Eurographics Conference on Visualization (EuroVis) 2018
J. Heer, H. Leitte, and T. Ropinski
(Guest Editors)

Volume 37 (2018), Number 3

SetCoLa: High-Level Constraints for Graph Layout

Jane Hoffswell, Alan Borning, and Jeffrey Heer

Paul G. Allen School of Computer Science & Engineering, University of Washington

Abstract

Constraints enable flexible graph layout by combining the ease of automatic layout with customizations for a particular domain. However, constraint-based layout often requires many individual constraints defined over specific nodes and node pairs. In addition to the effort of writing and maintaining a large number of similar constraints, such constraints are specific to the particular graph and thus cannot generalize to other graphs in the same domain. To facilitate the specification of customized and generalizable constraint layouts, we contribute SetCoLa: a domain-specific language for specifying high-level constraints relative to properties of the backing data. Users identify node sets based on data or graph properties and apply high-level constraints within each set. Applying constraints to node sets rather than individual nodes reduces specification effort and facilitates reapplication of customized layouts across distinct graphs. We demonstrate the conciseness, generalizability, and expressiveness of SetCoLa on a series of real-world examples from ecological networks, biological systems, and social networks.

CCS Concepts

•***Human-centered computing*** → *Graph drawings;*

1. Introduction

By using an appropriate graph layout, node-link diagrams can effectively convey properties of the network structure, such as the hierarchy or network connectedness. Such visualizations are common across many domains, including social networks [Sco88, RST*98, FHH*01, MRV*03, FHD*11], biological systems [BR01, SMO*03, SND05, LK05, KNJ*07, BMGK08, GOB*10], and ecological networks [Lav96, Yod98, CJC03, HKA*04, HSBK*06, BDB*11a, Kea16, BGL16, Kru17]. Graph layouts may also utilize domain-specific properties to emphasize relevant patterns in the data. In a biological pathway, nodes can be layered by their subcellular location to visualize the cellular structure (Figure 1). The *"transcriptionally regulated genes"* layer can further show the outcomes of this network, grouped by molecular function.

Many domain-specific layout techniques address particular needs for customized layouts [GD03, SMO*03, BMGK08, Kea17a, Kea17b]. These techniques leverage common structural properties that are significant to the domain, such as known data hierarchies including cellular structure (Figure 1) or trophic level (Figure 7, 8), as a guiding property of the layout. However, these techniques rarely generalize beyond the domain for which they were designed. Furthermore, many other domains lack customized layout tools despite their potential utility. When a layout technique does not exist for the domain of interest, users must either fit their data to available techniques or design and implement a new algorithm. Creating a customized layout algorithm requires both domain and programming expertise, and so introduces a gap between analysis needs and the techniques available to handle those needs.

An alternative strategy is for the designer to specify constraints on the position of nodes based on domain-specific information.

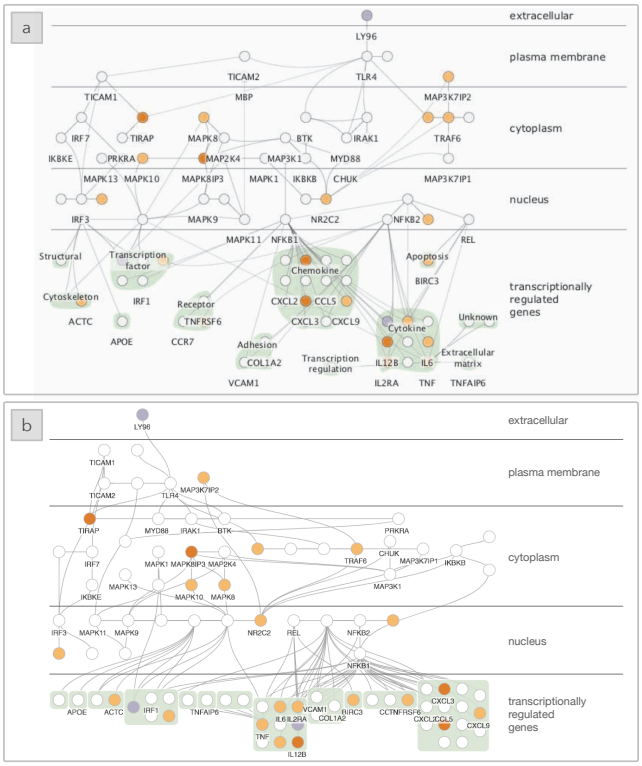


Figure 1: *The layout for the TLR4 biological system produced using (a) Cerebral [BMGK08], a domain-specific layout tool, as compared to (b) SetCoLa. The layers correspond to the location of the biomolecule within a cell and show immune response outcomes at the bottom of the graph, grouped by molecular function.*

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

However, many existing techniques for constraint layout require the user to define constraints on individual nodes or node pairs. This process can be labor intensive, requiring thousands of similar constraints and careful reasoning about which nodes should be constrained to produce the desired layout. Moreover, instance-level constraints (e.g., defined *extensionally* via node indices) prevent reuse of a layout across various graphs from the same domain.

To enable customized domain-specific layouts with reduced programming effort, we contribute SetCoLa: a domain-specific language for specifying high-level constraints for graph layout. Users partition nodes into sets based on node or graph properties, and apply layout constraints to these sets. This approach allows users to specify layout requirements at a high level, deferring the generation of instance-level constraints to the underlying runtime system. These constraint definitions reduce specification effort while enabling highly customized and reusable graph layouts.

We implemented a SetCoLa compiler, which generates instance-level constraints for WebCoLa [Dwy17], a JavaScript library for constraint-based graph layout. To demonstrate the expressiveness of SetCoLa, we recreate several customized layouts found in the scientific research literature. We show that users can compactly specify complex layouts that resemble those produced by custom layout engines, and can reapply these specifications across different graphs. Our SetCoLa specifications can reduce the number of constraints written by the user by one to two orders of magnitude.

2. Related Work

A great deal of prior work has contributed to graph visualization. We discuss general layout techniques and constraint approaches that motivate this work, then expand upon domain-specific layouts.

2.1. General Graph Layout

Graph layout approaches often leverage the underlying structure to produce the layout [HMM00, EGHM10, GFV13]. For node-link diagrams of hierarchical data, Reingold & Tilford's "tidy" layout [RT81] arranges graph nodes into compact, symmetrical tree layouts based on aesthetic properties. Radial layouts [BETT98, HMM00] follow similar procedures using polar coordinates, with a root node placed at the origin. Sugiyama-style layouts [STT81] visualize directed graphs by first assigning nodes to hierarchical layers and then iteratively adjusting node placement to minimize edge crossings. Force-directed techniques [Tut63, QB79, FR91, Kob12] use physical simulation and/or optimization methods that model repulsive forces between nodes and spring-like forces on edges, and attempt to minimize the overall energy. A number of popular tools support graph drawing, including D3.js [BOH11], Gephi [BHJ*09], Graphviz [EGK*01], and Cytoscape [SMO*03].

2.2. Constraint-Based Layout Techniques

Extending an existing layout method to support constraints enables customized layouts that emphasize important structural or aesthetic properties of the graph. Dig-CoLa [DK05] encodes the hierarchy of nodes as constraints and attempts to minimize the overall stress; this technique is a hybrid strategy that combines automatic hierarchical layout with undirected layouts to ensure downward pointing edges. IPSep-CoLa [DKM06] extends force-directed layouts to apply separation constraints on pairs of nodes to support properties such as customized node ordering or downward pointing edges. Dwyer and Robertson [DR09] present a strategy for supporting non-linear constraints (such as circle constraints) that does not constrain node positions along a single axis, and can incorporate these techniques within alternative layout strategies.

Kieffer et al. [KDMW13] present a force-directed, constraint-based layout for creating graphs with node and edge alignment, and demonstrate its effectiveness for interactive refinement within the interactive graph layout system Dunnart [DMW08a]. Dwyer and Wybrow developed libcola [DW18], which utilizes constraints within a force-directed graph layout [DMW08b] using stress majorization. Stress majorization [GKN04] is a technique used for graph layout that has been extended for efficient application on constrained layouts [DM07, WWS*18]. Compound constraints in libcola allow the application of constraints to a list of nodes in the graph and may further introduce dummy nodes for the layout. SetCoLa similarly supports this behavior with constraints defined over sets of graph nodes and further enables hierarhical composition of sets to support expressive, nested layouts.

WebCoLa [Dwy17] is a JavaScript library based on libcola for constraint-based layout in a web-programming context that can be used alongside D3 or Cytoscape. WebCoLa enables constraints on the alignment and position of nodes as well as the specification of high-level properties such as flow (to ensure edges point in the same direction) and non-overlapping constraints. While WebCoLa can support customized constraints, the specification of individual inter-node constraints can be labor intensive. SetCoLa aims to reduce the burden of specifying customized constraints to enable the design of domain-specific and generalizable layouts.

2.3. Domain-Specific Graph Visualization

Several techniques have been developed to reflect domain-specific concerns within graph layouts. However, these techniques tend to be highly-specialized, and so may not apply to other possible domains of interest. For example, ecological networks are a common visualization to include in publications to show the relationships amongst organisms in an ecosystem. Baskerville et al. produced a customized visualization of the Serengeti food web [BDB*11a] in which the nodes are positioned based on their trophic level (e.g., the role of the organism within the larger food chain) and further grouped based on a Bayesian classification of the elements; in addition to the static visualization, Baskerville et al. published an interactive version of the graph online [BDB*11b].

Despite frequently publishing visualizations of oceanic food webs [KSAS12, KSS13], Kearney describes several challenges around the design of such visualizations in a blog post [Kea16]; Kearney notes that the node placement algorithm should *"allow constraining y-position to match trophic level while allowing free movement in the x-direction. With no such algorithm seemingly readily available, I decided to create my own."* Kearney developed plugins for D3 [Kea17a] and Ecopath [Kea17b] to visualize food webs. Motivated by challenges such as those described by Kearney, SetCoLa aims to provide users with a lightweight means for authoring domain-specific constraints for customized layouts.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

Biological systems also benefit from customized visualizations. Cerebral [BMGK08] visualizes biological systems and supports interactive exploration across different experimental conditions. Genc and Dogrusoz [GD03] describe a constrained force-directed layout technique for visualizing biological pathways. Cytoscape [SMO*03] is a visualization system designed to explore biomolecular interaction networks and provides a framework for customized plugins, including a WebCoLa plugin for constraint-based layouts. CrowdLayout [SLML18] introduces a strategy for crowdsourcing biological network layouts from novices that produces more high-quality layouts than Cerebral or Graphviz.

Kieffer et al.'s work on incremental grid layouts [KDMW13] was motivated by related work for grid layouts of biological networks [BMGK08, KNJ*07, LK05], but aims to provide a more flexible mechanism for creating the constraints by supporting SBGN (Systems Biology Graphical Notation). In later work, Kieffer et al. [KDMW16] improve upon grid layout techniques by first identifying the aesthetic criteria humans use for manual graph layout, then producing a new algorithm named HOLA, which employs these techniques for improved, human-like layouts.

Social networks often leverage force-directed layout techniques to demonstrate the connectedness or clustering of the graph nodes [Sco88]. However, some network layouts may introduce additional separation or clustering to highlight properties specific to the social network, such as ethnographically-identified groups [RST*98], the timeline of disease exposure [FHH*01, MRV*03], or differences in reported relationship types [FHD*11].

For each of these domain areas, the visualizations are often created using specially designed tools or layout algorithms to leverage properties of the data specific to the domain of interest. With SetCoLa, we aim to reduce the barrier to creating customized graph layouts by providing a compact way to specify reusable domain-specific layouts that incorporate expert knowledge.

3. Design of SetCoLa

SetCoLa is a domain-specific language for concise specification of constrained graph layouts. To provide a reusable specification without explicit reference to individual nodes and edges, SetCoLa applies constraints to groups of nodes defined by shared attributes. The central abstraction in SetCoLa is a **set**. The simplest elements of a set are graph nodes; however, SetCoLa also supports hierarchical composition, with nested sets as elements.

A SetCoLa specification consists of one or more **constraint definitions**, along with an optional set of guides (reference elements that serve as positional anchors). An example SetCoLa specification for a small tree layout is shown in Figure 2a. Each constraint definition includes a **set definition** and **constraint application**. SetCoLa provides several operators for defining sets based on node attributes and structural relations in the graph. The result of a set definition produces one or more sets, which can have either distinct or overlapping elements. Each constraint definition can define one or more constraints (e.g., for position, ordering, or alignment), which are applied to the nodes within each separate set created by the set definition. In other words, constraints created for a constraint definition with multiple sets are applied to nodes *within* each individual set, not *between* the sets in the constraint definition.

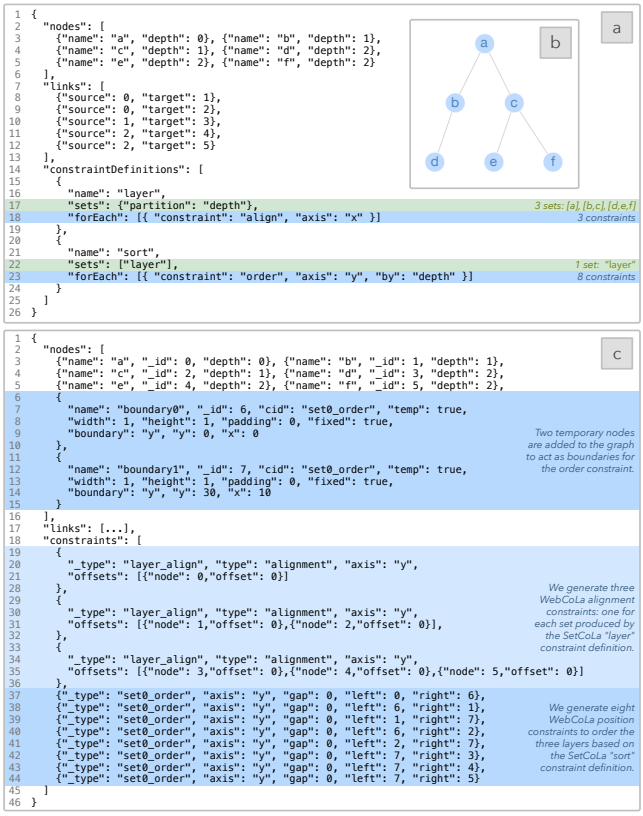


Figure 2: *(a) The SetCoLa specification and data for (b) a tree with six nodes. The nodes are split into three sets based on their* `depth` *and aligned. A new set definition uses composition to include only the "layer" set and orders each layer by its depth to form the tree. (c) The WebCoLa specification created by the SetCoLa compiler.*

The SetCoLa compiler takes an input graph and specification, and produces a set of instance-level constraints for an existing constrained graph layout solver. In this work, we target Dwyer et al.'s WebCoLa library [Dwy17] in order to support interactive, web-based layouts. The SetCoLa compiler generates one or more WebCoLa constraints for each SetCoLa constraint and produces a specification for WebCoLa (Figure 2c). In the following sections, we discuss the design of SetCoLa including the process for specifying sets, the types of constraints currently supported in SetCoLa, and how such constraints are applied over node sets. Our implementation of SetCoLa is available at the link: https://github.com/uwdata/setcola.

3.1. Specifying Sets in SetCoLa

We provide several operators for defining sets based on node attributes or structural properties, which include (1) *partitioning* nodes into disjoint sets, (2) specifying (potentially overlapping) sets via *predicates*, (3) *collecting* nodes into sets based on key expressions, and (4) *composing* previously defined sets. Each of these set definitions produces one or more sets for the constraint definition. For partitions, predicates, or collections, the constraint definition may designate a set from which the elements should be drawn; by default, the elements are simply all graph nodes. For each subsection, we show a sample SetCoLa constraint in the header.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

When new sets are defined, each set propagates the consistent properties of its elements as a single property of its own. For example, in Figure 2a, when the user partitions the nodes based on their `depth` (Line 17), each new set is given a `depth` property the value of which matches all the internal elements. This property can later be referenced in the SetCoLa specification when the sets are treated as elements (e.g., in the order constraint, Line 23).

3.1.1. Partitioning Nodes into Sets

Ex: `partition` by `depth`

The partition operator creates a collection of disjoint sets based on properties of the node (e.g., Figure 2a, Line 17). Given n nodes to partition, this operator can produce at most $O(n)$ sets. The user may further limit the number of sets included by providing specific property values to `include` or `exclude` when producing the sets. For each node, we identify a key based on the node values for the partition properties and create sets based on the key. The include parameter allows the user to identify particular node values to look for when partitioning and includes only those values; the exclude property does the opposite. For example in Figure 5, Line 5, the user partitions the nodes by the `group` property and ignores the nodes for which the value of `group` is *"other."* This parition will produce three sets with the nodes separated by `group`, and the 13 nodes with group *"other"* will not appear in any of these sets.

3.1.2. Specifying Sets with Predicates

Ex: `node.depth == 2`

For more flexibility in the definition of sets, users can specify a concrete list of sets, each defined by an arbitrary boolean expression. For each expression, each node is evaluated by the expression to determine if it should be included in the set. The user may refer to properties of the node using dot syntax; for example, `node.depth` refers to the depth property. In Figure 5, Lines 13, 14, and 15, we define three sets based on the `group` property of the nodes. The set on Line 14 includes nodes that are in the group *"younger white women"* and nodes in the group *"other."* Users may optionally specify a `name` for the set, which may be referred to in subsequent composite set definitions. With this specification strategy, it is possible to create node sets that are not disjoint, and may thus lead to unsatisfiable constraints.

3.1.3. Collecting Nodes Using Keys

Ex: `collect [id(), id(neighbors)]`

To combine the flexibility of *predicates* with the automation of *partition*, users may specify sets as a union based on key expressions. For each element on which the constraint definition is applied, each key expression is evaluated to identify the nodes in that set. For example, in Figure 6, Line 34, the user creates a constraint definition that applies only to nodes with type *"unknown"* (in this case, only one node). On Line 35, the user creates a set definition with two key expressions: one key expression identifies the `_id` of the node (e.g., `node._id`), and the other key expression identifies the `_id`s of the neighboring nodes (e.g., `node.neighbors.extract("_id")`). The one set produced contains the element itself and all its neighbors. We also include several built-in properties for identifying structural relationships in the graph, which are described in Section 3.2.

3.1.4. Composing Previously Defined Sets

Ex: `[ "set1", "set2" ]`

Sets may also be defined as hierarchical compositions of previously defined sets. For example, in Figure 2a the first constraint definition (named "layer") produces three sets via partition (Line 17). The next constraint definition performs composition, referencing only the "layer" set (Line 22): the result is a single set that contains the three layer sets as elements. For composition, users may refer to any named entities previously defined in the specification (e.g., previous set definitions or named sets produced from *predicates*). In the current version of SetCoLa, we only support composition via set union, though future work should explore the types of set definitions that other composition strategies could enable.

3.2. Built-In Properties of the Graph Structure

In addition to defining constraints relative to node properties, it may also be important to define constraints on properties of the graph structure. We support this with a number of built-in accessors. These properties are automatically computed and added to the graph specification only when they are used in one of the SetCoLa constraints. These properties are only computed if such a property does not already exist on the nodes and are subject to a number of expectations regarding the graph input; for graphs that do not meet these expectations, users are shown a warning and required to compute the properties themselves. In this section, we describe each built-in property and discuss the expectations for use.

`_id` The node index in the graph specification. This property is always computed regardless of whether or not it is referenced by the user. The `_id` is a unique identifier and is used to convert the SetCoLa constraints to low-level WebCoLa constraints.

`depth` One more than the max depth of the node's parents. Root nodes (any nodes with no edges for which the node is the target) have a depth of zero. The `depth` property is only computed for graphs that do not contain cycles.

`sources` The list of nodes that have edges for which the current node is the target (e.g., all parent nodes).

`targets` The list of nodes that have edges for which the current node is the source (e.g., all child nodes).

`neighbors` The list of nodes that have edges connected to the current node. This property is the union of the `sources` and `targets` properties. Neighbors may also take an optional value as input, which returns all nodes with a graph distance less than or equal to the specified value.

`incoming` The list of edges in which the current node is the target (e.g., the edges connecting the current node to all sources).

`outgoing` The list of edges in which the current node is the source (e.g., the edges connecting the current node to all targets).

`edges` The list of edges that contain the current node. This property is the union of the `incoming` and `outgoing` edges.

`degree` The number of `neighbors`.

In the original graph specification, the links are defined by a source and target node. However, whether or not these links are directed or undirected is up to the user in how they are treated in the graph layout. For example, properties such as `neighbors` and `edges` are more appropriate than `sources` for undirected graphs. We selected this list of properties as common structural elements applicable to a variety of layout specifications. For example, the

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

Graph	Figure	Nodes	Links	Constraint Definitions	SetCoLa Constraints	WebCoLa Constraints	Ratio (WebCoLa/SetCoLa)
Small Tree	2	6	5	2	2	11	6
Syphilis Social Network	4, 5	41	74	5	9	166	18
TLR4 Network	1, 6	91	124	5	8	363	45
Kruger Food Web	7	16	30	2	3	39	13
Serengeti Food Web	8	161	592	6	18	939	52

Figure 3: *The number of nodes, links, and constraints for each example. The columns labeled **Constraint Definitions** and **SetCoLa Constraints** list the number of definitions or constraints written by the user. We compare the number of **SetCoLa Constraints** to the number of **WebCoLa Constraints** generated by the SetCoLa compiler to determine the factor by which the number of constraints increases (**Ratio**).*

`depth` property is useful for producing hierarchical tree layouts and the `sources`, `targets`, and `neighbors` properties depict the relationship between nodes as dictated by the graph edges. There are many other properties that could be useful for graph layouts that are not included here and this list could easily be extended in the future to include other common properties.

In addition to the built-in properties, we include several operators for manipulating the resulting lists of elements: `length`, `reverse`, `contains`, `sort`, and `extract`. The function `length()` returns the length of the list, `reverse()` reverses the order of the list, and `contains(value)` determines if the list contains the identified value. The user may also choose to `sort` the list based on a property of the elements or `extract` the value of each element for a particular property. Values can also be extracted from nodes or edges individually using the dot syntax at any point. The user may use standard array access to extract elements from the list (e.g., the first element of the list is `list[0]`).

4. SetCoLa Constraints and WebCoLa Implementation

Users may specify one or more constraints for each constraint definition. These constraints apply to the nodes within each set produced by the set definition. The SetCoLa compiler converts each SetCoLa constraint into one or more constraints in Dwyer et al.'s WebCoLa library [Dwy17], which computes the final layout. Figure 3 shows the number of **Constraint Definitions** and **SetCoLa Constraints** written by the user. We compare the number of **SetCoLa Constraints** to the number of **WebCoLa Constraints** generated by the SetCoLa compiler to show the factor by which the number of constraints increases: **Ratio (WebCola/SetCoLa)**. This ratio is a conservative estimate of the impact of SetCoLa, since some SetCoLa constraints are not directly converted to WebCoLa.

In SetCoLa, the user may define guides to control the layout. In WebCoLa, we add a new node to the graph for each guide and generate constraints relative to this node. These temporary nodes are included in WebCoLa's layout but are hidden in the final visualization. WebCoLa constraints are defined based on the `_id` of the graph node. We leverage two of WebCoLa's constraints for our implementation: *alignment* constraints and *position* constraints. For other SetCoLa constraints, we approximate their behavior by imputing additional edges or by applying padding to the nodes. The current implementation of SetCoLa provides seven constraint types: `alignment`, `position`, `order`, `circle`, `cluster`, `hull`, and `padding`. These constraints were selected to produce a range of expressive graph layouts. In this section, we discuss the design, implementation, and utility of each SetCoLa constraint. We show a sample SetCoLa constraint in each section header.

4.1. Alignment Constraints

Ex: `align` *x* `axis`

Alignment constraints ensure that all nodes in the set share one of their coordinates. The user must specify the `axis` as either x or y (Figure 2a, Line `18`) and may also optionally identify an alignment `orientation`. The orientation enables different alignments for elements of varying size. By default, the orientation is defined as *center* and aligns the center point of each element. When the alignment axis is defined as x, the user may specify the orientation as either *top* or *bottom*, which introduces an offset to align the top or bottom of the elements. When the alignment axis is defined as y, the user may specify the orientation as either *right* or *left*.

These constraints are defined as follows. Suppose that the user defines the `axis` as x and the `orientation` as *top*. Then, for all nodes n_1 and n_2 in set S such that $n_1 \neq n_2$, we produce the constraint $n_1.y - n_1.height/2 = n_2.y - n_2.height/2$. Analogous constraints are produced for the other possible combinations of axis and orientation. Alignment constraints are one of the constraint types natively supported in WebCoLa. The WebCoLa alignment constraint takes the `_id` of all nodes that should be aligned and offsets for each node, which can be used to change the orientation.

4.2. Position Constraints

Ex: `position` *right* `of` "top_guide"

Position constraints ensure that all nodes in the set are positioned relative to a guide or previously named set. The user must specify the relative placement for the node as one of *left*, *right*, *above*, or *below* relative to the guide. The user may optionally define the size of the `gap` between the node and guide (Figure 5, Line `7`).

These constraints are defined as follows. Suppose the user defines the `position` as *left*, the guide as g, and the `gap` as v. For all nodes $n \in S_1$, we define a constraint that $n.x + v < g.x$. Position constraints are one of the constraint types natively supported in WebCoLa and are defined by the node `_ids`, `axis`, and desired `gap`. For each node in set S_1, we produce one position constraint relative to the specified guide. When the position constraint is defined relative to a named set S_2, we produce one position constraint for each pair of nodes (u, v) where $u \in S_1$ and $v \in S_2$.

4.3. Order Constraints

Ex: `order` *y* `axis by depth`

Order constraints enforce a sort order on the set elements. The user must specify the `axis` as either x or y and must define the node property `by` which the order is determined (Figure 2a, Line `23`). The user can optionally define an explicit list of values for a custom `order` (Figure 6, Line `20`); otherwise, the elements are ordered lexicographically `by` the specified property. The user may also indicate whether or not to `reverse` the order of the elements.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

These constraints are defined as follows. Suppose the user defines the `axis` as x and the property to sort `by` as `depth`. For all nodes n_1 and n_2 in set S such that $n_1 \neq n_2$ then $n_1.x < n_2.x$ if $n_1.depth < n_2.depth$. We optimize the implementation of this order constraint by only producing constraints between adjacent nodes in the sorted order; in other words, for a set S with n nodes we produce $O(n)$ constraints on the node positions.

When applying constraints to elements that are sets rather than to nodes directly, we create temporary boundary nodes and compute constraints relative to these boundaries. Consider a constraint definition that includes s sets. In this case, we define $s-1$ boundary guides $b_1, b_2, ..., b_{s-1}$. We then identify the order of the sets and produce constraints with the internal nodes for the set. For constraint definition C with s sets, let S_1 and S_2 be two adjacent sets such that $S_1 < S_2$ in the sort order. Let b_1 be the boundary between these two sets. We produce constraints such that for all nodes $n \in S_1$ then $n.x < b_1.x$ and for all nodes $m \in S_2$ then $b_1.x < m.x$. Users may optionally specify a `band` property (Figure 5, Line `18`) that determines a size for each set region to introduce fixed spacing between regions. In this case, we create s + 1 boundary guides and generate additional constraints at the start and end of the ordering.

4.4. Circle Constraints

Ex: `circle around` *center*

Circle constraints allow the user to specify a ring layout for a set of elements. The user must define the value `around` which to compute the layout. This value can be either a default *center* or a previously named guide. The user may optionally define a `radius` that defines the expected radius for the circle (Figure 5, Line `33`).

Circle constraints are not currently supported in WebCoLa. To demonstrate the utility of this constraint type we approximate the behavior in our WebCoLa implementation. To do this, we first add a temporary node or identify the guide to act as the center of the circle layout. We then add a link between each node in the set and the center. Finally, we link the set nodes in the circle with additional temporary edges to produce a chain. We compute the expected length for each edge based on the number of nodes in the circle and the `radius` defined by the user. This strategy approximates a circular layout (Figure 4b), though future work should explore the incorporation of alternative strategies for circle layouts [DR09].

4.5. Cluster Constraints

Ex: `cluster`

Cluster constraints encourage a tight clustering of the nodes into a dense group by aiming to reduce the distance between the nodes. This constraint does not currently introduce addtitional parameters; instead, sets that should be clustered are simply defined as such (Figure 8c, Line `12`). Cluster constraints are not currently supported in WebCoLa. In order to produce a clustered appearance, we add temporary edges between all nodes in the set to produce a clique and require the edges to have a length shorter than the size of the nodes, which pulls the nodes together. These temporary edges remain a part of the layout but are hidden from the user.

4.6. Hull Constraints

Ex: `hull`

Hull constraints create an enclosing boundary (hull) around the set elements and prevent any other nodes from residing within that boundary (Figure 6, Line `28`). This constraint produces a visual grouping of nodes that is more strict than the cluster constraint. These constraints are defined as follows. We produce a minimally enclosing rectangle B with properties $B.x1$, $B.x2$, $B.y1$, $B.y2$. For all nodes $n \in S$, we define constraints such that $B.x1 < n.x$, $n.x < B.x2$, $B.y1 < n.y$, and $n.y < B.y2$. For all nodes $m \notin S$, we define constraints such that $m.x < B.x1 \;||\; B.x2 < m.x$ and $m.y < B.y1 \;||\; B.y2 < m.y$. We implement hull constraints in WebCoLa using its built-in support for specifying groups, which produce a boundary around the nodes defined by their `_id`.

4.7. Padding Constraints

Ex: `padding` with `amount` *5*

Padding constraints enforce a minimum spacing around an element, without constraining the axis on which the padding is added. The user must define the `amount` of padding that should be added to the node (Figure 5, Line `27`). Our current implementation adds padding to the node geometry which essentially increases the size of the element when WebCoLa's non-overlap behavior is applied. In this implementation padding can only be specified to a given node once and impacts the spacing relative to all other nodes in the layout. Additional work is required to develop constraints that respect the padding only relative to certain set elements.

4.8. Application of Multiple Constraints

These constraints enable expressive layouts for several real-world examples. However, not all combinations of constraints produce desirable or satisfiable layouts. The current implementation of SetCoLa does not limit the number or type of constraints that can be applied within a constraint definition. For example, the user could produce contradictions by defining constraints that are the reverse of one another (e.g., two order constraints, one with the ordering reversed). Similarly, applying an alignment constraint to both the x and y axes would require the nodes to share the same position despite overall requirements in WebCoLa to avoid node overlap. These concerns are common in constraint-based systems, and are therefore not limited to SetCoLa. The high-level nature of SetCoLa's constraints can facilitate interpretation of contradictions since the constraints are defined relative to domain-specific properties of the nodes rather than between individual nodes in the graph using only the node `_id`.

While some combinations of constraints produce contradictory or overconstrained layouts, many combinations can produce highly expressive layouts. For example, the small tree in Figure 2 effectively combines node alignment with a total ordering on the sets to produce a simple specification for a tree layout. Position constraints generally allow the user to arrange the layout relative to global elements, whereas order constraints introduce additional sort requirements between nodes within a particular set. Combining multiple (non-contradictory) position constraints allows the user to constrain node positions to particular areas of the visualization, and thus produce overall constraints on the size of the output or to introduce distinct regions of interest based on node properties.

5. Real-World Examples Reproduced in SetCoLa

To demonstrate the conciseness and expressiveness of SetCoLa for domain-specific graph layout, we reproduce several real-world

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

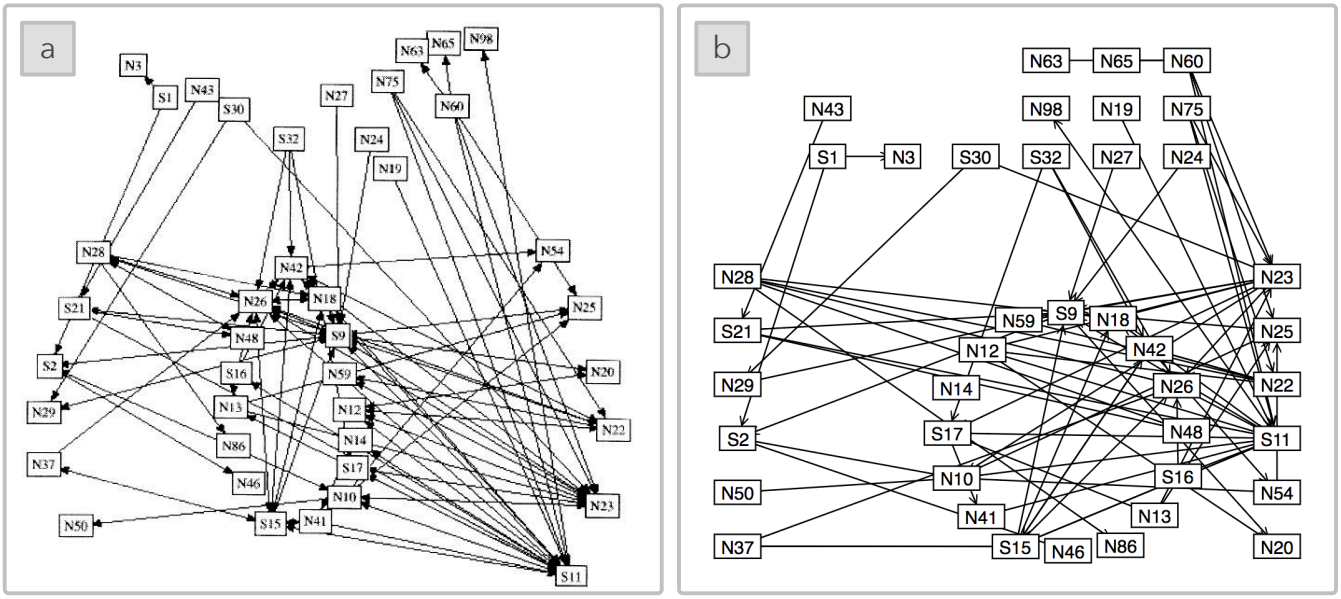


Figure 4: *The layout for the syphilis social network from (a) Rothenberg et al. [RST*98]. (b) We recreated and improved the layout in SetCoLa by introducing additional padding, alignment, and circle constraints to further highlight the relative number of interactions among the different groups. For both figures, the nodes are split into three groups, from left to right: young affluent white men, younger white women, and young African-American men. Individuals not associated with any of these "core" groups are positioned above the others. In both figures, individuals diagnosed with syphilis during the outbreak are labeled with an "S" on the node label.*

```
"guides": [{"name": "top", "y": 50}, {"name": "boundary", "y": 100}, {"name": "bottom", "y": 400}],
"constraintDefinitions": [
  {
    "name": "ethnographic groups",
    "sets": {"partition": "group", "exclude": ["other"]},                                   3 sets
    "forEach": [
      {"constraint": "position", "position": "below", "of": "boundary", "gap": 75},         28 constraints
      {"constraint": "position", "position": "above", "of": "bottom", "gap": 30 }           28 constraints
    ]
  },
  {
    "sets": [[
      {"expr": "node.group === 'young white men'"},
      {"expr": "node.group === 'younger white women' || node.group === 'other'"},          3 sets
      {"expr": "node.group === 'young african american men'"}
    ]],
    "forEach": [
      {"constraint": "order", "axis": "x", "by": "_exprIndex", "band": 300, "gap": 30}     82 constraints
    ]
  },
  {
    "name": "other individuals",
    "sets": {"partition": "group", "include": ["other"]},                                   1 set
    "forEach": [
      {"constraint": "position", "position": "below", "of": "top", "gap": 5},             13 constraints
      {"constraint": "position", "position": "above", "of": "boundary", "gap": 5},        13 constraints
      {"constraint": "padding", "amount": 8}                                              padding added to 13 nodes
    ]
  },
  {
    "name": "women",
    "sets": {"partition": "group", "include": ["younger white women"]},                     1 set
    "forEach": [{"constraint": "circle", "around": "center", "radius": 75}]                 32 new edges added
  },
  {
    "name": "men",
    "sets": {"partition": "group", "exclude": ["younger white women", "other"]},           2 sets
    "forEach": [
      {"constraint": "align", "axis": "y"},                                               2 constraints
      {"constraint": "padding", "amount": 10}                                             padding added to 12 nodes
    ]
  }
]
```

Figure 5: *The SetCoLa specification for the syphilis social network shown in Figure 4. The code is annotated with the number of sets produced (green), the number of WebCoLa constraints generated for the final layout (blue), and the behavior of SetCoLa constraints not directly converted to WebCoLa constraints (purple).*

examples that visualize social networks [RST*98], biological systems [BMGK08], and ecological networks [Kru17, BDB*11a]. We compare our recreated visualizations to the original layouts and discuss the benefits of our technique for creating highly customized graph layouts. For each recreated example, the layout of the nodes is produced entirely in SetCoLa (e.g., no manual tweaking of the node positions). The nodes in each graph are not given intial starting positions; instead we use WebCoLa to first apply a force-directed layout with no constraints before computing the final layout based on the constraints produced by the SetCoLa compiler. We manually added labels to the final figures to better match the originals. We include the specification for each of our three major examples (Figure 5, 6, 8c), and annotate the specification with the number of sets produced for each constraint definition (green), the number of WebCoLa constraints generated for each SetCoLa constraint (blue), and the behavior of SetCoLa constraints that are not directly translated to WebCoLa constraints (purple).

5.1. Syphilis Social Network

Social networks can be a powerful way to understand inter-personal relationships and are useful for tracking the spread of diseases that result from personal contact [RST*98, FHH*01, MRV*03, FHD*11]. The ability to track and identify at risk individuals can lead to treatment and help manage the spread of the disease. In addition to the links between individuals, structuring the layout by node properties such as the social or ethnographically-identified group may reveal additional details about how the disease is spread.

Rothenberg et al. discuss an ethnographic approach to identify the "core" groups in a social network to better understand the transmission of syphilis amongst sexual partners [RST*98]. They found that there were three primary groups involved in the sexual network under study: young affluent white men, younger white women, and young African-American men, which are visualized from left to right in Figure 4a. The authors note that several outsiders to these "core" groups (visualized as the top cluster of Figure 4a) played a significant role in the network: *"Visualization of these groups and all their sex partners uncovered the importance of several people not specifically identified with these groups."*

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

```
"guides": [ {"name": "left_guide", "x": 250}, {"name": "right_guide", "x": 750} ],
"constraintDefinitions": [
 {
  "name": "boundary",                                                    1 set that contains all nodes
  "forEach": [
    {"constraint": "position", "position": "right", "of":"left_guide", "gap": 10},     91 constraints
    {"constraint": "position", "position": "left", "of":"right_guide", "gap": 10}      91 constraints
  ]
 },
 {
  "sets": {"partition": "type", "exclude": ["unknown", "downstream genes"]},          4 sets
  "forEach": [{"constraint": "padding", "amount": 15}]                  padding added to 47 nodes
 },
 {
  "sets": [{"partition": "type", "exclude": ["unknown"]}],                            1 set
  "forEach": [
    {
      "constraint": "order", "axis": "y", "by": "type",
      "band": 100, "gap": 30,                                           180 constraints
      "order": ["extracellular", ..., "downstream genes"]
    }
  ]
 },
 {
  "from": {"expr": "node.type === 'downstream genes'"},
  "sets": {"partition": "group"},                                        12 sets
  "forEach": [
    {"constraint": "hull", "style": "visible"},                          12 WebCoLa groups created
    {"constraint": "cluster"},                                           145 new edges added
    {"constraint": "padding", "amount": 5}                               padding added to 43 nodes
  ]
 },
 {
  "from": {"expr": "node.type === 'unknown'"},
  "sets": {"collect": ["node._id", "node.neighbors.extract('_id')"]},     1 set
  "forEach": [{"constraint": "align", "axis": "x"}]                      1 constraint
 }
]
```

Figure 6: *The SetCoLa specification for the TLR4 biological system shown in Figure 1.*

We reproduced this visualization with SetCoLa (Figure 4b) and included a number of additional constraints on the layout apart from the separation constraints that are visible in the original image. In particular, we included a circle constraint on the group of younger white women to more strongly enforce the result shown in the original figure and applied an alignment constraint on the two groups of young men, as well as some additional padding. The SetCoLa specification is shown in Figure 5.

The simplest recreation of this figure uses three constraint definitions and three SetCoLa constraints to produce 123 WebCoLa constraints. Our modified layout (Figure 4b, Figure 5) includes five constraint definitions, nine SetCoLa constraints, and generates 166 WebCoLa constraints (Figure 3). With a small number of user-defined constraints, we can update the layout to produce one that more effectively communicates the groupings. The alignment and circle constraints emphasize the group relationships by introducing shared visual properties amongst the nodes. The more grid-like layout also facilitates scanning of the nodes. The WebCoLa constraint solver includes a procedure to reduce the length of edges to a user-defined length. This behavior encourages the circle of women to shift towards the group of African-American men, further demonstrating the relatively larger number of interactions between the two groups, as described in the original paper.

5.2. TLR4 Network

Biological networks are a common domain requiring customized visualizations to capture the cellular structure of the nodes in addition to the links contained in the network. Cerebral [BMGK08] is a visualization tool designed to show variations in biological networks across experimental conditions. While such layouts are commonly produced by hand, Barsky et al. show that Cerebral can automatically and efficiently arrange the nodes by the location of the biomolecule within a cell. The immune response outcomes are positioned at the very bottom of the figure and grouped by biological function. The result of Cerebral's layout on the TLR4 network is shown in Figure 1a. Our reproduction in SetCoLa is shown in Figure 1b, with the specification shown in Figure 6.

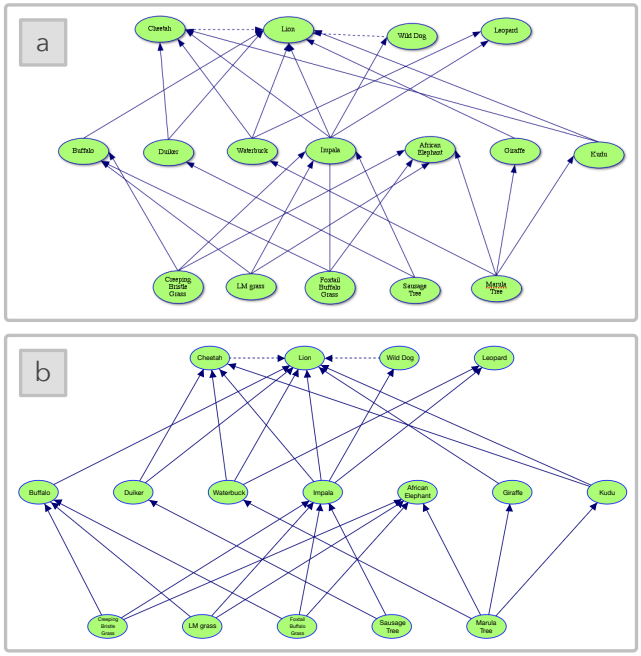


Figure 7: *A subset of the food web for Kruger National park arranged by trophic level (i.e., carnivore, herbivore, and plant), as seen on the website [Kru17] and (b) recreated using SetCoLa.*

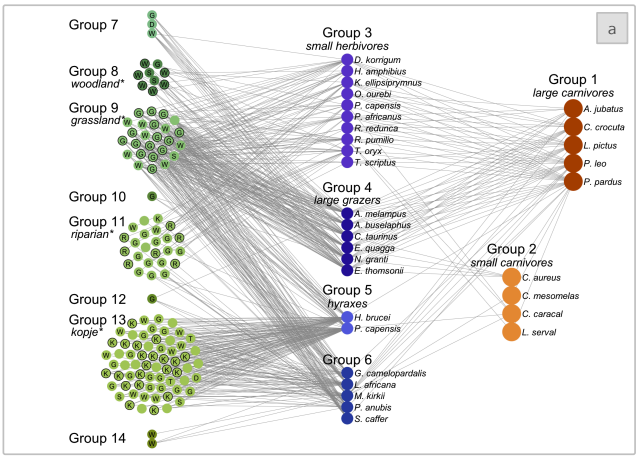
Our recreated layout includes five constraint definitions with eight SetCoLa constraints, which generates 363 WebCoLa constraints (Figure 3). Similar to Cerebral, this SetCoLa specification could easily be reused across different graphs in the domain since the specification itself does not refer to the individual nodes but to the high-level properties desired by the layout. We demonstrate such reapplication across several biological networks from InnateDB [BFL*12] in the supplemental material.

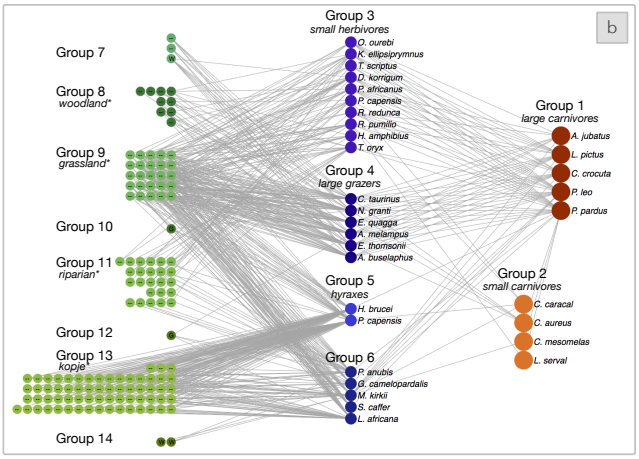
5.3. Serengeti Food Web

Food webs visualize complex producer-consumer relationships in ecological systems. Despite the challenges in creating an informative visualization [Kea16], food webs are a common presentation strategy for this information [Lav96, Yod98, CJC03, HKA*04, HSBK*06, BDB*11a, KSAS12, KSS13, BGL16, Kru17]. Small or simplified food webs may be drawn by hand, but many real world ecosystems can have hundreds of interconnected organisms. In such cases, a customized layout may be useful for reasoning about the structure of the ecological system.

Small food webs exhibit several of the properties of larger food webs, such as a node hierarchy arranged by the node's trophic level (e.g., the element's role within the food web). For example, Figure 7a visualizes a subset of the species found in Kruger National Park [Kru17]. We can easily recreate the layout (Figure 7b) with a small number of constraints on the nodes; for this specification, we include two constraint definitions with three constraints (Figure 3). In particular, we constrain each trophic level to be aligned and enforce an ordering of the layers that respects the food web hierarchy. We also include a constraint to order each layer by a predefined `order` property on the

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.





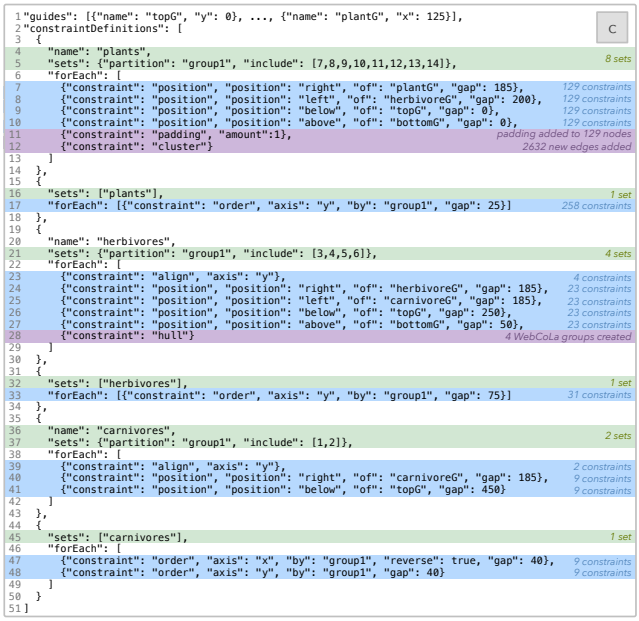


Figure 8: *The layout for the Serengeti food web from (a) Baskerville et al. [BDB*11a] as compared to (b) the layout recreated with SetCoLa. Nodes are grouped by trophic level clusters produced from a Bayesian analysis method. (c) The SetCoLa specification.*

nodes to exactly match the original visualization. This SetCoLa specification could easily be applied to other small food webs to produce a similar layout. However, as the food web gets more complex with more nodes associated with each trophic level, it may become necessary to relax the alignment constraints or introduce additional clustering to highlight other structures within the layout.

One example of a larger ecological network is the Serengeti food web from Baskerville et al. [BDB*11a, BDB*11b], which depicts the relationships among 161 plants, herbivores, and carnivores with 592 links between entities. Baskerville et al. employ a Bayesian analysis method to produce related clusters in each trophic level. They then visualize the results with a customized layout showing both the trophic hierarchy and the group clustering (Figure 8a). The Bayesian analysis approach and customized visualization highlight relationships between the plant habitats and underlying network structure that may be hard to identify from the data alone.

We reproduce this layout in SetCoLa (Figure 8b). For this specification (Figure 8c), we author six constraint definitions that create sets for each group in the layout and sets representing each tropic level. For the carnivores and herbivores, we constrain the position of the nodes within the visualization region, introduce alignments on the nodes, and manage the order with which they are displayed. For the plant sets, we apply cluster constraints to enforce a tighter grouping of the nodes. For this specification, we define a total of 18 SetCoLa constraints, which generate 939 WebCoLa constraints (Figure 3). One noticeable difference in the SetCoLa layout is that the plant nodes (groups `7-14`) form grid-like rather than "organic" clusters. This behavior results from the current implementation of the cluster constraint, which approximates the layout by adding additional edges. The node positions are also impacted by various position constraints, the edges to nodes outside each group, and WebCoLa's non-overlap constraint.

Baskerville et al. note that *"We have not included invertebrates (insects and parasitic helminths) or birds"* in their published food web, though they *"hypothesize that the general conclusions will be largely robust to the addition of more species."* One advantage of SetCoLa is that the layout is independent of the individual nodes, so the authors could reuse the layout specification to visualize future iterations of the Serengeti food web or explore similar structures across different ecological communities.

6. Limitations & Future Work

There are a number of useful areas for future work, including optimizations for the current constraint generation procedure, the development of useful debugging tools to facilitate the user's understanding of unsatisfiable constraints, and the evolution of constraint solvers more closely integrated with SetCoLa.

6.1. Prototyping and Constraint Generation

SetCoLa allows users to define constraints that apply to groups of nodes rather than applying constraints to individual nodes one at a time. By deferring this specification complexity to the underlying constraint solver, the user can more easily prototype the layout and make changes that have a large overall impact with a small number of written constraints. However, the current SetCoLa compiler

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

is not optimized to reduce the number of constraints produced. In particular, the procedure for generating order constraints adds potentially superfluous inter-node constraints. For the small tree example (Figure 2c), the SetCoLa compiler produces 11 WebCoLa constraints. The SetCoLa compiler creates two constraints: $b.y < boundary1.y$ (Line 44) and $c.y < boundary1.y$ (Line 46), in addition to a constraint that $b.y == c.y$ (Line 32-36), thus making one of the first two constraints superfluous. Future work should explore whether these redundant constraints have a significant performance impact, and if they do, investigate optimizations to reduce the generation of unnecessary constraints.

6.2. Debugging and Unsatisfiable Constraints in SetCoLa

SetCoLa specifications may include sets that are *not* disjoint, which can produce unsatisfiable constraints. The user may also specify unsatisfiable constraints indirectly through combinations of set definitions and constraint applications. Finally, some specifications may be under-constrained and thus produce layouts that do not meet the user's expectations. Concerns surrounding debugging and unsatisfiable constraints are not exclusive to SetCoLa, and can also arise in WebCoLa and other constraint-based systems.

For the constraints described in this paper, it is possible to determine if conflicts arise at program runtime and highlight such conflicts. One advantage of the SetCoLa abstraction is that the original user constraints are defined on the high-level properties of the nodes, which makes it easier to understand why conflicts occur. In order to debug the constraints, the user may first inspect the sets produced by SetCoLa to check for inconsistencies. By identifying nodes that exist in multiple sets, users can more easily understand the source of potential conflicts. Each WebCoLa constraint generated by the SetCoLa compiler is annotated with the SetCoLa constraint from which it was generated; these annotations allow the user to map more easily between the output constraints and the original SetCoLa constraints. While these properties may help with the debugging process, future work should explore additional strategies for debugging the graph layout.

6.3. Limitations of SetCoLa's Expressiveness

The current SetCoLa implementation requires the graph to be fully formed at input, including all properties (beyond the ones computed in Section 3.2). All the edges and nodes in this graph are treated with equal weight in terms of the constraints, thus limiting the user's ability to introduce preferences regarding the importance of the nodes or links. Furthermore, there are cases in which the user may want to break links, duplicate parts of the graph, or otherwise modify the underlying structure based on properties of interest. The current SetCoLa implementation does not support operations to modify the importance or structure of the input graph, though this would be an interesting area for future work.

6.4. Limitations Arising from the Constraint Solver

Our implementation with WebCoLa allows us to demonstrate the utility of SetCoLa for creating customized domain-specific layouts that can be reapplied across graphs in the same domain. However, our current implementation was limited in part by what WebCoLa currently supports. For example, we were unable to directly express SetCoLa's circle constraint in WebCoLa. However, circle constraints have been identified in the WebCoLa wiki as an area of future work, and once they are supported in the underlying constraint solver, it should be straightforward to use this improved support. Our work contributes new strategies for the specification of graph layout constraints, but does not aim to create a highly optimized constraint solver. WebCoLa is a useful library on which to build and demonstrate our approach, but future work might explore how new or existing constraint solvers might co-evolve alongside this high-level language for constraint specification.

We also encountered some behavioral mismatches between the implementation of WebCoLa and our expectations for the graph layout. For example, WebCoLa utilizes a default link length for the layout which attempts to optimize node positions to produce links as close to the desired link length as possible. While this technique can be useful for highlighting the underlying structure of the graph, it has a significant effect on the layout that is produced by WebCoLa that may vary from what is specified in SetCoLa. This behavior can be beneficial for some layouts. As noted in Section 5.1, in the syphilis social network (Figure 4b), the circle is drawn slightly off center between the groups since more links exist between the women and the African-American men than between the women and the white men, which emphasizes the strength of these connections. The underlying constraint solver can thus significantly impact the resulting layout by implicitly encoding additional preferences. In future work, it may be useful to support additional parameters expressing global preferences for the graph layout, which would then be passed on to the underlying solver. Another direction would be to accommodate multiple solvers, which might encode different preferences of these kinds, and to select among them, either automatically or as specified by the user.

Another interesting behavior of WebCoLa is that a global non-overlap constraint may be applied to the nodes at the start of the overall layout. This constraint prevents nodes from overlapping even at intermediate stages of the layout and may thus cause the layout to become stuck in a local optimum that still includes unsatisfied constraints. Furthermore, our use of dummy guide nodes with a fixed position may further complicate issues with local maxima. Future work might explore additional procedures for iteratively adding constraints to the graph layout, building up the final result incrementally. This technique would help to reduce some of the burdens on the layout to resolve all constraints at once and allow the user to incrementally improve the layout through the addition of new constraints that restart the underlying solver.

7. Conclusion

We present SetCoLa: a domain-specific language for specifying high-level constraints for customized graph layout. SetCoLa enables concise specification of layouts by applying constraints to node sets rather than individual nodes. These customized layouts can be reapplied to different graphs that share domain-specific properties. We implemented SetCoLa using the WebCoLa library [Dwy17] and demonstrate the expressiveness of SetCoLa on real-world examples from ecological networks, biological systems, and social networks. SetCoLa specifications reduce the number of constraints written by the user by one to two orders of magnitude, while enabling flexible and reusable domain-specific layouts.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

Acknowledgements

We thank the reviewers, the UW Interactive Data Lab, and many others for their helpful comments on this paper. Special thanks to Tim Dwyer, Neil Abernethy, and Brandon Haynes for their feedback. This work was supported by a Moore Foundation Data-Driven Discovery Investigator Award and the National Science Foundation (IIS-1758030).

References

[BDB*11a] BASKERVILLE E. B., DOBSON A. P., BEDFORD T., ALLESINA S., ANDERSON T. M., PASCUAL M.: Spatial guilds in the Serengeti food web revealed by a Bayesian group model. *PLoS Comput Biol 7*, 12 (2011), e1002321. URL: http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002321. 1, 2, 7, 8, 9

[BDB*11b] BASKERVILLE N. B., DOBSON A. P., BEDFORD T., ALLESINA S., ANDERSON T. M., PASCUAL M.: Serengeti food web. http://edbaskerville.com/research/serengeti-food-web/groups-figure3-interactive/, 2011. Accessed: 2017-10-18. 2, 9

[BETT98] BATTISTA G. D., EADES P., TAMASSIA R., TOLLIS I. G.: *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998. 2

[BFL*12] BREUER K., FOROUSHANI A. K., LAIRD M. R., CHEN C., SRIBNAIA A., LO R., WINSOR G. L., HANCOCK R. E., BRINKMAN F. S., LYNN D. J.: InnateDB: systems biology of innate immunity and beyond—recent updates and continuing curation. *Nucleic acids research 41*, D1 (2012), D1228–D1233. 8

[BGL16] BENSON A. R., GLEICH D. F., LESKOVEC J.: Higher-order organization of complex networks. *Science 353*, 6295 (2016), 163–166. 1, 8

[BHJ*09] BASTIAN M., HEYMANN S., JACOMY M., ET AL.: Gephi: an open source software for exploring and manipulating networks. *ICWSM 8* (2009), 361–362. 2

[BMGK08] BARSKY A., MUNZNER T., GARDY J., KINCAID R.: Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *IEEE transactions on visualization and computer graphics 14*, 6 (2008), 1253–1260. 1, 3, 7, 8

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics 17*, 12 (2011), 2301–2309. 2

[BR01] BECKER M. Y., ROJAS I.: A graph layout algorithm for drawing metabolic pathways. *Bioinformatics 17*, 5 (2001), 461–467. 1

[CJC03] COHEN J. E., JONSSON T., CARPENTER S. R.: Ecological community description using the food web, species abundance, and body size. *Proceedings of the National Academy of Sciences 100*, 4 (2003), 1781–1786. 1, 8

[DK05] DWYER T., KOREN Y.: Dig-CoLa: directed graph layout through constrained energy minimization. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on* (2005), IEEE, pp. 65–72. 2

[DKM06] DWYER T., KOREN Y., MARRIOTT K.: IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 821–828. 2

[DM07] DWYER T., MARRIOTT K.: Constrained stress majorization using diagonally scaled gradient projection. In *International Symposium on Graph Drawing* (2007), Springer, pp. 219–230. 2

[DMW08a] DWYER T., MARRIOTT K., WYBROW M.: Dunnart: A constraint-based network diagram authoring tool. In *International Symposium on Graph Drawing* (2008), Springer, pp. 420–431. 2

[DMW08b] DWYER T., MARRIOTT K., WYBROW M.: Topology preserving constrained graph layout. In *International Symposium on Graph Drawing* (2008), Springer, pp. 230–241. 2

[DR09] DWYER T., ROBERTSON G.: Layout with circular and other non-linear constraints using procrustes projection. In *International Symposium on Graph Drawing* (2009), Springer, pp. 393–404. 2, 6

[DW18] DWYER T., WYBROW M.: libcola — overview. http://www.adaptagrams.org/documentation/libcola.html, 2018. Accessed: 2018-03-08. 2

[Dwy17] DWYER T.: cola.js: Constraint-based layout in the browser. http://marvl.infotech.monash.edu/webcola/, 2017. Accessed: 2017-03-12. 2, 3, 5, 10

[EGHM10] EADES P., GUTWENGER C., HONG S.-H., MUTZEL P.: Graph drawing algorithms. In *Algorithms and theory of computation handbook* (2010), Chapman & Hall/CRC, pp. 6–6. 2

[EGK*01] ELLSON J., GANSNER E., KOUTSOFIOS L., NORTH S. C., WOODHULL G.: Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing* (2001), Springer, pp. 483–484. 2

[FHD*11] FU Z., HE N., DUAN S., JIANG Q., YE R., PU Y., ZHAO G., HUANG Z. J., WONG F. Y.: HIV infection, sexual behaviors, sexual networks, and drug use among rural residents in Yunnan Province, China. *AIDS and Behavior 15*, 5 (2011), 1017–1025. 1, 3, 7

[FHH*01] FITZPATRICK L. K., HARDACKER J. A., HEIRENDT W., AGERTON T., STREICHER A., MELNYK H., RIDZON R., VALWAY S., ONORATO I.: A preventable outbreak of tuberculosis investigated through an intricate social network. *Clinical infectious diseases 33*, 11 (2001), 1801–1806. 1, 3, 7

[FR91] FRUCHTERMAN T. M., REINGOLD E. M.: Graph drawing by force-directed placement. *Software: Practice and experience 21*, 11 (1991), 1129–1164. 2

[GD03] GENC B., DOGRUSOZ U.: A constrained, force-directed layout algorithm for biological pathways. In *International Symposium on Graph Drawing* (2003), Springer, pp. 314–319. 1, 3

[GFV13] GIBSON H., FAITH J., VICKERS P.: A survey of two-dimensional graph layout techniques for information visualisation. *Information visualization 12*, 3-4 (2013), 324–357. 2

[GKN04] GANSNER E. R., KOREN Y., NORTH S.: Graph drawing by stress majorization. In *International Symposium on Graph Drawing* (2004), Springer, pp. 239–250. 2

[GOB*10] GEHLENBORG N., O'DONOGHUE S. I., BALIGA N. S., GOESMANN A., HIBBS M. A., KITANO H., KOHLBACHER O., NEUWEGER H., SCHNEIDER R., TENENBAUM D., ET AL.: Visualization of omics data for systems biology. *Nature methods 7* (2010), S56–S68. 1

[HKA*04] HINKE J., KAPLAN I., AYDIN K., WATTERS G., OLSON R., KITCHELL J. F.: Visualizing the food-web effects of fishing for tunas in the pacific ocean. *Ecology and Society 9*, 1 (2004). 1, 8

[HMM00] HERMAN I., MELANÇON G., MARSHALL M. S.: Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics 6*, 1 (2000), 24–43. 2

[HSBK*06] HARPER-SMITH S., BERLOW E. L., KNAPP R. A., WILLIAMS R. J., MARTINEZ N. D.: Dynamic food webs. In *Elsevier Inc.* (2006). 1, 8

[KDMW13] KIEFFER S., DWYER T., MARRIOTT K., WYBROW M.: Incremental grid-like layout using soft and hard constraints. In *Graph Drawing* (2013), pp. 448–459. 2, 3

[KDMW16] KIEFFER S., DWYER T., MARRIOTT K., WYBROW M.: Hola: Human-like orthogonal network layout. *IEEE transactions on visualization and computer graphics 22*, 1 (2016), 349–358. 3

[Kea16] KEARNEY K. A.: Food webs as network graphs. http://kellyakearney.net/2016/01/19/food-webs-as-network-graphs-1.html, 2016. Accessed: 2017-03-14. 1, 2, 8

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

[Kea17a] KEARNEY K. A.: d3-foodweb. https://github.com/kakearney/d3-foodweb, 2017. Accessed: 2017-03-14. 1, 2

[Kea17b] KEARNEY K. A.: foodwebgraph-pkg. https://github.com/kakearney/foodwebgraph-pkg, 2017. Accessed: 2017-03-14. 1, 2

[KNJ*07] KOJIMA K., NAGASAKI M., JEONG E., KATO M., MIYANO S.: An efficient grid layout algorithm for biological networks utilizing various biological attributes. *BMC bioinformatics 8*, 1 (2007), 76. 1, 3

[Kob12] KOBOUROV S. G.: Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011* (2012). 2

[Kru17] Kruger national park: What is a food web? https://kruger-nationalpark.weebly.com/the-food-web.html, 2017. Accessed: 2017-11-38. 1, 7, 8

[KSAS12] KEARNEY K. A., STOCK C., AYDIN K., SARMIENTO J. L.: Coupling planktonic ecosystem and fisheries food web models for a pelagic ecosystem: Description and validation for the subarctic pacific. *Ecological Modelling 237* (2012), 43–62. 2, 8

[KSS13] KEARNEY K. A., STOCK C., SARMIENTO J. L.: Amplification and attenuation of increased primary production in a marine food web. *Marine Ecology Progress Series 491* (2013), 1–14. 2, 8

[Lav96] LAVIGNE D.: Cod food web. http://www.visualcomplexity.com/vc/project.cfm?id=47, 1996. Accessed: 2017-03-14. 1, 8

[LK05] LI W., KURATA H.: A grid layout algorithm for automatic drawing of biochemical networks. *Bioinformatics 21*, 9 (2005), 2036–2042. 1, 3

[MRV*03] MCELROY P., ROTHENBERG R., VARGHESE R., WOODRUFF R., MINNS G., MUTH S., LAMBERT L., RIDZON R.: A network-informed approach to investigating a tuberculosis outbreak: implications for enhancing contact investigations. *The International Journal of Tuberculosis and Lung Disease 7*, 12 (2003), S486–S493. 1, 3, 7

[QB79] QUINN N., BREUER M.: A forced directed component placement procedure for printed circuit boards. *IEEE Transactions on Circuits and systems 26*, 6 (1979), 377–388. 2

[RST*98] ROTHENBERG R. B., STERK C., TOOMEY K. E., POTTERAT J. J., JOHNSON D., SCHRADER M., HATCH S.: Using social network and ethnographic tools to evaluate syphilis transmission. *Sexually transmitted diseases 25*, 3 (1998), 154–160. 1, 3, 7

[RT81] REINGOLD E. M., TILFORD J. S.: Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 2 (1981), 223–228. 2

[Sco88] SCOTT J.: Social network analysis. *Sociology 22*, 1 (1988), 109–127. 1, 3

[SLML18] SINGH D. P., LISLE L., MURALI T., LUTHER K.: Crowdlayout: Crowdsourced design and evaluation of biological network visualizations. In *ACM Human Factors in Computing Systems (CHI)* (2018). URL: http://crowd.cs.vt.edu/wordpress/wp-content/uploads/2018/01/CrowdLayout_CHI_2018.pdf. 3

[SMO*03] SHANNON P., MARKIEL A., OZIER O., BALIGA N. S., WANG J. T., RAMAGE D., AMIN N., SCHWIKOWSKI B., IDEKER T.: Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research 13*, 11 (2003), 2498–2504. 1, 2, 3

[SND05] SARAIYA P., NORTH C., DUCA K.: Visualizing biological pathways: requirements analysis, systems evaluation and research agenda. *Information Visualization 4*, 3 (2005), 191–205. 1

[STT81] SUGIYAMA K., TAGAWA S., TODA M.: Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics 11*, 2 (1981), 109–125. 2

[Tut63] TUTTE W. T.: How to draw a graph. *Proceedings of the London Mathematical Society 3*, 1 (1963), 743–767. 2

[WWS*18] WANG Y., WANG Y., SUN Y., ZHU L., LU K., FU C.-W., SEDLMAIR M., DEUSSEN O., CHEN B.: Revisiting stress majorization as a unified framework for interactive constrained graph visualization. *IEEE transactions on visualization and computer graphics 24*, 1 (2018), 489–499. 2

[Yod98] YODZIS P.: Local trophodynamics and the interaction of marine mammals and fisheries in the benguela ecosystem. *Journal of Animal Ecology 67*, 4 (1998), 635–658. 1, 8

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.