is calculated in the Hue-Chroma-Luminance (HCL) color space. This has the advantage of generating colors of different hues in perceptually uniform colorfulness and brightness [99]. This means that colors would convert to the same grayscale value, if they only differ in hue but not in chroma and luminance [108]. In the tree color approach, a color's hue depends on the number of clusters at a tree level, while chroma and luminance depend on the tree's depth. As a result, the colors of clusters on the same tree level are perceptually uniform in chroma and luminance, while being readily distinguishable from colors on other levels [99].



(a) (b) (c)

Figure 3.40: Example SBGN map with three disconnected compartments, each of which contains child compartments with a maximum tree level of 2. The SBGN map is colored with three different tree color schemes, specified through the following settings. The chroma and luminance values are calculated in all color schemes with the user-defined values of $(C_1, \beta) = (60, 5)$ for chroma, and $(L_1, \gamma) = (70, -10)$ for luminance. The hue range is user-defined, with the range $[0, 360]$ in (a), with the range $[0, 120]$ in (b), and with the range $[120, 360]$ in (c). A fraction of 0.75 is chosen in all color schemes.

The color assignment is done as follows. We start at the tree's root with a user-defined hue range, e.g., $[0, 360]$, and traverse the tree in breadth-first order. The hue range is divided by the number of nodes on the current tree level, creating hue sub-ranges for each node. The center of a sub-range is then assigned as hue to the associated node. The hue range of nodes in the subsequent tree level is calculated from a node's parent range. To ensure that the nodes on the subsequent tree levels are assigned hues of sufficient distance to each other, the hue sub-ranges are trimmed at the start and end by a user-defined fraction factor in the range of $[0, 1]$. This procedure is continued until every node of the tree has a hue value assigned. The chroma C and luminance L value for every tree level i are calculated as follows:

$$C_i = (i-1)\beta^C + C_1, \tag{3.24}$$

$$L_i = (i-1)\gamma^L + L_1, \tag{3.25}$$

where C_1 and L_1 are user-defined values for the first tree level. The user-defined slope parameters β and γ influence the change in chroma and luminance per tree level i, respectively. The resulting color in HCL space is then transformed to RGB space for further usage. We apply the tree color approach on the cluster tree of our data structure, but exclude dummy compartments since they are not visualized. In doing so, we assign a color to each compartment, which is derived from its parent's color, while being distinguishable in color from its sibling compartments. The hue range as well as the fraction factor can be specified by the user. Figure 3.40 shows tree color schemes defined

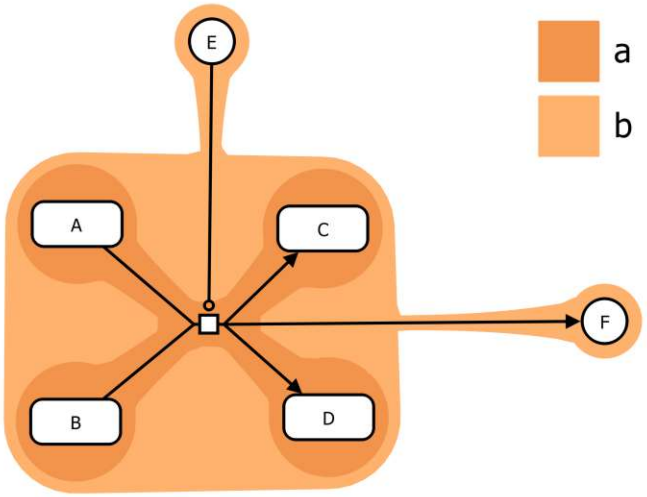


Figure 3.41: Colored visualization of a compartment in a compartment. It is colored with a tree color scheme defined as follows: The hue range is $[0, 120]$ with fraction 0.75. The chroma and luminance values are calculated from $(C_1, \beta) = (60, 5)$ and $(L_1, \gamma) = (70, -10)$, respectively. Since the parent compartment has only one child, both have the same hue value. Since the child compartment is on a tree level below the parent compartment, its chroma and luminance values differ from those of the parent.

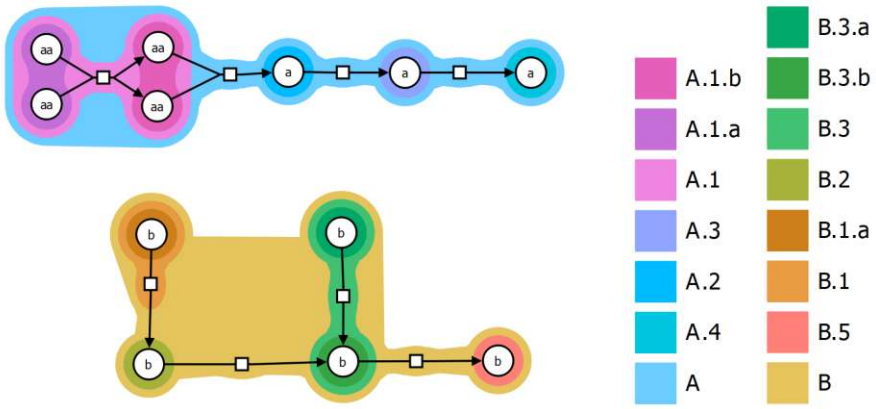


Figure 3.42: Colored visualization of two disjoint compartments A (light blue) and B (ocher). Both contain multiple nested child compartments. The color scheme has a hue range of $[0, 360]$ with fraction 0.75. The chroma value is calculated from $(C_1, \beta) = (60, 5)$ and the luminance from $(L_1, \gamma) = (60, 10)$. The compartment's hues vary based on the underlying cluster tree structure. The chroma and luminance values of the compartments vary based on the compartment's associated tree level.

by different hue ranges. Figure 3.41 demonstrates the variation in chroma and luminance values and Figure 3.42 shows the hue variations of an example data set with multiple nested and disjoint compartments. Additionally, we provide a color legend that shows a compartment's color besides its name.

3.7 Reduce Complexity

Graphs with hundreds or thousands of nodes are often too large to be displayed and too complex to comprehend. Exploring and interacting with such large graphs is also ineffective [13]. We address this problem by combining expand and collapse operations with motif simplification to reduce the complexity of large metabolic pathways **(C2)** [106].

The cluster tree structure (recall Section 3.3) is the base of the expand and collapse operations. This tree provides the cluster information and the graph's hierarchy, which are used to combine subgraphs of different cluster levels into one drawing [13]. Going up in the cluster tree successively abstracts, i.e., reduces compartments to motifs. Going down progressively displays more details, i.e., the motifs are enlarged to their compartments in order to reveal the compartments' content.

If a compartment is selected by a user, a collapse operation can be performed on the corresponding node in the cluster tree. To convey the visual metaphor of a collapse, we first hide all vertices and edges contained in the node's subgraph and then reduce the size of the remaining compartment's shape. If a collapsed compartment is contained in a parent compartment, the collapse operation does not influence the shape of the parent.

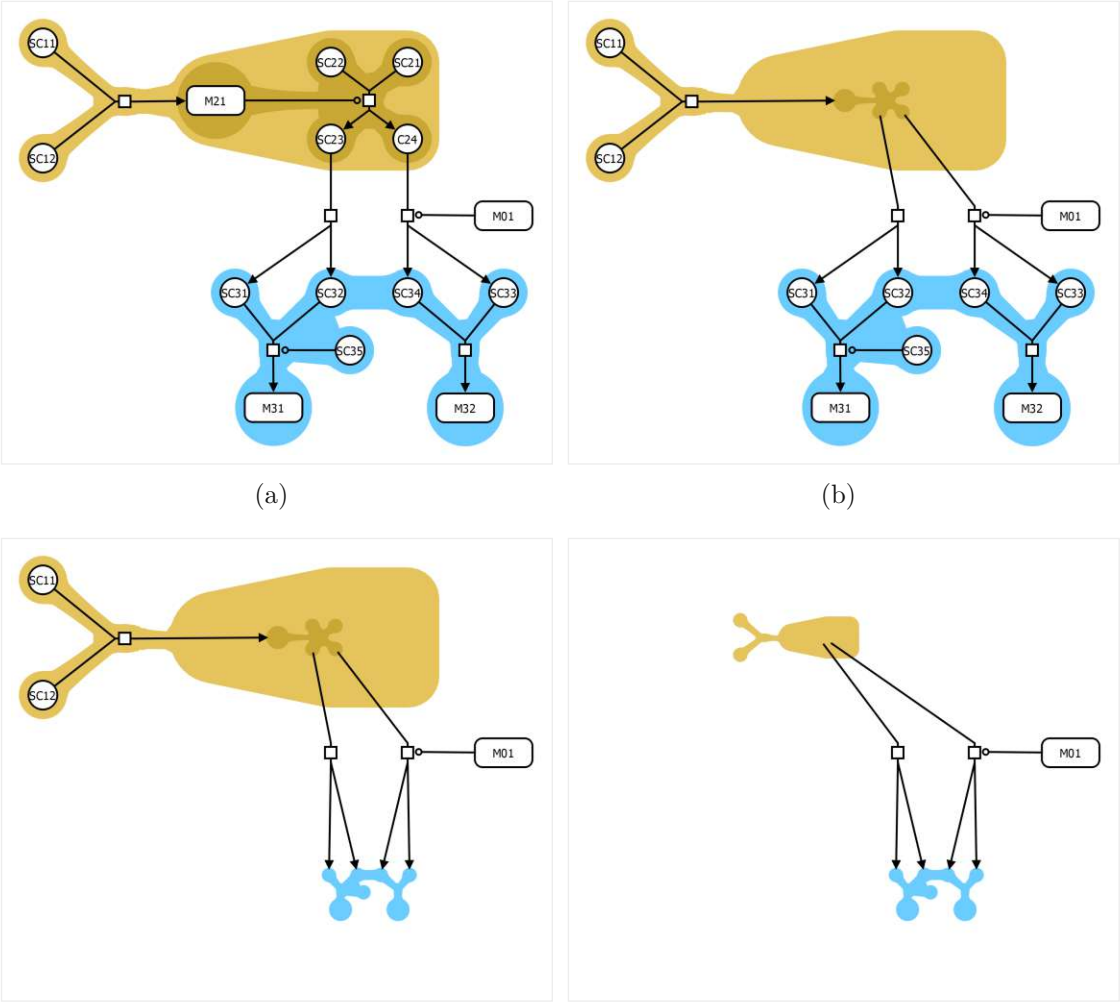


Figure 3.43: Complexity reduction using expand and collapse. A compartment is collapsed by downsizing it and displaying only its shape as glyph. This is analogous to motif simplification, where recurring parts of a graph are replaced by a glyph. (a) shows the fully expanded example. In (b) the innermost compartment is collapsed. Note, how its shape is scaled down while its parent compartment's shape remains unchanged. In (c) we continue collapsing until the drawing is fully collapsed in (d).

By substituting the subgraph with its downsized compartment shape, it is abstracted into a glyph of its structure [47]. Replacing parts of a graph with glyphs is analogous to *motif simplification* [47]. Motivated by the work of Gardner et al. [63], who used such motifs as illustrative representations of molecules and cell membranes, we use our elastic band representation of compartments as motifs. This collapse operation reduces the complexity of the graph and increases readability [47]. Figure 3.43 demonstrates multiple collapse operations.

Dogrusoz et al. [45] implemented the collapse operation by replacing the subgraph of the collapsed compartment with a new node. Edges connected to nodes in the collapsed subgraph are replaced by so-called meta edges that are connected to the newly introduced node. This changes the topology of the graph. However, the newly introduced node could have a high-degree, which would move the node to the center of the layout in a subsequent layout step. This could lead to a hairball and the resulting change in layout affects the mental map. To avoid this problem, we do not change the graph's topology. Instead, we internally only downsize the collapsed subgraph, i.e., reducing its edges' lengths and nodes' sizes.

The expand operation is inverse to the collapse operation. The compartment shape is scaled to its original size and contained nodes and edges are displayed again. This allows users to view details of a metabolic pathway on demand.

To summarize, our expand and collapse interaction preserves the mental map **(R21)**, while reducing the complexity of metabolic networks. The motif simplification supports the user to recognize biological structures through the distinguishable shapes created by our elastic band visualization.

3.8 Implementation

We implemented a prototype application to demonstrate our layout pipeline with the programming language C++, the parallel computing platform CUDA (v11.6), and the build tool CMake. The prototype was developed and tested on a system consisting of an Intel Core i7-9750H CPU @ 2.6 GHz with 32 GB RAM, an Nvidia RTX 2070 Max-Q GPU with 8 GB VRAM, and Windows 11 as operating system.

We built the prototype on a set of freely available software libraries, frameworks, and algorithms. They are described in the following:

- **Qt** (v5.15): is a modular cross-platform framework for application development. We use the Core, GUI, and OpenGL module. While Qt is proprietary, it is free for academic usage.
- **LibSBGN** (v2): provides us with the means to load and save data in the SBGN-ML format [3, 39].

- **boost graph library** (v17.3): provides a collection of C++ libraries, with the purpose of creating standardized tools for recurring operations, such as graph traversal, minimum spanning tree, and shortest path.
- **Open Graph Drawing Framework (OGDF)** (v2020.02): contains data structures and open graph algorithms. We adapted OGDF's pivot MDS implementation.
- **Eigen** (v3.3): is a linear algebra library for matrices, vectors, and solvers.
- **Computational Geometry Algorithms Library (CGAL)** (v5.2): offers efficient and reliable geometric algorithms. We use its convex hull algorithm.
- **Concaveman** provides a concave hull implementation based on the algorithm of Park et al. [2, 85].
- **Vectorized stress majorization**: Wang et al. provide a prototype implementation [11] of the vectorized stress majorization [103] in C++ and CUDA (v11.6). The prototype did not support all features mentioned in their work. While the authors stated that their layout algorithm is fully interactive, the precomputation time of matrices was excluded.

In the following we detail our prototype implementation in general and per pipeline step (recall Chapter 3). We elaborate on the integration and required adaptations of the above-mentioned frameworks, libraries, and algorithms. Qt provides the windowing system, basic user interaction, and file loading operations. It also offers a 2D rendering framework, the Graphics View, which allows us to draw and interact with custom 2D graphic items. For the most common glyphs of the SBGN, including those needed to draw our results, we implemented rendering and interaction in our own SBGN viewer. The subgraph implementation of the boost graph library is used as the backbone data structure for the SBGN map.

Acquire & Prepare

As described in Section 3.1, we created a small interaction tool to supplement missing compartment references in SBGN-ML files. For this purpose, we used LibSBGN to load the data into boost's subgraph, and then Qt to draw the SBGN map. The needed interactions are implemented in Qt, and the changes to the data set are then saved into a new SBGN-ML file, again using LibSBGN.

Load & Preprocess

LibSBGN is used to load the SBGN maps. To discern between different SBGN arc types, e.g., production, consumption, or modulatory arcs, we had to extend the LibSBGN library. The reason for this was that the data was parsed from the SBGN-ML format, but not interpreted accordingly.

Data Structure

We transform the loaded data into the clustered graph structure that we proposed in Section 3.3 (Algorithm 1), and store it in boost's subgraph structure for further processing.

Layout

We implemented all discussed initial embeddings (see Subsection 3.4.2). The used pivot MDS implementation from OGDF was adapted to enable a custom selection of pivot nodes. We also extended the pivot MDS implementation to handle disconnected components by calculating the initial embedding for every component separately.

For the initial and constraint layout we used the prototype implementation of the vectorized stress majorization of Wang et al. [103] as a base. We extended and adapted it to support most of the features Wang et al. [103] proposed, e.g., custom target edge lengths, and the transfer of some of Wang et al.'s [103] constraints to CUDA to recalculate $\mathbf{d}_{ij}$ in every layout iteration. Dijkstra's shortest path algorithm [41] from boost is used to calculate the graph theoretical distances. Inspired by Wang et al.'s [103] direction, metric, and shape constraints, we implemented our own domain-specific constraints. We also used boost for graph operations such as traversing, determining node degrees, or calculating disconnected components. For the required vector and matrix calculations, we used Eigen. The convex hull algorithm from CGAL is used to calculate non-overlap constraints.

For our multilevel layout implementation we used boost and Qt. We also created a multilevel layout wizard to inspect and analyze the application of the layout approach at every tree node while traversing through the cluster tree.

Postprocess

In the postprocessing step we used Eigen, boost, and Qt. They are used to calculate the flow direction of the graph and to flip the graph horizontally and/or vertically.

Visualize

Our elastic band visualization is implemented in Qt's Graphics View. For the concave hull we used Concaveman [2]. To determine the minimum spanning tree we used boost's implementation of Kruskal's algorithm [74]. The tree color scheme implementation follows the approach of Tennekes et al. [99]. We implemented an option to switch between bounding box, convex hull, and our elastic band visualizations.

Reduce Complexity

The interaction for the complexity reduction is implemented with Qt. The underlying graph structure uses the boost graph library.

The use of multiple independent frameworks required the implementation of several interfaces and conversions, since every framework uses its own graph implementation. In summary, our implementation encompasses the entire procedure, i.e., from loading the data to the final drawing of metabolic pathways in SBGN, in a single framework.

CHAPTER 4

Results and Discussion

In this chapter we present and discuss results of our work. Firstly, we propose a set of metrics to evaluate the quality of our layout approach. Then, we present four results of a selected variation of metabolic pathways acquired from Reactome [5]. We compare each result with its manually-arranged layout that was created by domain experts and present the corresponding quality metrics during the layout process.

4.1 Quality Metrics

The quality metrics assess the quality of node link diagrams. As reviewed in Subsection 2.5.6, there are common metrics to evaluate syntactic graph drawing requirements, e.g., edge crossings or edge bends. While we evaluate basic metrics of syntactic requirements, we also focus on the domain-specific aspect of our graph drawings and propose metrics to evaluate the *SBGN-ness* of our resulting drawings **(C3)**.

Although we use a graph drawing algorithm based on the stress function, we do not evaluate the stress value in this work. Wang et al. [103] already demonstrated that the stress value is smallest in an unconstrained layout and increases by adding constraints. The authors also mention that a small stress value does not necessarily indicate a layout of high quality for every use-case [65, 103].

Node Overlap Metric

The *node overlap metric* M_{NO} evaluates the syntactic requirement of non-node overlap **(R02)**. This metric depicts the number of node overlaps, i.e., two overlapping nodes are counted as one overlap. A smaller value indicates a better layout:

$$M_{\text{NO}} = \#\text{node overlaps}. \tag{4.1}$$

Edge Crossing Metric

The *edge crossing metric* M_{EC} assesses the syntactic requirement of minimal edge crossings **(R05)**. It counts the number of edge intersections between all edges, including real and dummy edges. A smaller value indicates a better result:

$$M_{\text{EC}} = \#\text{edge intersections}. \tag{4.2}$$

Node Misplacement Metric

The *node misplacement metric* M_{NM} evaluates the semantic requirement of clustering and containment **(R14)**. A node is considered misplaced if it is positioned inside a compartment it is not associated with. The three dummy vertices of a process glyph count in this metric as a single node. We compute two values, the absolute number of misplaced nodes, and the percentage of misplaced nodes:

$$M_{\text{NM}} = \frac{\#\text{misplaced nodes}}{\#\text{nodes}} \cdot 100. \tag{4.3}$$

A misplace percentage of 0% means that all nodes are properly placed within their associated compartments. In contrast, 100% misplacement means that all nodes of the SBGN map are wrongly placed. Our node misplacement metric differs from the metric of Meidiana et al. [81], which assesses the quality of the clustering. In our work, the clustering is already defined by compartment references in the input data sets. For that reason, we do not need to evaluate the clustering.

Flow Deviation Metric

The *flow deviation metric* M_{F} evaluates the semantic requirement of flow direction **(R15)**. It describes the deviation to a preferrable reading direction. Firstly, we compute the flow direction $\mathbf{f}$ as the average direction of all production arcs:

$$\mathbf{f} = \frac{1}{|A|} \cdot \sum_{a \in A} \mathbf{d}(a), \tag{4.4}$$

with A being the set of all production arcs and $\mathbf{d}(a)$ being their directions. We then define the following preferred reading directions:

$$\mathbf{r}_{lr} = (1,0)^T, \quad \mathbf{r}_{tb} = (0,-1)^T, \quad \mathbf{r}_d = (1,-1)^T, \quad \mathbf{r}_c = (0,0)^T, \tag{4.5}$$

with $\mathbf{r}_{lr}$ being from left to right, $\mathbf{r}_{tb}$ being from top to bottom, $\mathbf{r}_d$ being the diagonal from top-left to bottom-right, and $\mathbf{r}_c$ being a circular reading direction. The flow deviation metric is now computed as the minimum length of the reading directions substracted by the normalized flow direction:

$$M_{\text{F}} = \min\left\{\|\mathbf{r}_{lr} - \hat{\mathbf{f}}\|, \|\mathbf{r}_{tb} - \hat{\mathbf{f}}\|, \|\hat{\mathbf{r}}_d - \hat{\mathbf{f}}\|, \|\mathbf{r}_c - \hat{\mathbf{f}}\|\right\}, \tag{4.6}$$

where a smaller number indicates a better match with one preferred reading direction.

Process Orthogonal Deviation Metric

The *process orthogonal deviation metric* M_{PO} assesses the syntactic requirement of link orthogonality **(R08)** for the handles of all process glyphs. For each handle i we compute the angular deviation ω_i to the closest grid axis in degrees:

$$\omega_i = \mathrm{rad2deg}(\angle \max(|\mathbf{h} \cdot \mathbf{e}_0|, |\mathbf{h} \cdot \mathbf{e}_1|)), \tag{4.7}$$

with $\mathbf{h}$ being the handle's direction, $\mathbf{e}_0 = (1,0)^T$, and $\mathbf{e}_1 = (0,1)^T$. We then compute the orthogonal deviation of one process glyph $\sigma_\perp$ as follows:

$$\sigma_\perp = (\omega_0 + \omega_1)/90, \tag{4.8}$$

where ω_0 and ω_1 are the angular deviations of both handles of a process glyph, respectively. By dividing their angular deviations with 90°, we normalize the orthogonal deviation to the interval $[0,1]$. The process orthogonal deviation metric is then computed as follows:

$$M_{\mathrm{PO}} = \frac{1}{|P|} \cdot \sum_{p \in P} \sigma_\perp(p), \tag{4.9}$$

where P is the set of all process glyphs of the entire SBGN map. The M_{PO} describes how far all process handles deviate from orthogonality, where a value of zero means they are all orthogonal, and a value of one means they are diagonal.

Process Angle Deviation Metric

The *process angle deviation metric* M_{PA} evaluates the syntactic requirement to maximize the minimum angle **(R16)** as well as the semantic requirement of node-link connection points **(R17)** for process glyphs and their incoming as well as outgoing edges. For each process glyph we calculate the process handle direction and generate a process angle reference shape according to Section 3.4.3, but aligned to the handle's direction. Then, we compare each incoming and outgoing real edge of the process glyph with its closest ideal edge of the reference shape:

$$\rho(e) = (\hat{\mathbf{d}}_i(e) \cdot \hat{\mathbf{d}}_r(e) + 1)/2, \tag{4.10}$$

where $\hat{\mathbf{d}}_i(e)$ is the normalized direction of the ideal edge, and $\hat{\mathbf{d}}_r(e)$ is the normalized direction of the real edge. To obtain an angular deviation $\rho(e)$ in the interval $[0,1]$, we shift the result of the dot product. The deviation of the incoming $m^{\mathrm{in}}_{\mathrm{PA}}(p)$ and outgoing $m^{\mathrm{out}}_{\mathrm{PA}}(p)$, as well as all edges $m_{\mathrm{PA}}(p)$ for a single process glyph are calculated as follows:

$$m^{\mathrm{in}}_{\mathrm{PA}}(p) = \frac{1}{|I|} \cdot \sum_{e \in I} \rho(e), \tag{4.11}$$

$$m^{\mathrm{out}}_{\mathrm{PA}}(p) = \frac{1}{|O|} \cdot \sum_{e \in O} \rho(e), \tag{4.12}$$

$$m_{\mathrm{PA}}(p) = (m^{\mathrm{in}}_{\mathrm{PA}}(p) + m^{\mathrm{out}}_{\mathrm{PA}}(p))/2, \tag{4.13}$$

with I being the set of incoming edges and O being the set of outgoing edges of process glyph p. The combined metrics over all process glyphs are then computed as follows:

$$M_{\text{PA}}^{\text{in}} = \frac{100}{|P|} \cdot \sum_{p \in P} m_{\text{PA}}^{\text{in}}(p), \tag{4.14}$$

$$M_{\text{PA}}^{\text{out}} = \frac{100}{|P|} \cdot \sum_{p \in P} m_{\text{PA}}^{\text{out}}(p), \tag{4.15}$$

$$M_{\text{PA}} = \frac{100}{|P|} \cdot \sum_{p \in P} m_{\text{PA}}(p). \tag{4.16}$$

These angle deviations are given as percentages, where 0% means that the real edges completely coincide with the ideal edges of the reference shape, and 100% means that they point in the opposite direction.

Modulatory Arc Deviation Metric

The *modulatory arc deviation metric* M_{MA} assesses the quality of the connection between modulatory arcs and process glyphs. It evaluates the semantic requirement of node-link connection points **(R17)**. For each modulatory arc r we define its valid connecting region to its process glyph according to the SBGN. This region is defined as the circle sector $\pm 45°$ around the normalized vector $\hat{\mathbf{n}}(r)$ that is perpendicular the process glyph's handle direction. This region exists in both half spaces generated by the process glyph's handles. The region membership $\phi(r)$ of a modulatory arc is calculated as follows:

$$\phi(r) = \begin{cases} 1 & \text{if } \angle|\hat{\mathbf{d}}(r) \cdot \hat{\mathbf{n}}(r)| \leq \pi/4, \\ 0 & \text{otherwise,} \end{cases} \tag{4.17}$$

with $\hat{\mathbf{d}}(r)$ being the normalized direction vector of the modulatory arc. The modulatory arc deviation metric is then computed over all modulatory arcs R of the SBGN map:

$$M_{\text{MA}} = \frac{100}{|R|} \cdot \sum_{r \in R} (1 - \phi(r)). \tag{4.18}$$

This metric describes the percentage of modulatory arcs that deviate from their valid connecting region, where 0% means all are inside and 100% means all are outside.

Total Deviation Metric

To provide a single factor that indicates the overall SBGN deviation, we compute the *total deviation metric* as follows:

$$M_{\text{total}} = (M_{\text{NM}} + M_{\text{PO}} + M_{\text{PA}} + M_{\text{MA}})/4, \tag{4.19}$$

where 0% means the layout is nearly SBGN-compliant, and 100% means the layout is far from adhering to the SBGN.

4.2 Eukaryotic Translation Elongation

The biological process of eukaryotic translation, transforms messenger Ribonucleic acid (mRNA) into proteins within an eukaryotic cell [70]. An eukaryotic cell, is a cell consisting of cytosol (the liquid inside a cell membrane) and a nucleus (the cell kernel). It consists of four phases: initiation, elongation, termination, and recapping. Our first result data set describes the second phase, the *eukaryotic translation elongation* (stable identifier: R-HSA-156842) [6]. The eukaryotic translation elongation is part of the Homo sapiens' metabolism of proteins.

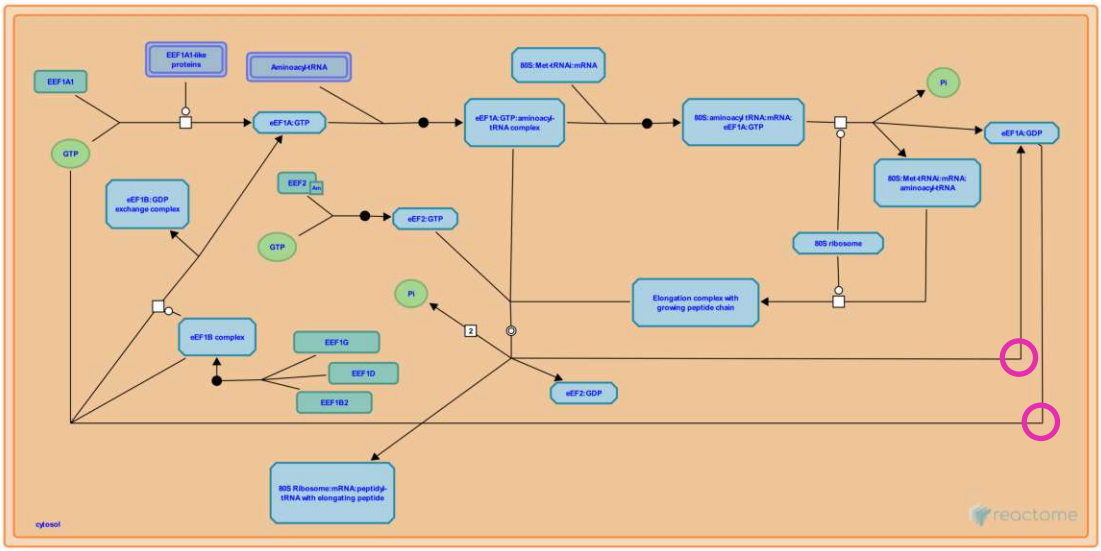
The data sets consists of 51 vertices and 56 edges. Out of this 51 vertices, 27 vertices belong to 9 process glyphs. Recall from Section 3.3 that every process glyph consists of three vertices. The SBGN map has one compartment, representing the cytosol, which encapsulates all vertices. The data set also consists of one connected component. We selected this data set to first demonstrate our approach on a network with a moderate number of nodes and only one compartment.

The original manually-arranged layout of the eukaryotic translation elongation's SBGN map is shown in Figure 4.1a. To avoid edge crossings, two edges are bent, as depicted with the magenta circles. The layout has no overlaps, no edge crossings, and no misplaced nodes. The data flow direction is circular clockwise. Figure 4.1b shows the loaded data set, which deviates from the original layout since we do not model edge bends. Its quality metrics are shown in Table 4.1. By removing the edge bends, 13 edge crossings where introduced. Our metrics report one node overlap, which is caused by the overly large bounding box of the process glyph depicted by the magenta arrow in Figure 4.1b. The modulatory arcs deviate the most with 25%. Since there are only 4 modulatory arcs in the data set, a single deviation has a large impact. The total deviation from the SBGN is 8.9%.

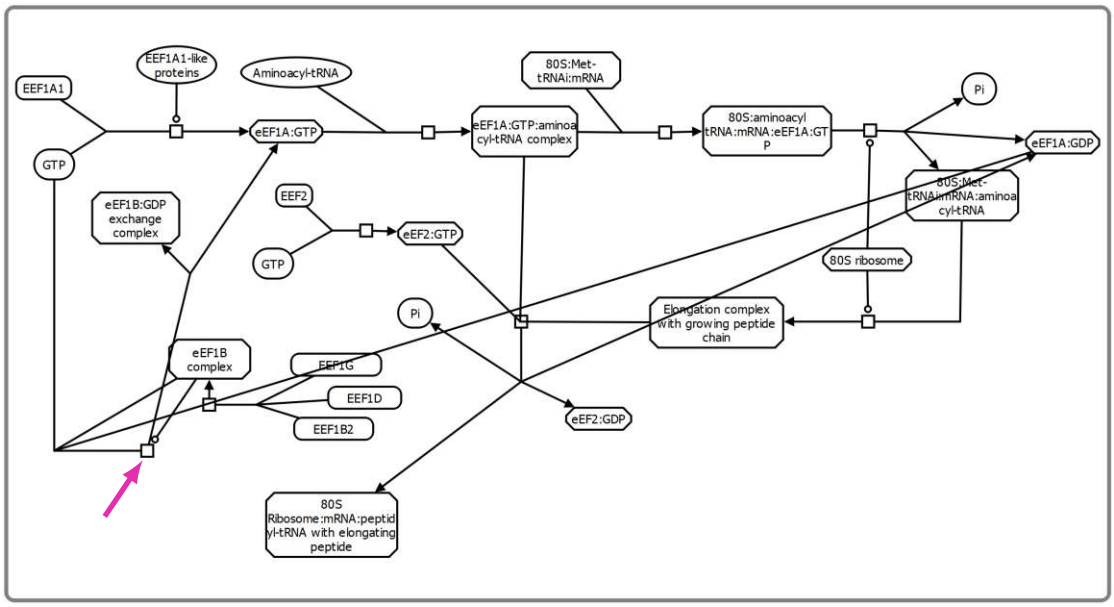
We then apply our layout process. Since the data set has only one component in one compartment we do not need to apply our multilevel approach. We use pivot MDS to calculate the initial embedding (Figure 4.2a), followed by an unconstrained stress majorization (Figure 4.2b). Then we constrain this initial layout in four refinement steps. We apply the constraints as proposed in Section 3.4.3. Starting with the iterative non-node overlap constraint (Figure 4.2c), we skip the compartment constraint as there is only one. Then we apply the process orthogonal constraint (Figure 4.2d), followed by the angle constraint (Figure 4.2e) and the modulatory arc constraint (Figure 4.2f). The constraints of previous refinement steps stay active in the next one. Parameters, weights, and the quality metrics are reported after every layout and refinement step, as shown in Table 4.1. The initial layout has with 16.5% a larger deviation from SBGN than the manual layout. Process handles are not orthogonal, reflected in a M_{PO} value of 54.6%, this is much larger than the value of the manual layout with 2.5%. The single node overlap and 13 edge crossings of the manual layout are balanced in the initial layout to seven overlaps and seven crossings. Each refinement step is specialized to improve one quality metric, as reflected in our results, e.g., the orthogonality of process handles

improved from 60.3% to 0.1% through the process orthogonal constraint. After applying all constraints we could reduce the total deviation from SBGN from 16.5% to 5.7% in the final layout result. In contrast to the manual layout, the deviations are more evenly distributed. The postprocessing step did not change the layout.

The convex hull visualization of the data set is shown in Figure 4.3a and our colored elastic band visualization (hue range $[0, 360]$ and fraction of 0.75) is presented in Figure 4.3b.

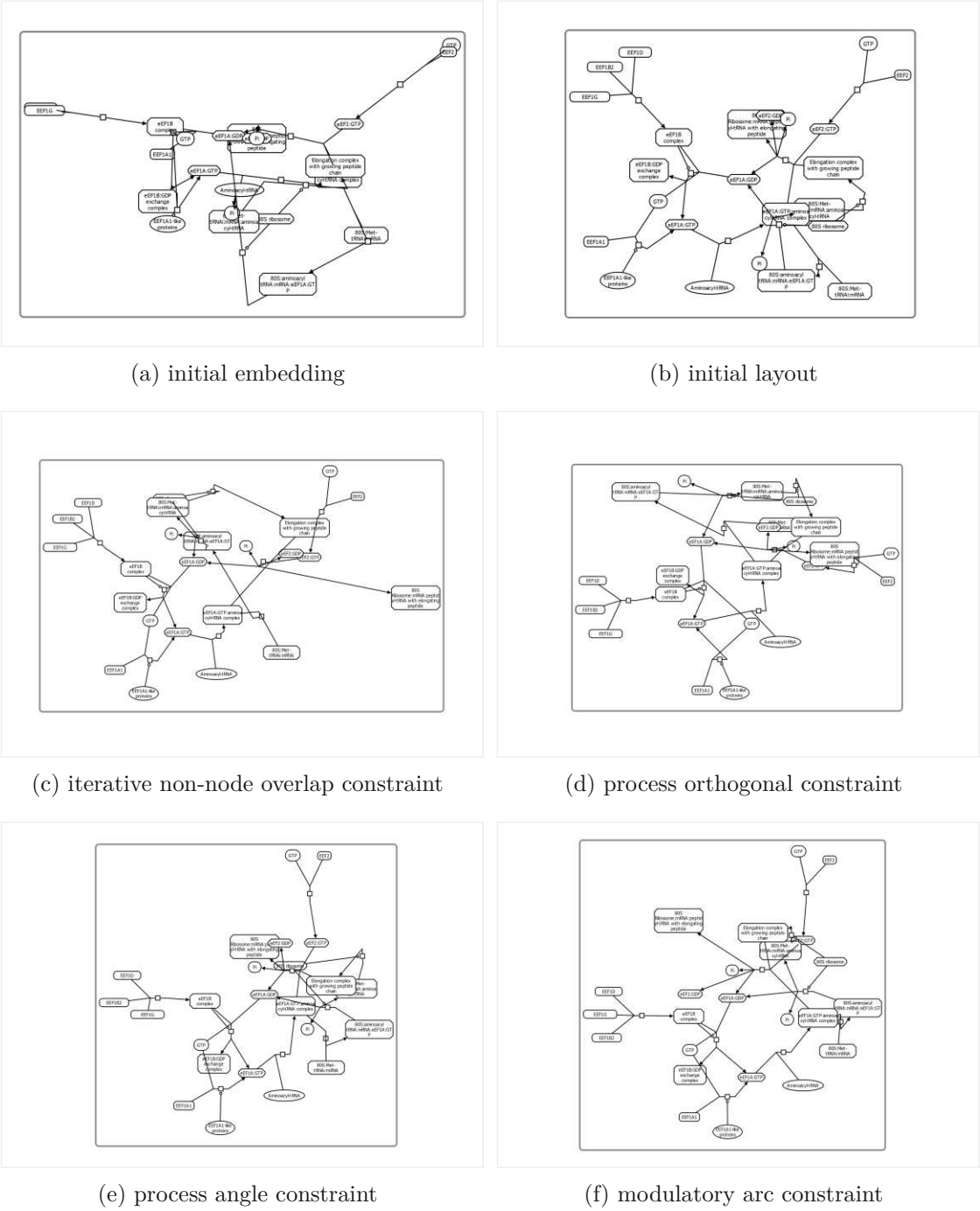


(a) manually-arranged layout (Image taken from [6])



(b) loaded layout

Figure 4.1: Eukaryotic translation elongation (R-HSA-156842). (a) shows the manually-arranged layout and (b) shows the same data set when loaded in our SBGN viewer. Since we do not support edge bends (magenta circles), our layout differs from the manually-arranged one. The process node depicted with the magenta arrow leads to a node-overlap caused by its elongated handles.



(a) initial embedding

(b) initial layout

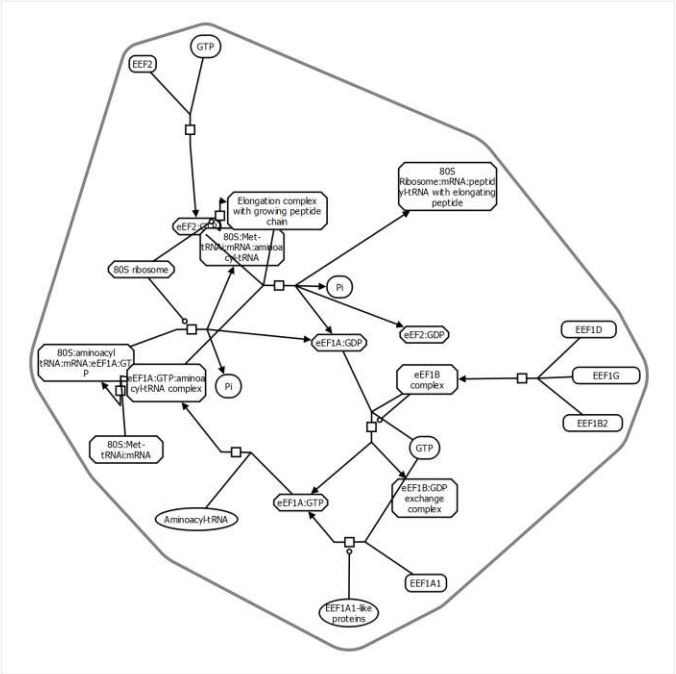
(c) iterative non-node overlap constraint

(d) process orthogonal constraint

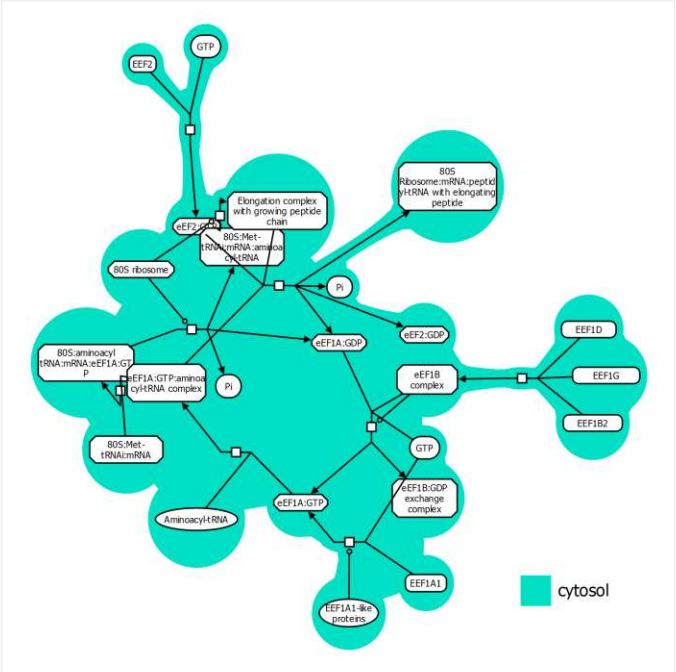
(e) process angle constraint

(f) modulatory arc constraint

Figure 4.2: The layout procedure of eukaryotic translation elongation (R-HSA-156842). The initial embedding is shown in (a) and the initial layout is presented in (b). The result of the iterative non-node overlap constraint is shown in (c). The effects of the process orthogonal and process angle constraints are shown in (d) and (e), respectively. After applying the modulatory arc constraint, the final layout is presented in (f). The parameters and quality metrics for each step are shown in Table 4.1.



(a) convex hull



(b) elastic band

Figure 4.3: Visual mapping of the final layout of the eukaryotic translation elongation (R-HSA-156842). This layout was flipped during postprocessing step, compare to Figure 4.2f. (a) shows the automatic layout with a convex hull. (b) shows the same layout with our elastic band visualization colored with the tree color approach with a starting hue range of $[0, 360]$ and fraction 0.75.

Table 4.1: Parameters and quality metrics of eukaryotic translation elongation (R-HSA-156842). The corresponding layouts of the steps described in the columns can be found in Figure 4.1 and Figure 4.2. For each steps we report our quality metrics (Section 4.1) and the layout's parameters for reproducibility.

# nodes = 51 # edges = 56 Metric	manually-arranged (loaded) layout	initial embedding (pivot MDS)	initial layout	iterative non-node overlap constraint	process orthogonal constraint	process angle constraint	modulatory arc constraint
# node overlaps	1	33	7	3	4	4	6
# edge crossings	13	5	7	9	11	8	6
# node misplacement	0	0	0	0	0	0	0
M_{NM} [%]	0.0	0.0	0.0	0.0	0.0	0.0	0.0
M_{F}	0.40	0.25	0.18	0.05	0.18	0.08	0.23
M_{PO} [%]	2.5	30.5	54.6	60.3	0.1	1.7	1.0
M_{PA} [%]	8.1	16.2	11.4	18.5	40.8	23.5	21.7
$M_{\mathrm{PA}}^{\mathrm{in}}$ [%]	13.6	9.6	8.6	13.1	27.5	20.0	22.0
$M_{\mathrm{PA}}^{\mathrm{out}}$ [%]	2.7	23.6	14.2	24.0	54.1	26.3	21.3
M_{MA} [%]	25.0	0.0	0.0	25.0	50.0	25.0	0.0
M_{total} [%]	8.9	11.7	16.5	26.0	22.7	12.6	5.7
Parameter							
edge elongation	–	–	off	off	off	off	off
edge length [px]	–	–	80	80	80	80	80
handle edge length [px]	–	–	20	20	20	20	20
iterations	–	–	50	50	50	50	50
constraint weight	–	–	–	2	3	4	5

4.3 Protein Repair

Proteins are constantly adversely impacted by internal and external influences. For example, Reactive Oxygen Species (ROS), like H_2O_2, cause damage by reacting with molecules inside a cell and, thus, affecting its functions. Impaired cell functions can lead to different diseases and disorders or influence the aging process. The *protein repair* system (stable identifier: R-HSA-5676934) within a cell facilitates the reversal of this damage to some amino acid side chains [7]. Our second result data set shows how the protein repair system reverses the oxidation of the protein Methionine (L-Met) caused by H_2O_2 [7, 34]. This specific process takes place inside the human cells and belongs to the metabolism of proteins.

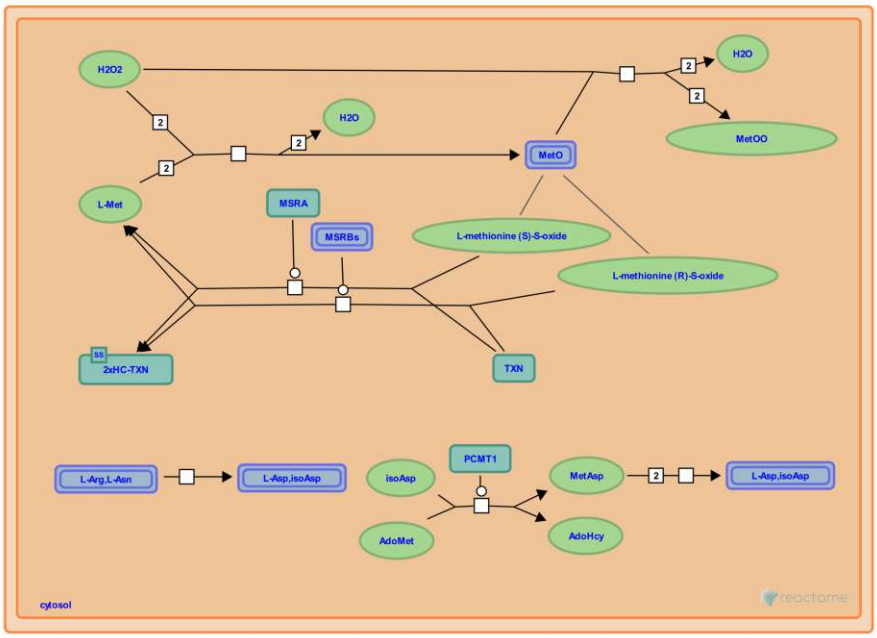
This data set consists of 41 vertices (including 7 process glyphs with 21 vertices) and 41 edges. Three separate biological processes represented by three disconnected graph components are contained within one compartment, the cytosol. The compartment holds all vertices. This data set was selected to demonstrate how our approach handles disconnected components.

In Figure 4.4a we show the manually-arranged layout of the data set [7]. Loading the data set in our tool reveals that two process glyphs have inverted handle directions, depicted with the magenta circles and shown in Figure 4.4b. This divergence between the loaded and the original layout is caused by a divergence in the data set. Thus, the loaded data set contains 14 edge crossings, which are mainly located in the magenta circles in Figure 4.4b, compared to the two edge crossings of the original layout. The loaded layout has no node overlaps, and no nodes are misplaced. The quality metrics are reported in Table 4.2. The metric most deviating from being optimal is the process angle deviation for the outgoing edges with 24.1%, but this may be caused by the inverted process node handles. The total deviation sums up to 3.1%.

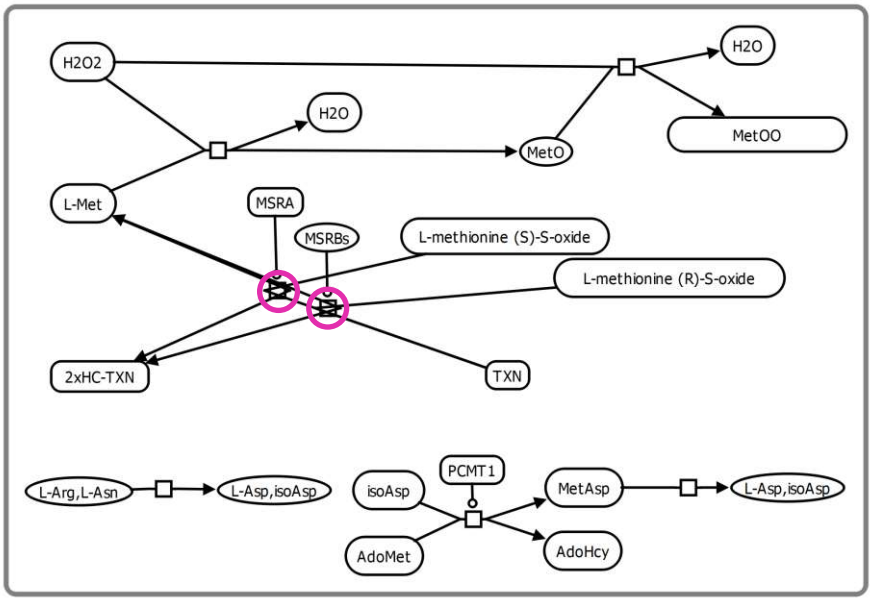
Since the data set consists of three disconnected components, we apply our multilevel approach to generate their arrangement. The quality measures are reported after every multilevel layout step is transferred to the network. Firstly, we create a global reference layout by applying pivot MDS for initial placement (Figure 4.5a) and then the initial layout through unconstraint stress majorization. As described in Subsection 3.4.4, this leads to untangled disconnected components placed on top of each other, shown in Figure 4.5b. In the initial embedding, the modulatory arc deviation is highest with 66.7%. In the initial layout, the highest deviation is the process orthogonal deviation with 43.6%. The total deviation of the initial layout is 12.6%. The global untangled initial layout is saved to be used as initial embedding in the subsequent leaf level layouts. We then start to layout the network top-down by traversing through its cluster tree. The rectangle in Figure 4.5c visualizes the single compartment that is placed at the cluster tree's root level. After applying this top level layout, all nodes of the data set collapse to one point. This is reflected in the high number of 351 node overlaps in this step. Level one of the cluster tree holds the three disconnected components, represented as circles in Figure 4.5d. They are placed by applying a vertical alignment (depicted

as V in Table 4.2). This alignment decreasing the number of node overlaps from 351 to 151. Afterwards we arrange and constrain the first leaf node of the cluster tree. The initial embedding for this subgraph is taken from the global initial layout. Then, we perform an initial layout with unconstraint stress majorization. Next, we apply the constraints in the following order: iterative non-node overlap, process orthogonal, process angle, and finally modulatory arc constraint. The result of the first leaf node's layout is presented in Figure 4.5e. Again, the number of node overlaps decreased, this time to 31, and additionally the other quality metrics improved. The same procedure is applied to the two subsequent leaf nodes of the cluster tree, shown in Figure 4.5f and Figure 4.5g, resulting in the final layout given in Figure 4.5h. The continuous improvement of the layout can be traced through the change in total deviation throughout our multilevel approach. The total deviation starts at 59.2% and improves to 1.4%. This is a better score than the initial layout and the loaded layout received. The postprocessing step was not applied.

In Figure 4.6a we show the convex hull representation of the final layout and in Figure 4.6b we present the colored elastic band visualization. The hue range is $[0, 120]$ with a fraction of 0.75.

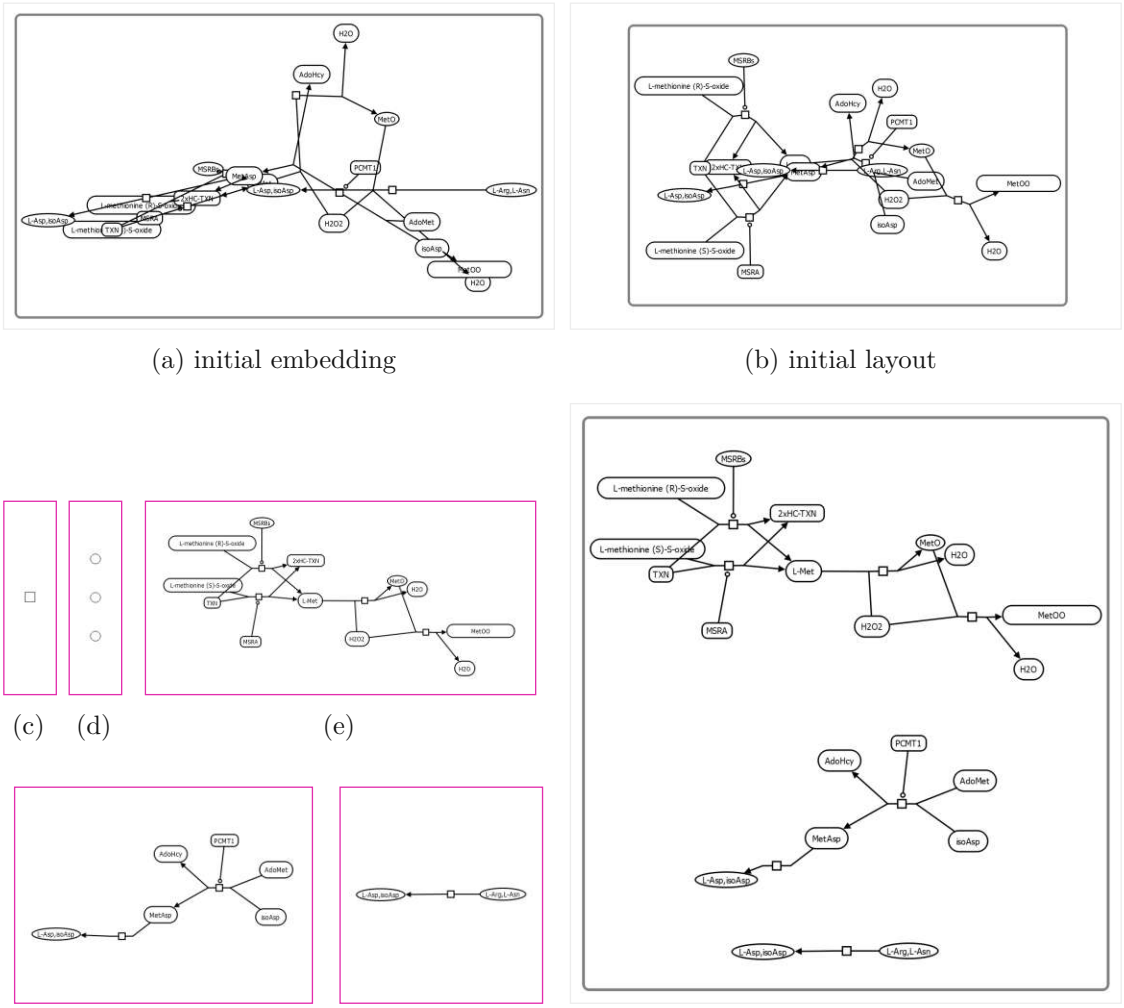


(a) manually-arranged layout (Image taken from [7])



(b) loaded layout

Figure 4.4: Protein repair of Methionine (L-Met) (R-HSA-5676934). (a) shows the manually-arranged layout and (b) shows the same data set loaded with our SBGN viewer. The handles of two process glyphs (magenta circles) have inverted edge directions, caused by a discrepancy between the stored data set and the image displayed in (a).

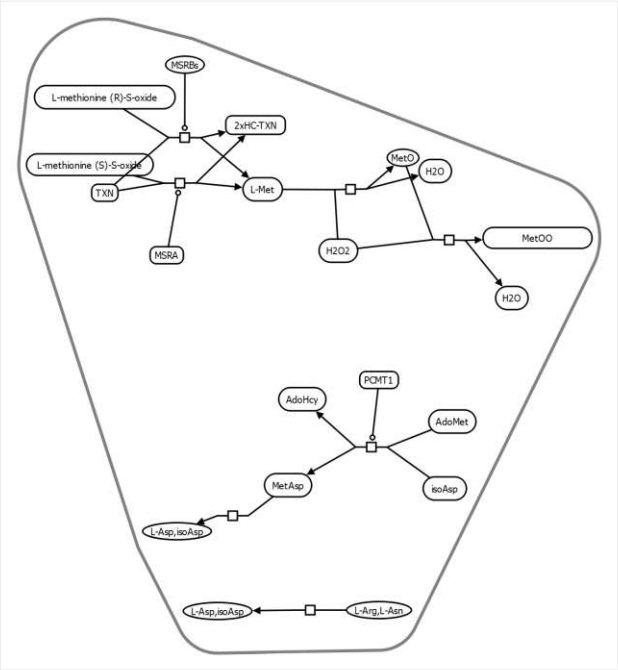


(a) initial embedding (b) initial layout

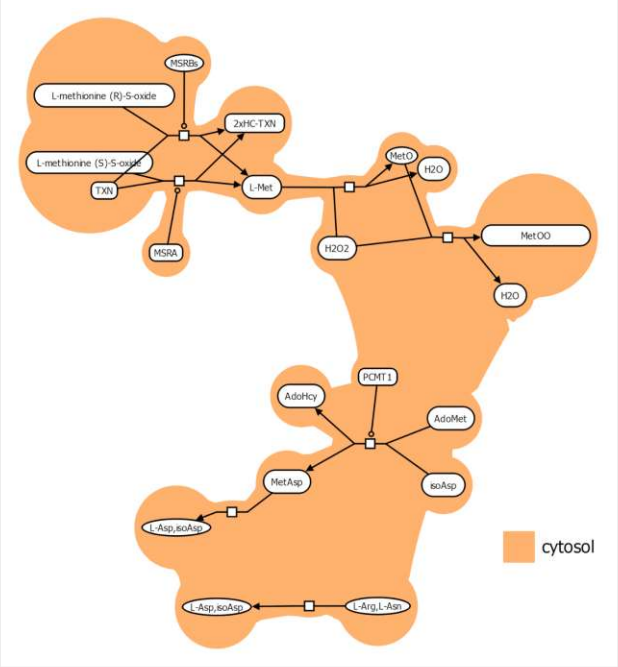
(c) (d) (e)

(f) (g) (h) final layout

Figure 4.5: The layout procedure of the protein repair of Methionine (L-Met) (R-HSA-5676934). The global initial embedding and initial layout are shown in (a) and (b), respectively. (c) shows the compartments' placement at the cluster trees root level, followed by the vertical arrangement of the three disconnected components of level one shown in (d). The separate arrangements of the cluster tree's leaf nodes are presented in (e), (f), and (g). The combined final layout is shown in (h).



(a) convex hull



(b) elastic band

Figure 4.6: Visual mapping of the final layout of the protein repair system for Methionine (L-Met) (R-HSA-5676934). (a) shows the automatic layout with a convex hull. (b) shows the same layout with our elastic band visualization colored with the tree color approach using a starting hue range of $[0, 120]$ and a fraction of 0.75.

Table 4.2: Parameters and quality metrics of protein repair of Methionine (L-Met) (R-HSA-5676934). The corresponding layouts of the steps described in the columns can be found in Figure 4.4b and Figure 4.5. Each column reports our quality metrics from Section 4.1 for its associated layout step. The magenta box depicts the multilevel layout steps. A vertical alignment (V) was applied to place disconnected components at tree level 1. We documented the parameters of the layout steps for reproducibility, constraints are reported as weight;iterations.

# nodes = 41 # edges = 41 Metric	manually-arranged (loaded) layout	initial embedding (pivot MDS)	initial layout	cluster tree level 0 after Figure 4.5c	cluster tree level 1 after Figure 4.5d	cluster tree level 2 (leaf) after Figure 4.5e	cluster tree level 2 (leaf) after Figure 4.5f	cluster tree level 2 (leaf) after Figure 4.5g, final layout
# node overlaps	0	30	8	351	151	31	3	0
# edge crossings	14	12	12	0	0	3	3	3
# node misplacement	0	0	0	0	0	0	0	0
M_{NM} [%]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
M_{F}	0.33	0.05	0.07	0.41	0.42	0.40	0.27	0.27
M_{PO} [%]	0.0	39.6	43.6	100.0	100.0	86.7	29.7	1.1
M_{PA} [%]	12.4	9.2	6.7	36.6	23.9	10.7	4.5	4.5
$M_{\text{PA}}^{\text{in}}$ [%]	0.8	2.9	1.5	35.2	21.4	10.5	4.8	4.8
$M_{\text{PA}}^{\text{out}}$ [%]	24.1	16.2	11.8	38.0	26.4	10.8	4.2	4.2
M_{MA} [%]	0.0	66.7	0.0	100.0	100.0	33.3	0.0	0.0
M_{total} [%]	3.1	28.9	12.6	59.2	56.0	32.7	8.6	1.4
Parameter/Constraint								
edge elongation		–	off	–	–	off	off	off
edge length [px]		–	80	80	80	80	80	80
handle edge length [px]		–	20	20	20	20	20	20
iterations		–	50	–	–	–	–	–
disconnected components				–	V	–	–	–
iterative non-node overlap constraint				–	–	2;50	2;50	2;50
compartment constraint				–	–	–	–	–
process orthogonal constraint				–	–	3;50	3;50	3;50
process angle constraint				–	–	4;100	4;100	4;100
modulatory arc constraint				–	–	10;200	10;200	10;200

4.4 Protein Methylation

After protein biosynthesis, enzymatic modification of these proteins can take place. This process is called Post-Translational Modification (PTM). One of these PTMs is the methylation of proteins, where methyl groups are added to proteins. The communication between different PTMs controls cellular functions like protein synthesis, signal transduction, and DNA repair. The third result data set shows the *protein methylation* of the amino acids arginine and lysine on non-histone proteins. This process is an important regulator of cellular signal transduction pathways [8, 29]. It takes place inside the human cells and belongs to the metabolism of proteins.

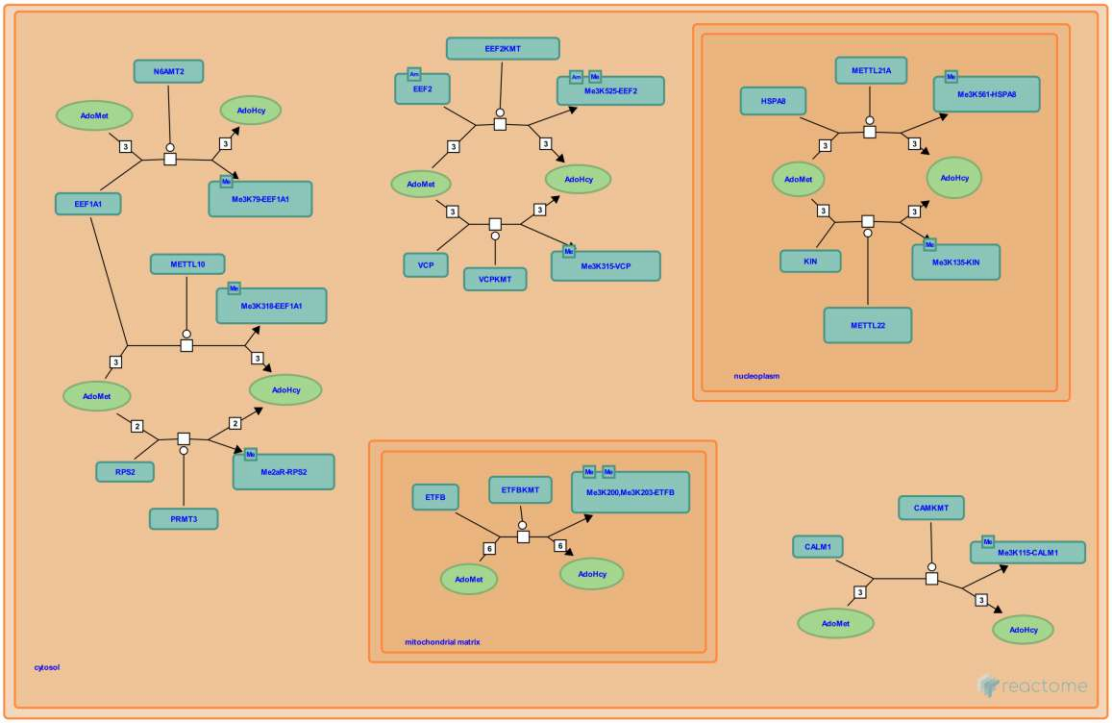
The data set consists of 65 vertices (27 vertices are part of 9 process nodes) and 63 edges. There are no differences between the original, manually-arranged layout and the loaded layout. Five separate processes take place inside the cytosol, two of them in subcellular locations, i.e., the nucleoplasm and the mitochondrial matrix. Hence, the data set consists of one parent compartment (cytosol) that contains three disconnected components and two child compartments (nucleoplasm, mitochondrial matrix). Each of these child compartments contains one disconnected component. The data set was selected to demonstrate nested compartment structures and the handling of disconnected components.

Figure 4.7a shows the manually-arranged data set [8]. Comparing it to the data loaded in our tool (shown in Figure 4.7b), we see no differences. The orthogonal process deviation is the most prominent one with 5.6%. The loaded data is nearly SBGN-conform with a total deviation of 1.8%. The quality metrics of the loaded layout and the layout process are reported in Table 4.3.

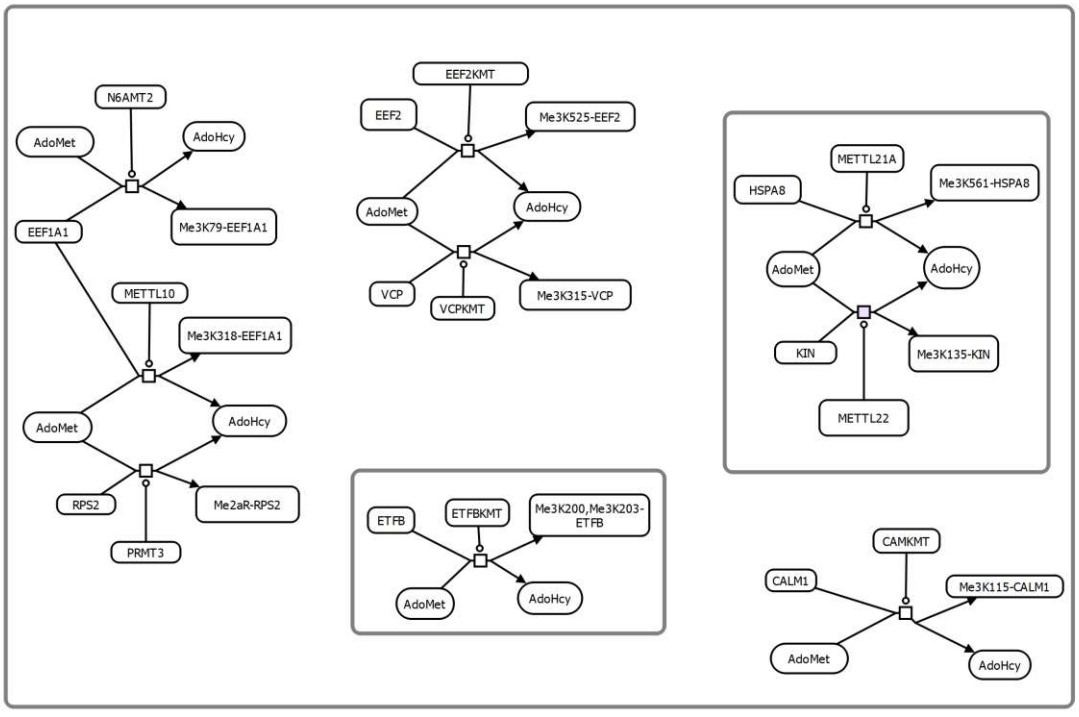
Since the data set consists of disconnected components, as well as nested compartments, we apply our multilevel approach, and again report the quality metrics after every multilevel layout step. The global reference layout is created by an initial pivot MDS embedding (Figure 4.8a), followed by the unconstrained stress majorization (Figure 4.8b). As a result, the disconnected components are placed on top of each other. We now apply our multilevel approach. The process can be observed in Figure 4.8, highlighted with magenta outlines. Compared to our second data set (Section 4.3), we are traversing two levels, since there are compartments contained inside a compartment. On the first level, we apply a horizontal layout strategy (depicted as H in Table 4.3) for the five disconnected components (Figure 4.8d). In every leaf node we apply our layout approach. As initial embedding we take a subpart of the global reference layout, apply the initial layout and subsequently the constraints. Compartment constraints were not necessary and therefore omitted. The final layout is shown in Figure 4.8l. We can observe that the number of misplaced nodes already resolved to zero after the second multilevel layout step (Figure 4.8d). Two of the subgraphs did not untangle optimally (Figure 4.8g and Figure 4.8i). Their layout could be improved by changing the weights of the constraints or the number of iterations. The total deviation was reduced from 30.2% for

the initial layout to 1.0% for the final layout. The SBGN-ness of the final layout is better than the loaded layout. A reason for this could be the alignment of the process glyphs' handles. The postprocessing step was not applied.

In Figure 4.9a we show the convex hull representation of the final layout and in Figure 4.9b we present the colored elastic band visualization. The hue range is $[0, 120]$ with a fraction of 0.75.

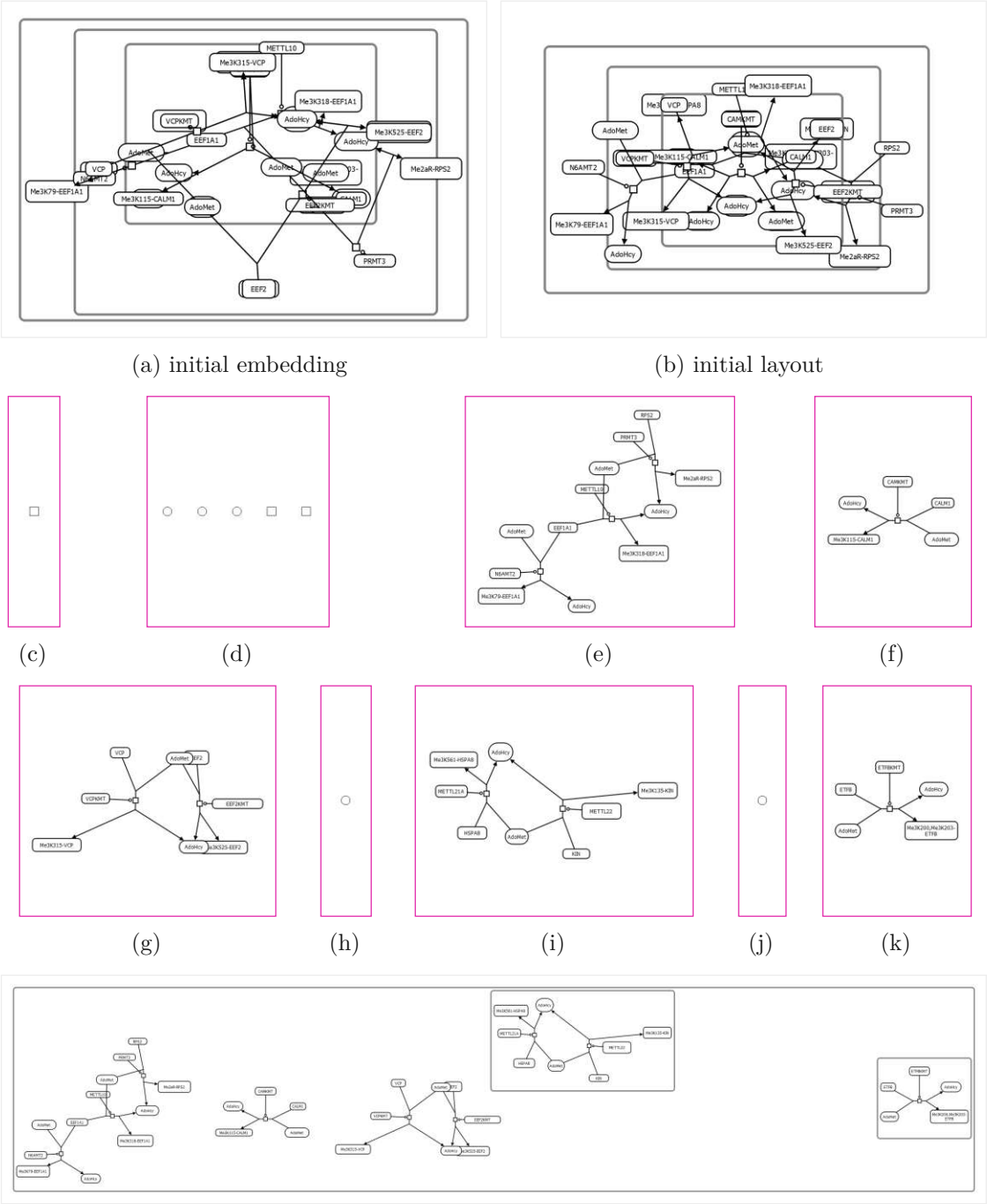


(a) manually-arranged layout (Image taken from [8])



(b) loaded layout

Figure 4.7: Protein methylation (R-HSA-8876725). (a) shows the manually-arranged layout and (b) shows the same data set loaded with our SBGN viewer.



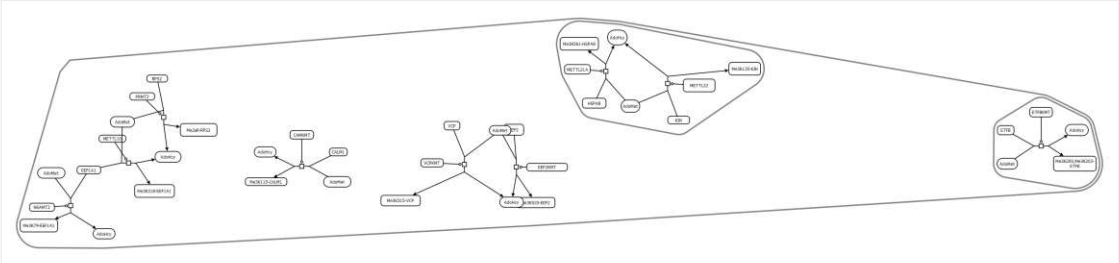
(a) initial embedding (b) initial layout

(c) (d) (e) (f)

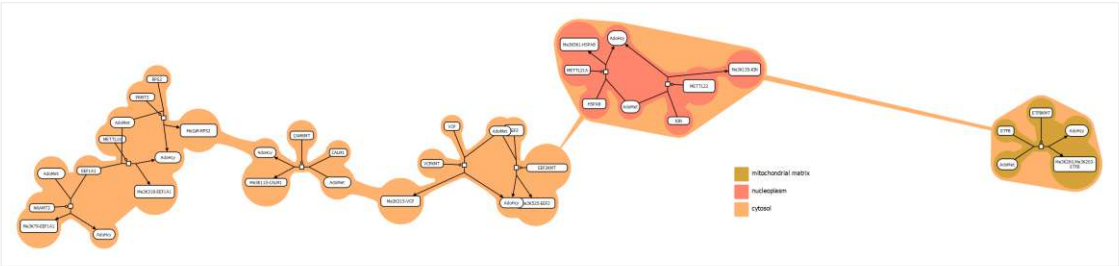
(g) (h) (i) (j) (k)

(l) final layout

Figure 4.8: The layout procedure of the protein methylation (R-HSA-8876725). The global initial embedding and initial layout are shown in (a) and (b), respectively. (c) shows the compartments placement at the cluster tree's root level, followed by the horizontal arrangement of the five disconnected components of level one shown in (d). The separate arrangements of the cluster tree's leaf nodes are presented in (e), (f), (g), (i), and (k). (h) and (j) represent the child compartments of the two rectangles shown in (d). The combined final layout is shown in (l).



(a) convex hull



(b) elastic band

Figure 4.9: Visual mapping of the final layout of protein methylation (R-HSA-8876725). (a) shows the automatic layout with a convex hull. (b) shows the same layout with our elastic band visualization, colored with the tree color approach using a starting hue range of $[0, 120]$ and fraction 0.75.

Table 4.3: Parameters and quality metrics of protein methylation (R-HSA-8876725). The corresponding layouts of the steps described in the columns can be found in Figure 4.7b and Figure 4.8. The magenta box depicts the multilevel layout steps. A horizontal alignment (H) was applied to place disconnected components at tree level 1. Each column reports the quality metrics of Section 4.1 for the associated layout step. We documented the parameters of the layout steps for reproducibility, constraints are reported as weight;iterations.

# nodes = 65 # edges = 63 Metric	manually-arranged (loaded) layout	initial embedding (pivot MDS)	initial layout	cluster tree level 0	cluster tree level 1	cluster tree level 2 (leaf)	cluster tree level 2 (leaf)	cluster tree level 2 (leaf)	cluster tree level 2 (compartment)	cluster tree level 3 (leaf)	cluster tree level 2 (compartment)	cluster tree level 3 (leaf) final layout
# node overlaps	0	74	39	1081	306	179	144	73	68	17	17	2
# edge crossings	0	71	54	0	0	2	2	2	2	2	2	2
# node mispl.	0	37	36	47	0	0	0	0	0	0	0	0
M_{NM} [%]	0.0	78.7	76.6	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
M_{F}	0.11	0.14	0.17	0.06	0.44	0.25	0.18	0.33	0.33	0.18	0.18	0.24
M_{PO} [%]	5.6	26.2	42.5	100.0	100.0	67.0	55.9	34.2	34.2	12.2	12.2	1.1
M_{PA} [%]	1.7	10.7	1.7	26.2	34.6	19.3	16.5	11.5	11.5	4.3	4.3	2.7
$M_{\mathrm{PA}}^{\mathrm{in}}$ [%]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$M_{\mathrm{PA}}^{\mathrm{out}}$ [%]	3.4	21.5	3.5	52.4	69.2	38.6	33.1	23.0	23.0	8.6	8.6	5.5
M_{MA} [%]	0.0	0.0	0.0	100.0	100.0	66.7	55.6	33.3	33.3	11.1	11.1	0.0
M_{total} [%]	1.8	28.9	30.2	81.6	58.7	38.3	32.0	19.8	19.8	6.9	6.9	1.0
Parameter/Constraint												
edge elongation		–	off	–	–	off	off	off	–	off	–	off
edge length [px]		–	80	80	80	80	80	80	80	80	80	80
handle edge length [px]		–	20	20	20	20	20	20	20	20	20	20
iterations		–	50	–	–	–	–	–	–	–	–	–
disconnected components				–	H	–	–	–	–	–	–	–
iterative non-node overlap constraint				–	–	2;50	2;50	2;50	–	2;50	–	2;50
compartment constraint				–	–	–	–	–	–	–	–	–
process orthogonal constraint				–	–	3;50	3;50	3;50	–	3;50	–	3;50
process angle constraint				–	–	4;150	4;100	4;100	–	4;200	–	4;200
modulatory arc constraint				–	–	5;200	5;100	5;100	–	5;200	–	5;200

4.5 Nucleotide Excision Repair

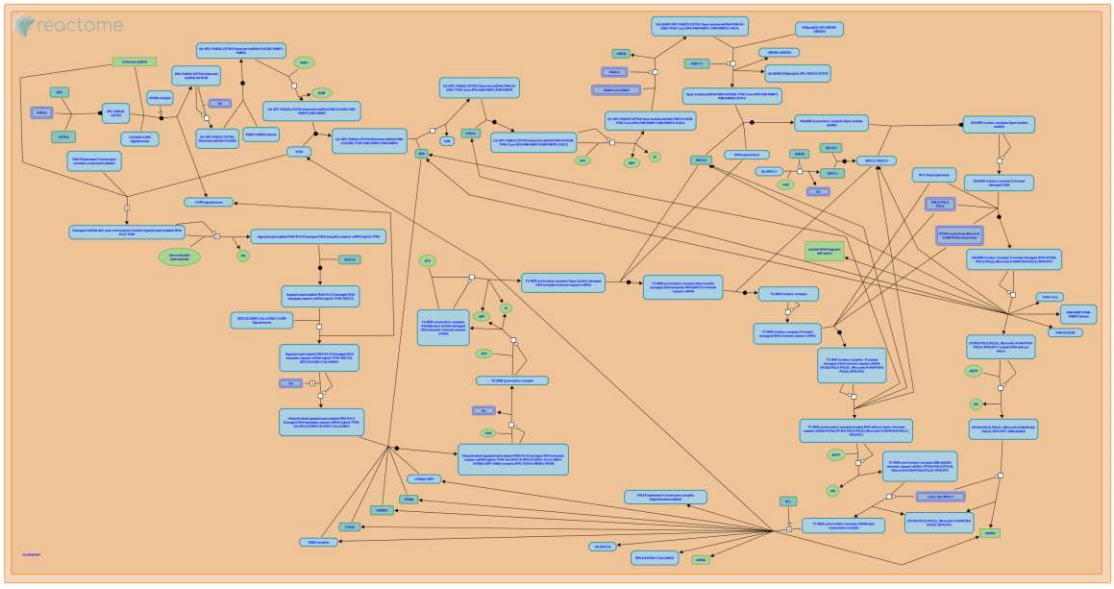
Nucleotide Excision Repair (NER) (stable identifier: R-HSA-5696398) is a process that facilitates cell survival and the recovery of DNA synthesis by removing bulky base damage from DNA enzymatically. The NER pathway is involved in removing mutations of DNA caused by ultraviolet light and genotoxic agents. This process takes place not only in human cells but also in other organisms like E. coli [9, 35].

The data set consists of 214 vertices (37 process nodes consisting of 111 vertices) and 253 edges. It has one compartment, representing the nucleoplasm that contains all vertices of the SBGN map in one connected component. We selected this data set because it contains large node labels and high-degree nodes. It also demonstrates the scalability of our approach.

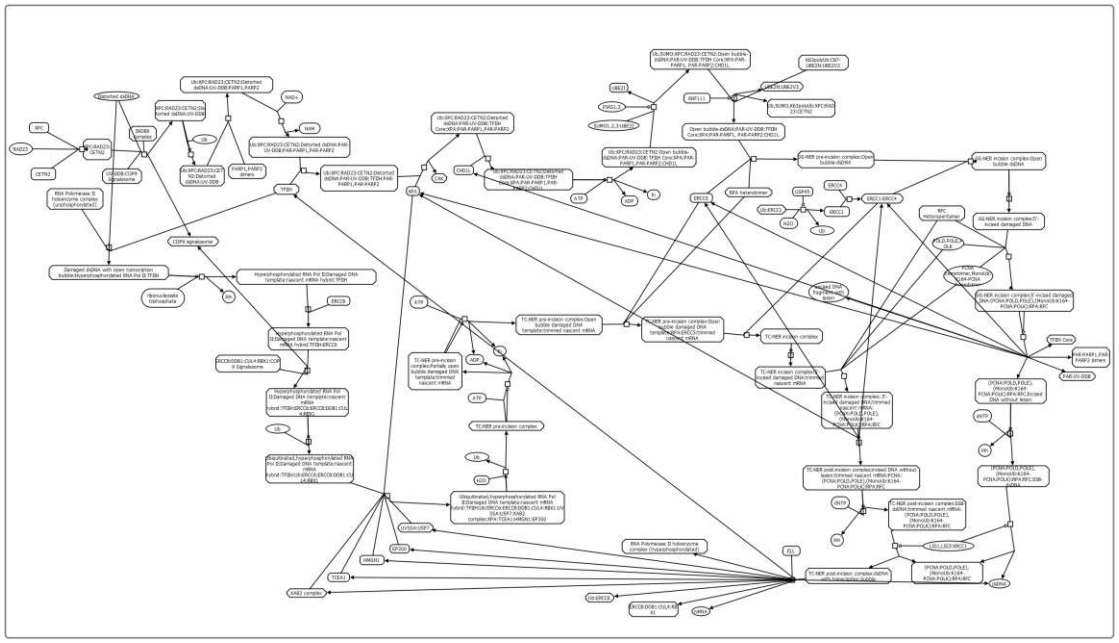
Figure 4.10a shows the original manually-arranged layout of the nucleotide excision repair process. It contains edge bends that are removed in the loaded layout shown in Figure 4.10b. The manual layout as well as the loaded layout already contain edge crossings (# 84). The most pronounced deviation from the SBGN-ness is the process angle deviation M_{PA} with 27.7%. The total deviation of the loaded layout is 25.9%. All quality metrics, including those of all steps of the layout process, are reported in Table 4.4.

To counteract the large node sizes (i.e., the bounding boxes and labels), even before the layout procedure starts, we chose a longer target edge length (120 px) for this particular data set. We then perform our layout procedure. Since the data set consists of only one compartment, we can omit the multilevel layout approach. The initial embedding is calculated with pivot MDS, as presented in Figure 4.11a. Subsequently, we run the unconstrained stress majorization with enabled edge elongation for high-degree nodes (Figure 4.11b). The constraints are applied in the usual order: iterative non-node overlap constraint (Figure 4.12a), process orthogonal constraint (Figure 4.12b), process angle constraint (Figure 4.13a), and finally the modulatory arc constraint (Figure 4.13b). The layout procedure could not substantially remove edge crossings, from 84 for the loaded layout, 50 for the initial layout, and 67 for the final layout. Nevertheless, the overall SBGN-ness of the final layout is better than the loaded and initial one, with 17.3% versus 25.9% and 31.5%, respectively.

The convex hull visualization of the data set is shown in Figure 4.14a and the elastic band visualization is displayed in Figure 4.14b. Its colored version (hue range $[180, 360]$ and fraction 0.75) is presented in Figure 4.15a. The entire metabolic pathway is collapsed in Figure 4.15b.

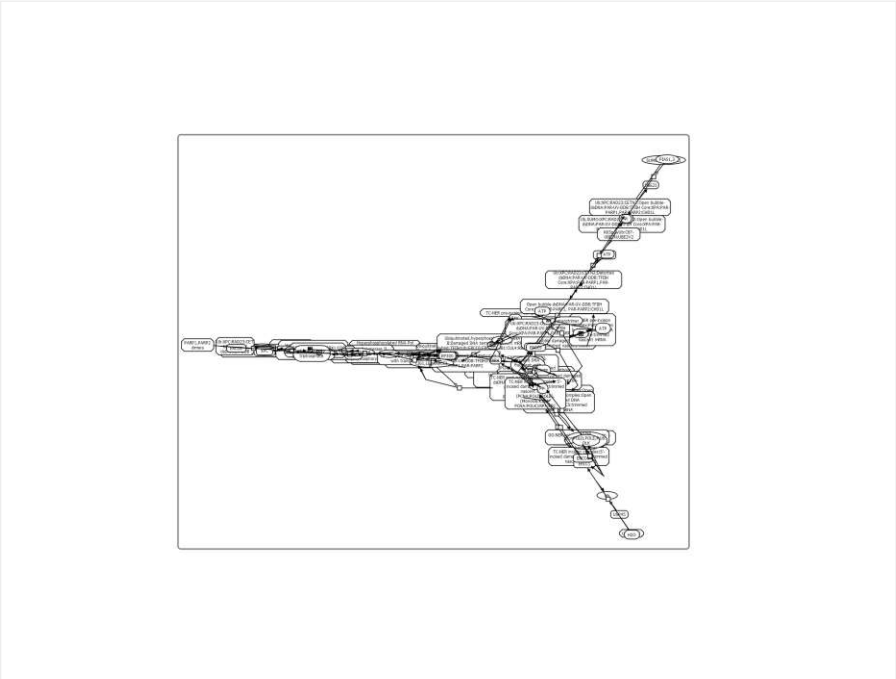


(a) manually-arranged layout (Image taken from [9])

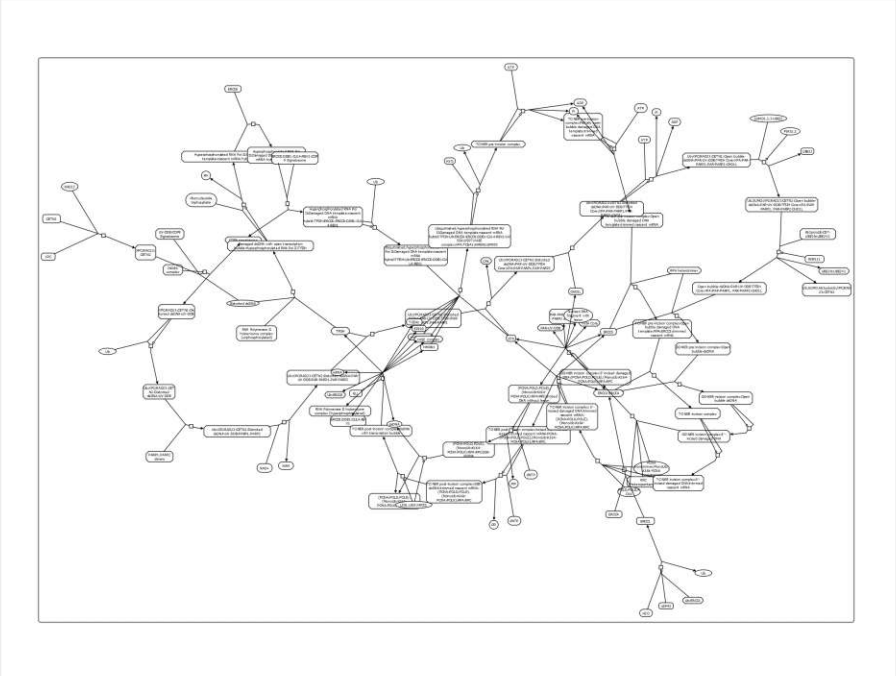


(b) loaded layout

Figure 4.10: Nucleotide excision repair (R-HSA-5696398). (a) shows the manually-arranged layout and (b) shows the same data set loaded with our SBGN viewer.

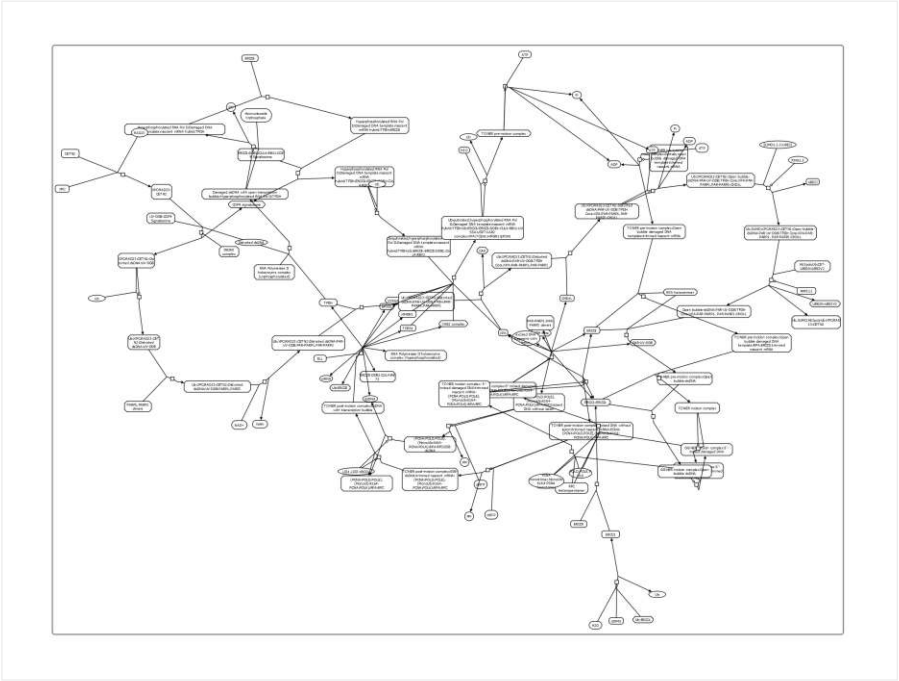


(a) initial embedding

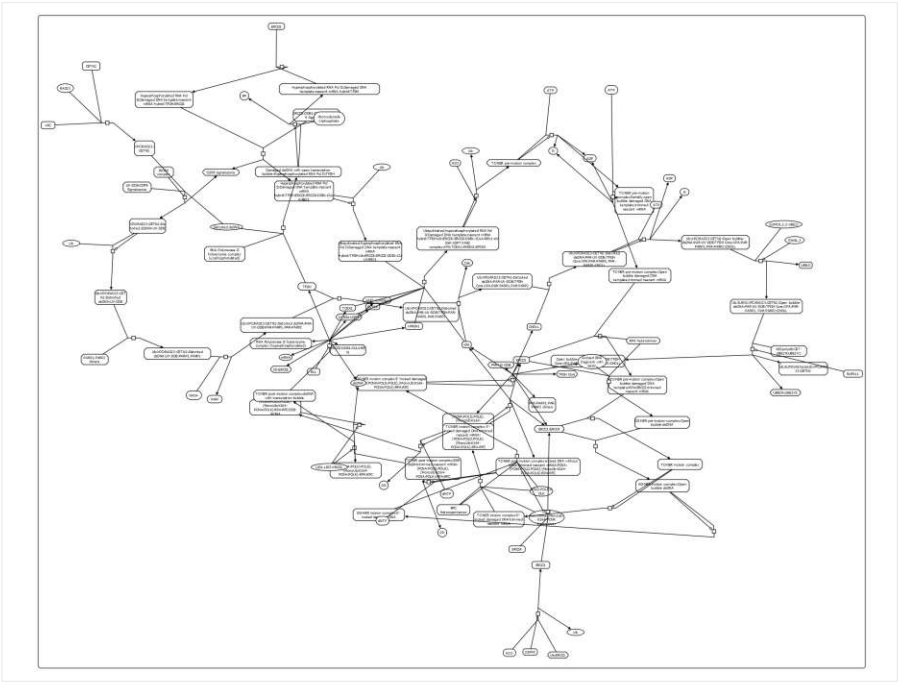


(b) initial layout

Figure 4.11: The initial layout procedure of nucleotide excision repair (R-HSA-5696398). (a) shows the initial embedding calculated with pivot MDS. (b) shows the initial layout created with unconstrained vectorized stress majorization and elongated edges for high-degree nodes.

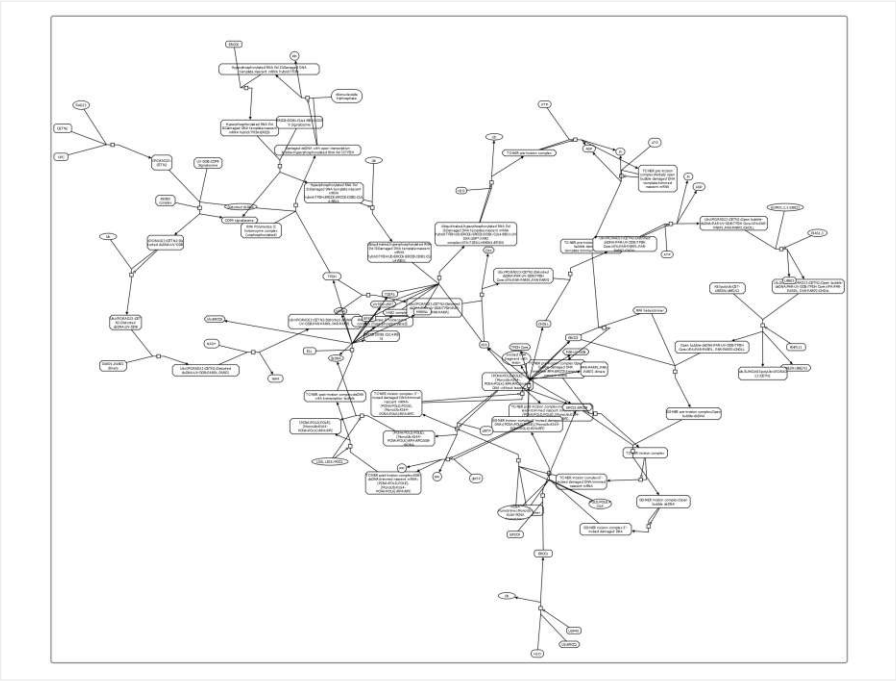


(a) iterative non-node overlap constraint

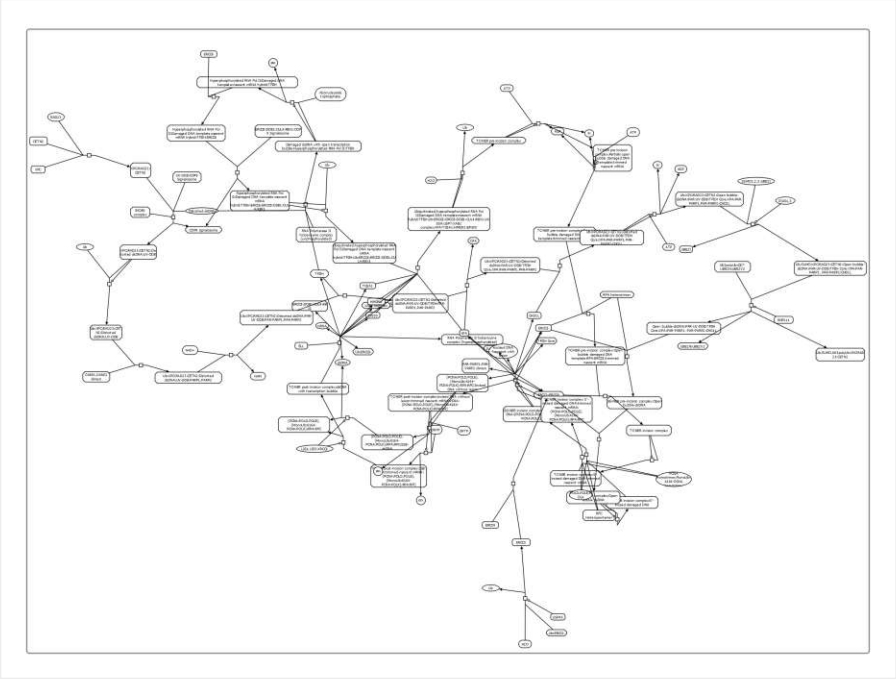


(b) process orthogonal constraint

Figure 4.12: Constraint layout procedure of nucleotide excision repair (R-HSA-5696398). The iterative non-node overlap constraint is applied in (a), followed by the process orthogonal constraint in (b).

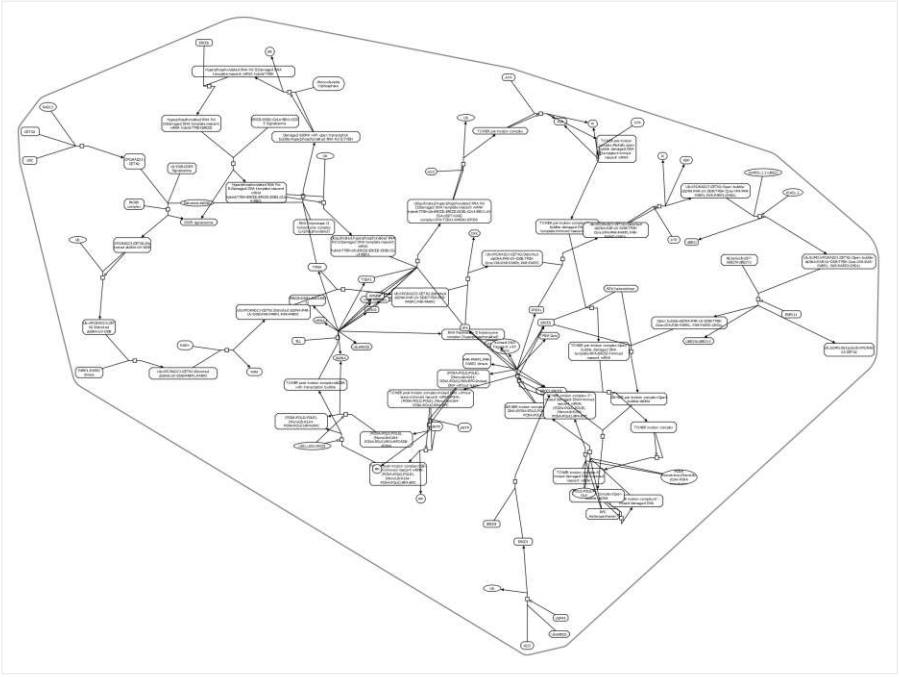


(a) process angle constraint

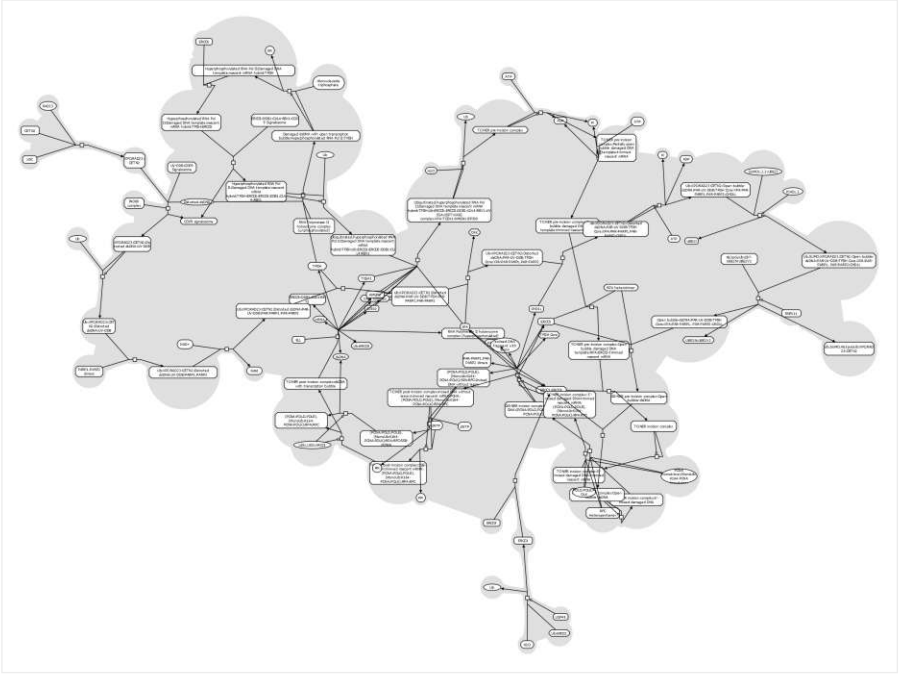


(b) modulatory arc constraint

Figure 4.13: Constraint layout procedure of nucleotide excision repair (R-HSA-5696398). The process angle constraint is applied in (a), followed by the modulatory arc constraint in (b).

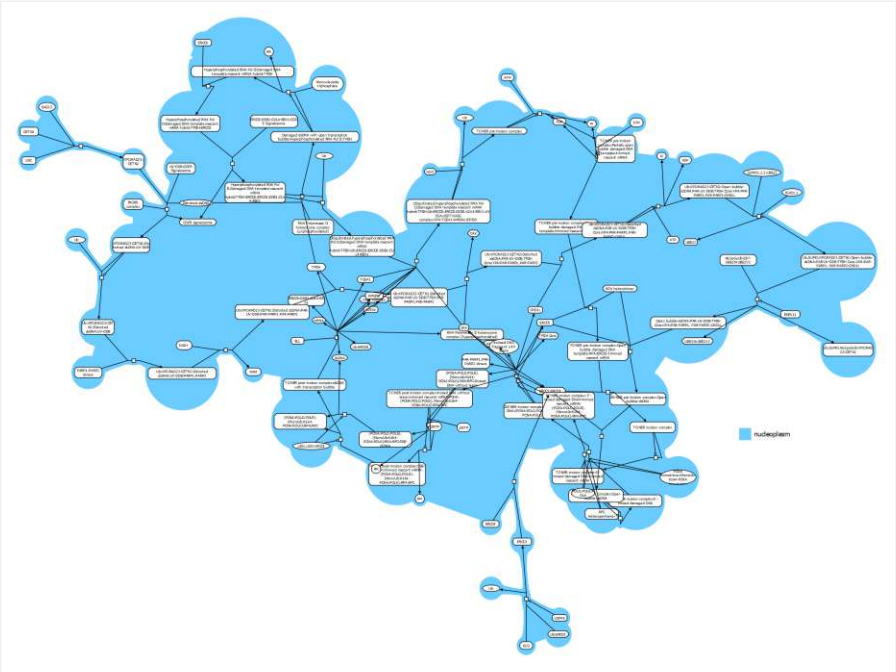


(a) convex hull

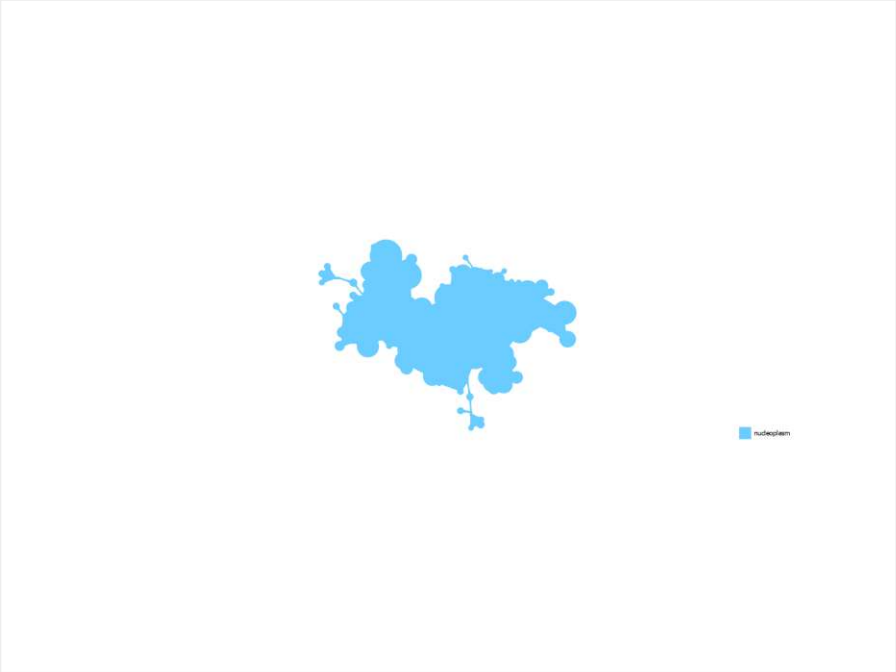


(b) elastic band

Figure 4.14: Visual mapping of nucleotide excision repair (R-HSA-5696398). (a) shows the automatic layout with the convex hull. (b) shows the same layout with our elastic band visualization.



(a) elastic band



(b) collapsed

Figure 4.15: Visual mapping of nucleotide excision repair (R-HSA-5696398). (a) shows the elastic band visualization colored with the tree color approach using a starting hue range of $[180, 360]$ and fraction 0.75. (b) presents the entire metabolic pathway collapsed.

Table 4.4: Parameters and quality metrics of nucleotide excision repair (R-HSA-5696398). The corresponding layouts of the steps described in the columns can be found in Figure 4.10, Figure 4.11, Figure 4.12, and Figure 4.13. For each steps we report the quality metrics (Section 4.1) and the layout's parameters for reproducibility.

# nodes = 214 # edges = 253 Metric	manually-arranged (loaded) layout	initial embedding (pivot MDS)	initial layout	iterative non-node overlap constraint	process orthogonal constraint	process angle constraint	modulatory arc constraint
# node overlaps	3	774	26	18	14	22	18
# edge crossings	84	111	50	82	86	59	67
# node misplacement	0	0	0	0	0	0	0
M_{NM} [%]	0.0	0.0	0.0	0.0	0.0	0.0	0.0
M_{F}	0.16	0.13	0.14	0.17	0.18	0.16	0.14
M_{PO} [%]	15.8	46.5	51.6	48.3	0.5	2.6	2.7
M_{PA} [%]	27.7	33.8	19.2	19.0	28.3	23.7	26.5
$M_{\mathrm{PA}}^{\mathrm{in}}$ [%]	25.2	40.2	29.3	23.7	32.8	30.1	33.8
$M_{\mathrm{PA}}^{\mathrm{out}}$ [%]	30.3	27.4	9.1	14.2	23.8	17.4	19.2
M_{MA} [%]	60.0	95.0	55.0	40.0	45.0	35.0	40.0
M_{total} [%]	25.9	43.8	31.5	26.8	18.5	15.3	17.3
Parameter							
edge elongation	–	–	on	on	on	on	on
edge length [px]	–	–	120	120	120	120	120
handle edge length [px]	–	–	20	20	20	20	20
iterations	–	–	50	50	50	200	200
constraint weight	–	–	–	2	3	10	10

4.6 Discussion and Limitations

The results demonstrate that our approach generally improves the SBGN-ness of metabolite networks. Although the complex interplay of multiple constraints is said to be problematic [65, 86], the results confirm that combining constraints, including their order and weights, works well. The multilevel approach is effective in separating compartments and disconnected components. Therefore, the additional compartment constraint was not necessary for our results. The elastic band visualizations applied to real data resemble 2D embeddings of 3D molecular structures that fit the application domain nicely. The current setup requires several parameters to be specified by user input. However, there is a stable default configuration with minor adjustments that was applicable to all the results. This indicates that our approach is capable of being fully automated in the future. Moreover, our holistic approach seems to be viable.

The current approach has several limitations. We do not support the graph drawing requirements of equal node distribution ~~(R03)~~ and fixed node position ~~(R13)~~. The reason for this is that the vectorized stress majorization framework mostly supports relative constraints formulated through vector directions. The two requirements ~~(R03)~~ and ~~(R13)~~ would need to be formulated as absolute constraints.

Our current shape constraints are limited to simple symmetric shapes because the ICP algorithm is not sufficient to correctly match points of more complex shapes. Since the ICP algorithm is not aware of the point connectivity, an incorrectly matched sequence of points leads to malformed result shapes.

The multilevel layout does not consider disjoint subgraphs and connected subgraphs within the same level. To support this case, we need to extend our layout strategy.

In Subsection 3.4.2 we proposed a technique for edge elongation to resolve *hair balls* in the layout. This technique gets counteracted by the application of constraints in the subsequent layout steps. To solve this problem, the elongation needs to be incorporated into the constraints.

The flow direction metric is quite difficult to interpret. Changing the flow direction metric to report the deviation for each reading direction and reporting the flow direction for each disconnected component could be more informative.

CHAPTER 5

Conclusion and Future Work

In this chapter we conclude the thesis and provide an outline of future work directions.

5.1 Conclusion

In this work we presented a holistic approach for visualizing metabolic pathways in the SBGN. We collected and analyzed the domain-specific requirements and devised a pipeline to complete the endeavor of creating an automatic layout approach.

We demonstrated that the vectorized stress majorization can be customized to generate constraint layouts of SBGN maps, by formulating domain-specific constraints, such as the process orthogonal constraint and the process angle constraint. We addressed the handling of disconnected components and disjoint subgraphs during the layout process by using a multilevel approach. We proposed a visualization approach that is based on elastic bands to form distinctive shapes of subcellular locations. Coupled with an expand and collapse interaction, complex metabolic networks can be simplified to facilitate exploration.

To evaluate the results we proposed several quality metrics that measure the SBGN-ness of a layout. The results and evaluation demonstrated that our approach can faithfully represent SBGN maps.

As concluding remark, we believe that it is more fruitful to consider the automatic creation of SBGN maps as a holistic problem, rather than creating isolated solutions to subproblems that are incompatible with each other.

5.2 Future Work

There are several possible future avenues that we can pursue. The consistency of the input data can be improved. Advanced tools to evaluate and correct missing information,

e.g., compartment references, are needed and can be build upon the LibSBGN's validation tool [101]. The communities' drawing convention of placing nodes on double compartment boundaries is not mapped to the SBGN-ML format. To make this convention usable, an automatic data transformation that preserves the meaning of the SBGN map is needed.

Every step of the pipeline can be improved and extended. As discussed in Subsection 3.4.3, there is potential to formulate further domain-specific layout constraints or improve the ones already devised. To improve the recognition of similar subcomponents and substructures of metabolic pathways, they should be drawn in the same way. There is a need for a global shape catalogue that can be used to recreate layouts for similar structures. These structures could be seamlessly integrated into the vectorized stress majorization as shape constraints. Additionally, global color maps could be defined for recurring subcellular locations to prevent ambiguities and enable fast understanding. The vectorized stress majorization's shape constraint can be improved by replacing the ICP matching through a path-aware matching procedure.

The quality metrics were a first attempt to evaluate metabolic networks in SBGN. In order to assess their usability, a more thorough evaluation must be conducted.

List of Figures

List of Tables

List of Algorithms

Acronyms

ATP Adenosintriphosphat. 8, 15, 37

Cerebral Cell Region-Based Rendering And Layout. 18

CGAL Computational Geometry Algorithms Library. 90, 91

DAG Directed Acyclic Graph. 6, 17

Dig-CoLa Constrained Layout of Digraphs. 21–23, 59

HCL Hue-Chroma-Luminance. 85, 86

ICP Iterative Closest Point. 68, 69, 123, 126

IPSep-CoLa Incremental Procedure for Separation Constraint Layout of graphs. 22, 23, 48

KEGG Kyoto Encyclopedia of Genes and Genomes. 5, 9, 27

LCDE Laplacian Constrained Distance Embedding. 23

MDS Multi Dimensional Scaling. 20, 21, 25, 30, 38, 51–54, 56, 58, 71, 75, 90, 91, 97, 102, 103, 108, 109, 114, 115, 117, 122, 127

MPD Minimal Penetration Depth. 61–63, 127

mRNA messenger Ribonucleic acid. 97

NER Nucleotide Excision Repair. 115

OGDF Open Graph Drawing Framework. 90, 91

PATIKA Pathway Analysis Tool for Integration and Knowledge Acquisition. 20, 28

PTM Post-Translational Modification. 109

ROS Reactive Oxygen Species. 103

SBGN Systems Biology Graph Notation. xi, xiii, 1–4, 9–16, 18–20, 23–25, 27–31, 33–37, 39, 40, 42–44, 46–48, 50, 51, 55–61, 66–69, 75, 78, 79, 85, 90, 92, 94–99, 105, 109–111, 115, 116, 123, 125–127

SBGN-ML Systems Biology Graph Notation - Markup Language. 28, 30, 31, 33–39, 41, 42, 45, 50, 89, 90, 126, 127

SBML Systems Biology Markup Language. 27, 28, 31, 35

TSM Topology-Shape-Metrics. 18, 19, 23, 24, 55

Bibliography

[1] Biomodels. https://www.ebi.ac.uk/biomodels/. Accessed: 2022-11-21.

[2] concaveman-cpp. https://github.com/sadaszewski/concaveman-cpp. Accessed: 2022-11-21.

[3] Libsbgn 2. https://github.com/fbergmann/libSBGN2. Accessed: 2022-11-21.

[4] Newt editor. http://web.newteditor.org/. Accessed: 2022-11-21.

[5] reactome. https://reactome.org/. Accessed: 2022-11-21.

[6] Eukaryotic translation elongation (R-HSA-156842). https://reactome.org/content/detail/R-HSA-156842/, . Accessed: 2023-02-07.

[7] Protein repair (R-HSA-5676934). https://reactome.org/content/detail/R-HSA-5676934/, . Accessed: 2023-02-08.

[8] Protein methylation (R-HSA-8876725). https://reactome.org/content/detail/R-HSA-8876725/, . Accessed: 2023-02-12.

[9] Nucleotide excision repair (R-HSA-5696398). https://reactome.org/content/detail/R-HSA-5696398, . Accessed: 2023-02-13.

[10] Uniprot. https://www.uniprot.org/. Accessed: 2022-11-21.

[11] Vectorized stress majorization. https://github.com/Ideas-Laboratory/vectorized_stress_majorization. Accessed: 2022-11-21.

[12] yed - graph editor. https://www.yworks.com/products/yed. Accessed: 2022-11-21.

[13] J. Abello, S. G. Kobourov, and R. Yusufov. Visualizing Large Graphs with Compound-Fisheye Views and Treemaps. In *International Symposium on Graph Drawing*, pages 431–441. Springer, 2004. doi: 10.1007/978-3-540-31843-9_44.

[14] M. Albrecht, A. Kerren, K. Klein, O. Kohlbacher, P. Mutzel, W. Paul, F. Schreiber, and M. Wybrow. On open problems in biological network visualization. In *International Symposium on Graph Drawing*, pages 256–267. Springer, 2009. doi: 10.1007/978-3-642-11805-0_25.

[15] B. Alper, N. Riche, G. Ramos, and M. Czerwinski. Design study of linesets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2259–2267, 2011. doi: 10.1109/TVCG.2011.186.

[16] D. Archambault, T. Munzner, and D. Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, 2007. doi: 10.1109/TVCG.2007.46.

[17] G. D. Bader, M. P. Cary, and C. Sander. Pathguide: a pathway resource list. *Nucleic Acids Research*, 34(suppl1):D504–D506, 01 2006. ISSN 0305-1048. doi: 10.1093/nar/gkj126.

[18] M. Balzer and O. Deussen. Level-of-detail visualization of clustered graph layouts. In *2007 6th International Asia-Pacific Symposium on Visualization*, pages 133–140. IEEE, 2007. doi: 10.1109/APVIS.2007.329288.

[19] A. Barsky, J. L. Gardy, R. E. Hancock, and T. Munzner. Cerebral: a cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation. *Bioinformatics*, 23(8):1040–1042, 2007. doi: 10.1093/bioinformatics/btm057.

[20] G. Bartel, C. Gutwenger, K. Klein, and P. Mutzel. An experimental evaluation of multilevel layout methods. In *International Symposium on Graph Drawing*, pages 80–91. Springer, 2011. doi: 10.1007/978-3-642-18469-7_8.

[21] T. Batik, S. Terziadis, Y.-S. Wang, M. Nöllenburg, and H.-Y. Wu. Shape-guided mixed metro map layout. *arXiv preprint arXiv:2208.14261*, 2022. doi: 10.48550/arXiv.2208.14261.

[22] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998. doi: 10.5555/551884.

[23] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum*, 36(1):133–159, 2017. doi: 10.1111/cgf.12791.

[24] M. Y. Becker and I. Rojas. A graph layout algorithm for drawing metabolic pathways. *Bioinformatics*, 17(5):461–467, 2001. doi: 10.1093/bioinformatics/17.5.461.

[25] C. Bennett, J. Ryall, L. Spalteholz, and A. Gooch. The Aesthetics of Graph Visualization. In D. W. Cunningham, G. Meyer, and L. Neumann, editors, *Computational*

Aesthetics in Graphics, Visualization, and Imaging, pages 57–64. The Eurographics Association, 2007. ISBN 978-3-905673-43-2. doi: 10.2312/COMPAESTH/COMPAESTH07/057-064.

[26] F. T. Bergmann, S. M. Keating, R. Gauges, S. Sahle, and K. Wengler. SBML Level 3 package: Render, Version 1, Release 1. *Journal of Integrative Bioinformatics*, 15 (1), 2018. doi: 10.1515/jib-2017-0078.

[27] J. O. Berkey and P. Y. Wang. Two-Dimensional Finite Bin-Packing Algorithms. *Journal of the Operational Research Society*, 38(5):423–429, 1987. doi: 10.1057/jors.1987.70.

[28] M. Bertamini and J. Wagemans. Processing convexity and concavity along a 2-D contour: figure-ground, structural shape, and attention. *Psychonomic bulletin & review*, 20(2):191–207, 2013. doi: 10.3758/s13423-012-0347-2.

[29] K. K. Biggar and S. S.-C. Li. Non-histone protein methylation as a regulator of cellular signalling and function. *Nature Reviews Molecular Cell Biology*, 16(1):5–17, Dec. 2014. doi: 10.1038/nrm3915.

[30] B. J. Bornstein, S. M. Keating, A. Jouraku, and M. Hucka. LibSBML: an API library for SBML. *Bioinformatics*, 24(6):880–881, 2008. doi: 10.1093/bioinformatics/btn051.

[31] F. J. Brandenburg, M. Forster, A. Pick, and F. Schreiber. BioPath Visualization of Biochemical Pathways. In *Computer science and biology: Proceedings of the German Conference on Bioinformatics, GCB 2001*, pages 11–15, 2001.

[32] U. Brandes and C. Pich. Eigensolver Methods for Progressive Multidimensional Scaling of Large Data. In *International Symposium on Graph Drawing*, pages 42–53. Springer, 2006. doi: 10.1007/978-3-540-70904-6_6.

[33] A. Bretto. Hypergraph Theory. *An introduction. Mathematical Engineering. Cham: Springer*, 2013.

[34] N. Chondrogianni, I. Petropoulos, S. Grimm, K. Georgila, B. Catalgol, B. Friguet, T. Grune, and E. S. Gonos. Protein damage, repair and proteolysis. *Molecular Aspects of Medicine*, 35:1–71, Oct. 2012. doi: 10.1016/j.mam.2012.09.001.

[35] M. Christmann, M. T. Tomicic, W. P. Roos, and B. Kaina. Mechanisms of human DNA repair: an update. *Toxicology*, 193(1-2):3–34, Nov. 2003. doi: 10.1016/S0300-483X(03)00287-7.

[36] J. D. Cohen. Drawing graphs to convey proximity: An incremental arrangement method. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 4(3): 197–229, 1997. doi: 10.1145/264645.264657.

[37] C. Collins, G. Penn, and S. Carpendale. Bubble Sets: Revealing Set Relations with Isocontours over Existing Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009. doi: 10.1109/TVCG.2009.122.

[38] D. Croft, A. F. Mundo, R. Haw, M. Milacic, J. Weiser, G. Wu, M. Caudy, P. Garapati, M. Gillespie, M. R. Kamdar, et al. The Reactome Pathway Knowledgebase. *Nucleic Acids Research*, 42(D1):D472–D477, 2014. doi: 10.1093/nar/gkx1132.

[39] T. Czauderna, C. Klukas, and F. Schreiber. Editing, validating and translating of SBGN maps. *Bioinformatics*, 26(18):2340–2341, 2010. doi: 10.1093/bioinformatics/btq407.

[40] E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, G. Nisanci, R. Cetin-Atalay, and M. Ozturk. PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. *Bioinformatics*, 18(7):996–1003, 2002. doi: 10.1093/bioinformatics/18.7.996.

[41] E. W. Dijksta. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[42] D. P. Dobkin, E. R. Gansner, E. Koutsofios, and S. C. North. Implementing a general-purpose edge router. In *International Symposium on Graph Drawing*, pages 262–271. Springer, 1997. doi: 10.1007/3-540-63938-1_68.

[43] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir. A Compound Graph Layout Algorithm for Biological Pathways. In J. Pach, editor, *International Symposium on Graph Drawing*, pages 442–447, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31843-9. doi: 10.1007/978-3-540-31843-9_45.

[44] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir. A layout algorithm for undirected compound graphs. *Information Sciences*, 179(7):980–994, 2009. doi: 10.1016/j.ins.2008.11.017.

[45] U. Dogrusoz, A. Karacelik, I. Safarli, H. Balci, L. Dervishi, and M. C. Siper. Efficient methods and readily customizable libraries for managing complexity of large networks. *PloS one*, 13(5), 2018. doi: 10.1371/journal.pone.0197238.

[46] M. Drmota, B. Gittenberger, G. Karigl, and P. A. *Mathematik für Informatik*. Heldermann Verlag, 2007.

[47] C. Dunne and B. Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3247–3256, 2013. doi: 10.1145/2470654.2466444.

[48] T. Dwyer. Scalable, Versatile and Simple Constrained Graph Layout. *Computer Graphics Forum*, 28(3):991–998, 2009. doi: 10.1111/j.1467-8659.2009.01449.x.

[49] T. Dwyer and Y. Koren. Dig-CoLa: directed graph layout through constrained energy minimization. In *IEEE Symposium on Information Visualization*, pages 65–72. IEEE, 2005. doi: 10.1109/INFVIS.2005.1532130.

[50] T. Dwyer and K. Marriott. Constrained stress majorization using diagonally scaled gradient projection. In *International Symposium on Graph Drawing*, pages 219–230, 2008. doi: 10.1007/978-3-540-77537-9_23.

[51] T. Dwyer and L. Nachmanson. Fast Edge-Routing for Large Graphs. In *International Symposium on Graph Drawing*, pages 147–158. Springer, 2010. doi: 10.1007/978-3-642-11805-0_15.

[52] T. Dwyer, Y. Koren, and K. Marriott. IPSep-CoLa: An Incremental Procedure for Separation Constraint Layout of Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):821–828, 2006. doi: 10.1109/TVCG.2006.156.

[53] T. Dwyer, K. Marriott, and M. Wybrow. Integrating Edge Routing into Force-Directed Layout. In *International Symposium Graph Drawing and Network Visualization*, pages 8–19, 2006. ISBN 978-3-540-70903-9. doi: 10.1007/978-3-540-70904-6_3.

[54] T. Dwyer, Y. Koren, and K. Marriott. Constrained graph layout by stress majorization and gradient projection. *Discrete Mathematics*, 309(7):1895–1908, 2009. doi: 10.1016/j.disc.2007.12.103.

[55] J. H. Elder and S. W. Zucker. Evidence for boundary-specific grouping. *Vision research*, 38(1):143–152, 1998. doi: 10.1016/S0042-6989(97)00138-7.

[56] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz—Open Source Graph Drawing Tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001. doi: 10.1007/3-540-45848-4_57.

[57] A. Fabregat, K. Sidiropoulos, P. Garapati, M. Gillespie, K. Hausmann, R. Haw, B. Jassal, S. Jupe, F. Korninger, S. McKay, L. Matthews, B. May, M. Milacic, K. Rothfels, V. Shamovsky, M. Webber, J. Weiser, M. Williams, G. Wu, L. Stein, H. Hermjakob, and P. D'Eustachio. The Reactome pathway Knowledgebase. *Nucleic Acids Research*, 44(D1):D481–D487, 12 2015. ISSN 0305-1048. doi: 10.1093/nar/gkv1351.

[58] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, and G. D. Bader. Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2):309–311, 2016. doi: 10.1093/bioinformatics/btv557.

[59] A. Funahashi, Y. Matsuoka, A. Jouraku, M. Morohashi, N. Kikuchi, and H. Kitano. CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks. *Proceedings of the IEEE*, 96(8):1254–1265, 2008. doi: 10.1109/JPROC.2008.925458.

[60] E. R. Gansner and S. C. North. Improved Force-Directed Layouts. In *International Symposium on Graph Drawing*, pages 364–373. Springer, 1998. doi: 10.1007/3-540-37623-2_28.

[61] E. R. Gansner, Y. Koren, and S. North. Graph Drawing by Stress Majorization. In *International Symposium on Graph Drawing*, pages 239–250. Springer, 2004. doi: 10.1007/978-3-540-31843-9_25.

[62] E. R. Gansner, Y. Hu, and S. Kobourov. GMap: Visualizing graphs and clusters as maps. In *IEEE Pacific Visualization Symposium (PacificVis)*, pages 201–208. IEEE, 2010. doi: 10.1109/PACIFICVIS.2010.5429590.

[63] A. Gardner, L. Autin, B. Barbaro, A. J. Olson, and D. S. Goodsell. CellPAINT: Interactive Illustration of Dynamic Mesoscale Cellular Environments. *IEEE Computer Graphics and Applications*, 38(6):51–66, 2018. doi: 10.1109/MCG.2018.2877076.

[64] R. Gauges, U. Rost, S. Sahle, K. Wengler, and F. T. Bergmann. The Systems Biology Markup Language (SBML) Level 3 Package: Layout, Version 1 Core. *Journal of Integrative Bioinformatics*, 12(2):550–602, 2015. doi: 10.1515/jib-2015-267.

[65] H. Gibson, J. Faith, and P. Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357, 2013. doi: 10.1177/1473871612455749.

[66] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003. doi: 10.1093/bioinformatics/btg015.

[67] R. Jianu, A. Rusu, Y. Hu, and D. Taggart. How to display group information on node-link diagrams: An evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 20(11):1530–1541, 2014. doi: 10.1109/TVCG.2014.2315995.

[68] T. Kamada, S. Kawai, et al. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989. doi: 10.1016/0020-0190(89)90102-6.

[69] M. Kanehisa. The KEGG database. In *'In Silico' Simulation of Biological Processes: Novartis Foundation Symposium*, volume 247, pages 91–103. Wiley Online Library, 2002. doi: 10.1002/0470857897.ch8.

[70] L. D. Kapp and J. R. Lorsch. The Molecular Mechanics of Eukaryotic Translation. *Annual Review of Biochemistry*, 73:657–704, 2004. doi: 10.1146/annurev.biochem.73.030403.080419.

[71] A. Kerren and F. Schreiber. Network visualization for integrative bioinformatics. In *Approaches in Integrative Bioinformatics*, pages 173–202. Springer, 2014. doi: 10.1007/978-3-642-41281-3_7.

[72] S. Kieffer, T. Dwyer, K. Marriott, and M. Wybrow. HOLA: Human-like Orthogonal Network Layout. *IEEE Transactions on Visualization and Computer Graphics*, 22 (1):349–358, 2016. doi: 10.1109/TVCG.2015.2467451.

[73] M. Klimenta. *Extending the Usability of Multidimensional Scaling for Graph Drawing*. PhD thesis, Universität Konstanz, 2012.

[74] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. doi: 10.2307/2033241.

[75] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. doi: 10.1007/BF02289565.

[76] V. Latora, V. Nicosia, and G. Russo. *Complex networks: principles, methods and applications*. Cambridge University Press, 2017.

[77] N. Le Novere, M. Hucka, H. Mi, S. Moodie, F. Schreiber, A. Sorokin, E. Demir, K. Wegner, M. I. Aladjem, S. M. Wimalaratne, et al. The Systems Biology Graphical Notation. *Nature Biotechnology*, 27(8):735, 2009. doi: 10.1038/nbt.1558.

[78] Q. Li. Smooth Piecewise Polynomial Blending Operations for Implicit Shapes. *Computer Graphics Forum*, 26(2):157–171, 2007. doi: 10.1111/j.1467-8659.2007.01011.x.

[79] Z. Liu, T. Itoh, J. Q. Dawson, and T. Munzner. The Sprawlter Graph Readability Metric: Combining Sprawl and Area-aware Clutter. *IEEE Transactions on Visualization and Computer Graphics*, 26(6):2180–2191, 2020. doi: 10.1109/TVCG.2020.2970523.

[80] E. M. Maniadi and I. G. Tollis. Analysis and visualization of metabolic pathways and networks: A hypegraph approach. In *IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 109–112. IEEE, 2014. doi: 10.1109/BHI.2014.6864316.

[81] A. Meidiana, S.-H. Hong, P. Eades, and D. Keim. A Quality Metric for Visualization of Clusters in Graphs. In *International Symposium on Graph Drawing and Network Visualization*, pages 125–138. Springer, 2019. doi: 10.1007/978-3-030-35802-0_10.

[82] H. Mi, F. Schreiber, N. Le Novère, S. Moodie, and A. Sorokin. Systems Biology Graphical Notation: Activity Flow language Level 1. *Nature Precedings*, pages 1–41, 2009. doi: 10.1038/npre.2009.3724.1.

[83] M. B. Monika Wißmann. Anatomy-driven layouting forbrain network visualization. Master's thesis, Technische Universität Wien, Faculty of Informatics, 2022.

[84] S. Moodie, N. Le Novere, E. Demir, H. Mi, and A. Villéger. Systems Biology Graphical Notation: Process Description language Level 1 Version 1.3. *Journal of Integrative Bioinformatics*, 12(2):263, 2015. doi: 10.1515/jib-2015-263.

[85] J.-S. Park and S.-J. Oh. A New Concave Hull Algorithm and Concaveness Measure for n-dimensional Datasets. *Journal of Information Science and Engineering*, 28 (3):587–600, 2012. doi: 10.6688/JISE.2012.28.3.10.

[86] H. Purchase. Which aesthetic has the greatest effect on human understanding? In *International Symposium on Graph Drawing*, pages 248–261. Springer, 1997. doi: 10.1007/3-540-63938-1_67.

[87] H. C. Purchase. Metrics for Graph Drawing Aesthetics. *Journal of Visual Languages & Computing*, 13(5):501–516, 2002. doi: 10.1006/jvlc.2002.0232.

[88] H. Rohn, A. Junker, A. Hartmann, E. Grafahrend-Belau, H. Treutler, M. Klapperstück, T. Czauderna, C. Klukas, and F. Schreiber. VANTED v2: a framework for systems biology applications. *BMC Systems Biology*, 6(1):139, 2012. doi: 10.1186/1752-0509-6-139.

[89] M. Sari, I. Bahceci, U. Dogrusoz, S. O. Sumer, B. A. Aksoy, Ö. Babur, and E. Demir. SBGNViz: a tool for visualization and complexity management of SBGN process description maps. *PloS one*, 10(6):e0128985, 2015.

[90] F. Schreiber. High quality visualization of biochemical pathways in BioPath. *In Silico Biology*, 2(2):59–73, 2002.

[91] F. Schreiber, T. Dwyer, K. Marriott, and M. Wybrow. A generic algorithm for layout of biological networks. *BMC Bioinformatics*, 10:375, 2009. doi: 10.1186/1471-2105-10-375.

[92] M. Siebenhaller, S. S. Nielsen, F. McGee, I. Balaur, C. Auffray, and A. Mazein. Human-like layout algorithms for signalling hypergraphs: outlining requirements. *Briefings in Bioinformatics*, 21(1):62–72, 2020. doi: 10.1093/bib/bby099.

[93] V. Silva and J. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. *Advances in Neural Information Processing Systems*, 15, 2002.

[94] L. Smith, D. Wilkinson, M. Hucka, F. Bergmann, S. Hoops, S. Keating, S. Sahle, and J. Schaff. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core. *Nature Precedings*, pages 1–167, 2010. doi: 10.1038/npre.2010.4959.1.

[95] A. Sorokin, N. Le Novere, A. Luna, T. Czauderna, E. Demir, R. Haw, H. Mi, S. Moodie, F. Schreiber, and A. Villéger. Systems Biology Graphical Notation: Entity Relationship language Level 1 Version 2. *Journal of Integrative Bioinformatics*, 12(2):281–339, 2015. doi: 10.1515/jib-2015-264.

[96] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi: 10.1109/TSMC.1981.4308636.

[97] R. Tamassia. *Handbook of Graph Drawing and Visualization*. CRC press, 2013.

[98] R. M. Tarawaneh, P. Keller, and A. Ebert. A General Introduction To Graph Visualization Techniques. In C. Garth, A. Middel, and H. Hagen, editors, *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*, volume 27 of *OpenAccess Series in Informatics (OASIcs)*, pages 151–164, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-46-0. doi: 10.4230/OASIcs.VLUDS.2011.151.

[99] M. Tennekes and E. de Jonge. Tree Colors: Color Schemes for Tree-Structured Data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2072–2081, 2014. doi: 10.1109/TVCG.2014.2346277.

[100] W. S. Torgerson. Multidimensional scaling: I. Theory and method. *Psychometrika*, 17(4):401–419, 1952. doi: 10.1007/BF02288916.

[101] M. P. Van Iersel, A. C. Villéger, T. Czauderna, S. E. Boyd, F. T. Bergmann, A. Luna, E. Demir, A. Sorokin, U. Dogrusoz, Y. Matsuoka, et al. Software support for SBGN maps: SBGN-ML and LibSBGN. *Bioinformatics*, 28(15):2016–2021, 2012. doi: 10.1093/bioinformatics/bts270.

[102] R. Wang, Y. Perez-Riverol, H. Hermjakob, and J. A. Vizcaíno. Open source libraries and frameworks for biological data visualisation: A guide for developers. *Proteomics*, 15(8):1356–1374, 2015. doi: 10.1002/pmic.201400377.

[103] Y. Wang, Y. Wang, Y. Sun, L. Zhu, K. Lu, C.-W. Fu, M. Sedlmair, O. Deussen, and B. Chen. Revisiting Stress Majorization as a Unified Framework for Interactive Constrained Graph Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):489–499, 2017. doi: 10.1109/TVCG.2017.2745919.

[104] Y.-S. Wang and M.-T. Chi. Focus+Context Metro Maps. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2528–2535, 2011. doi: 10.1109/TVCG.2011.205.

[105] H.-Y. Wu, M. Nöllenburg, and I. Viola. Multi-Level Area Balancing of Clustered Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 28(7):2682–2696, 2020. doi: 10.1109/TVCG.2020.3038154.

[106] H.-Y. Wu, M. Nöllenburg, and I. Viola. Graph Models for Biological Pathway Visualization: State of the Art and Future Challenges. *arXiv preprint arXiv:2110.04808*, 2021. doi: 10.48550/arXiv.2110.04808.

[107] X. Yuan, L. Che, Y. Hu, and X. Zhang. Intelligent Graph Layout Using Many Users' Input. *IEEE Transactions on Visualization and Computer Graphics*, 18(12): 2699–2708, 2012. doi: 10.1109/TVCG.2012.236.

[108] A. Zeileis, K. Hornik, and P. Murrell. Escaping RGBland: Selecting Colors for Statistical Graphics. *Computational Statistics & Data Analysis*, 53(9):3259–3270, 2009. doi: 10.1016/j.csda.2008.11.033.