An Online Framework to Interact and Efficiently Compute Linear Layouts of Graphs

Michael A. Bekos, Mirco Haug, Michael Kaufmann, Julia Männecke

Institut für Informatik, Universität Tübingen, Tübingen, Germany
{bekos,mk}@informatik.uni-tuebingen.de
{mirco.haug,julia.maennecke}@student.uni-tuebingen.de

Abstract. We present a prototype online system to automate the procedure of computing different types of linear layouts of graphs under different user-specific constraints. The system consists of two main components; the client and the server sides. The client side is built upon an easy-to-use editor, which supports basic interaction with graphs, enriched with several additional features to allow the user to define and further constraint the linear layout to be computed.
The server side, which is available to multiple clients through a well-documented API, is responsible for the actual computation of the linear layout. Its algorithmic core is an extension of a SAT formulation [5] that is known to be robust enough to solve non-trivial instances in reasonable amount of time. However, it has also several known limitations and potential improvements that we address in this work (e.g., limited applicability to a particular type of linear layouts, no support for additional constraints, limited extendability e.t.c.).
As a proof of concept, we present our findings for a sketch of a proof of an important result in the field that was proposed by Yannakakis [31] back in 1986 (whose details, however, have not been published so far).

1   Introduction

Linear layouts of graphs have been fruitful subjects of intense research over the years, both from a combinatorial and from an algorithmic point of view, as they play an important role in various fields; see, e.g., [16]. Formally, a linear layout of graph $G = (V, E)$ consists of a linear order of its vertices (that is, a bijective function $\sigma : V \to \{1, \ldots, |V|\}$), and a partition of its edges into a particular number of sets. Different constraints on the edges that may reside in the same set, give rise to different types of linear layouts; see, e.g., [1,8,23,27,32]. In particular, there is a rich body of research on two specific types of linear layouts; the stack and the queue layouts.

In a *stack* (*queue*) *layout* of a graph, no two edges of the same set, called *stack* (*queue*) in this context, are allowed to cross (nest, respectively) [23,32], where two edges (u,v) and (w,z), such that $\sigma(u) < \sigma(v)$ and $\sigma(w) < \sigma(z)$, *cross* if $\sigma(u) < \sigma(w) < \sigma(v) < \sigma(z)$, and *nest* if $\sigma(u) < \sigma(w) < \sigma(z) < \sigma(v)$; see, e.g., Fig. 1. The *stack-number* (*queue-number*) of a graph is the minimum number
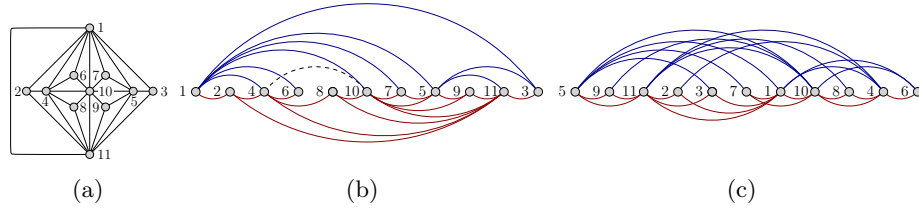
Fig. 1: Illustration of: (a) the Goldner-Harary graph [20], (b) a 3-stack layout of it, and (c) a 2-queue layout of it.

of stacks (queues) required by any of its stack layouts (of its queue layouts, respectively). Note that stack layouts are widely known also as *book embeddings*.

Known Results. There exists a plethora of theoretical results for each of the aforementioned types of linear layouts; in the following, we overview existing results for planar graphs, which is mainly the focus of our work. For a more detailed overview, we point the reader to [16].

– For stack layouts, the most notable result is to due Yannakakis, who back in 1986 showed that every planar graph admits a 4-stack layout [31,32] improving a series of earlier results [9,21,24]. However, a fundamental question, which still remains open in the field, is whether every planar graph admits a 3-stack layout or whether there exists one which requires four stacks in any of its stack layouts. Note that several subfamilies of planar graphs allow for layouts with fewer than four stacks; see, e.g., [4,6,11,12,17,21,25,26,28].
– For queue layouts, a breakthrough result is due to Dujmović et al. [15], who recently showed that every planar graph admits a 49-queue layout, improving previous (poly-)logarithmic bounds [3,14,13]; the best-known corresponding lower bound is four [2], which implies that the current gap between the two bounds is still very large. Again, several subfamilies of planar graphs allow for layouts with significantly fewer than 49 queues; see, e.g., [2,19,22,23,28].

Motivation. The primary motivation of this project stems from the aforementioned paper by Yannakakis, which had appeared at STOC in 1986 [31] and contained a sketch of a proof for the existence of a planar graph that does not admit a 3-stack layout (we provide an outline of this sketch in Section 3.1). The details of this proof, however, never appeared in a paper (in particular, the proof-sketch was not part of the subsequent journal version [32] of the extended abstract that had appeared at STOC [31]), and thus the problem of determining whether there exists a planar graph that requires four stacks in any of its stack layouts still remains unsolved, and clearly forms the most intriguing open problem in the field. Note that the arguments in the proof-sketch by Yannakakis [31] seem to be sound (apart from the fact that some of the gadget-graphs that are central in the proof have not been properly defined), and potentially may give rise to a formal proof.

So, in this work we wanted to dig the details of this proof-sketch, so to understand if (and more importantly where) the proof fails. In addition, we were interested in finding the claimed gadget-graphs, which are not given in the proof. To this end, we decided to continue working on a SAT formulation [5], that we proposed few years ago (for the problem of finding a stack layout of a given graph with a certain number of stacks), which we had to extend with several new features that are of independent value, even for future considerations.

Contribution. The main contribution of this work is a novel system, which is available online (algo.inf.uni-tuebingen.de/linearlayouts) and automates the procedure of computing different types of linear layouts of graphs (i.e., stack layouts, queue layouts or mixtures of these). Besides the description of the system and its functions, we deem important to stress how the extensions that we propose stem from realistic problems that we encountered, while trying to check the different parts of the proof-sketch by Yannakakis [31]. The first major issue that we encountered was the need to define additional constraints on the layouts to be computed (e.g., on the relative order of the vertices, or on the edges to appear at specific pages e.t.c.), that is, without, e.g., writing lines of code tailored to each of these constraints. A second major issue was the need to interact with the graph, that we were investigating, as efficiently as possible. To this end, we featured the system with an easy-to-use graph editor, which supports basic interaction with graphs, and simultaneously provides the necessary functionality to define and further constraint the linear layouts to be computed. Overall, we believe that the real value of the system is that, at its current state, it automates several of the standard procedures that a domain expert needs when interacting with linear layouts of graphs.

Paper Organization. In Section 2, we describe in details the extended features that we have implemented in the system together with some insights on the SAT formulation. Section 3 serves as a proof of concept for our system; we first present an outline of the proof-sketch by Yannakakis mentioned above, and then we present our findings for the different parts of the proof-sketch. We conclude in Section 4 with further considerations and plans.

2   Description of the System

Our system, as introduced in Section 1, consists of two main components, the client and the server sides, and introduces a series of innovations over its previous implementation [29]. The actual code of the system is available to the community at a `github` repository (github.com/linear-layouts/SAT). In the following, we describe in details the extended features that are currently supported both in the client (Section 2.1) and the server side (Section 2.2).

2.1   The Client Side

The client side is web-based, developed with standard tools (e.g., `HTML`, `jQuery` and `yFiles` [30], which is free for academic usage) and operates in two modes;
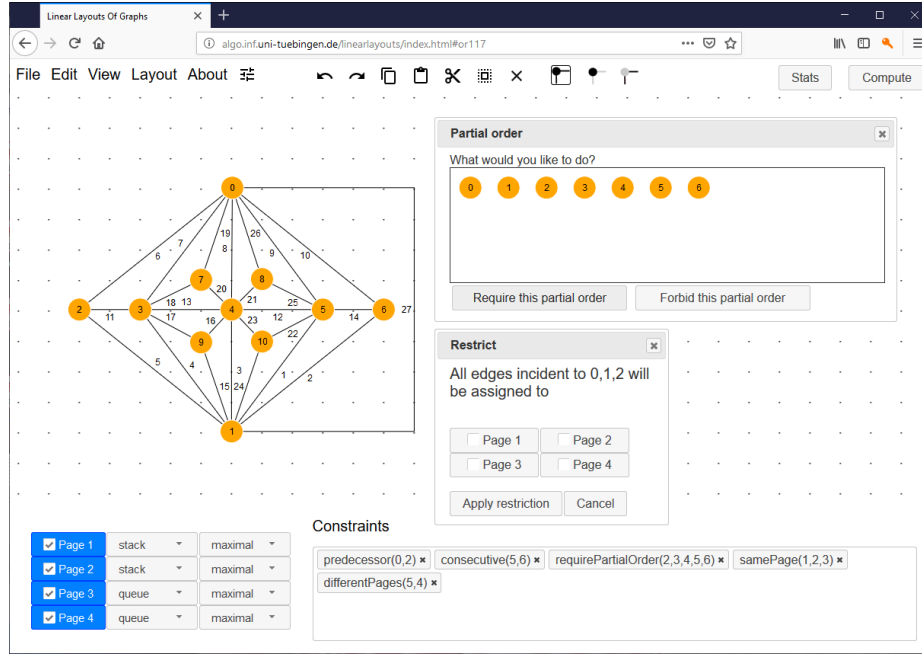
Fig. 2: A screenshot of the system in editing mode.

the *editing* and the *view* (see Figs. 2 and 4). In the editing mode, the user creates
the graph and specifies the constraints of the linear layout to be computed (if
any). The graph is created through a graph editor supporting basic interaction
with graphs (e.g., creation and deletion of vertices and edges, navigation over the
graph, panning, zooming, e.t.c.), that we configured appropriately to meet our
needs. The graph editor is accompanied with a *configuration panel* (refer to the
bottom part of Fig. 2), where the user can configure the type of the layout, as
well as, define different constraints on it. More precisely, the following functions
are currently supported:

Specification of the type of the linear layout. Through the configuration
panel, the user defines the number of available *pages* (i.e., stacks or queues) of
the linear layout to be computed. The current implementation supports up to
four pages, in total. In a subsequent step, the user defines the *type* (i.e., stack or
queue) of each of the pages of the linear layout. In this way, the system provides
support both for stack and queue layouts, but also for *mixed layouts*, in which
some of the pages are stacks, while some others are queues; see, e.g, [27].

Specification of structural constraints. In a stack (queue) layout of a graph,
the subgraphs induced by the edges of each of its stacks (queues) are outerpla-
nar [6] (arched-level planar [23], respectively). Besides the number and the type
of the available pages of the linear layout, the user can also impose additional
structural constraints on these subgraphs. Currently, for each of the available
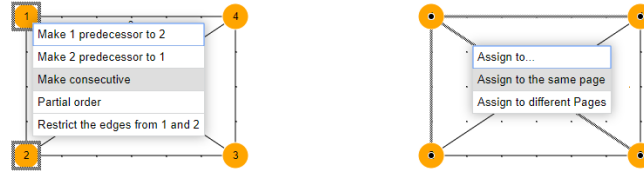
Fig. 3: A snapshot illustrating the options available on selected vertices and edges.

pages, the user can choose one of the following options; not to impose any restriction on the subgraph induced by the edges of a particular page (apart from those that are necessarily imposed by the type of the page), or to restrict the subgraph induced by the edges of a particular page to be either a matching or a tree.

Imposition of restrictions on the linear order. There exist several ways to impose restrictions on the linear order of vertices of the linear layout (see Fig. 3). The constraints are created through the editor and are stored in a separate component of the configuration panel (so that the user is able to remove them).

R.1 **Specification of successors and predecessors of vertices**: By selecting two vertices of the graph and by right-clicking on one of them, the user is able to set one of the selected vertices successor or predecessor of the other.

R.2 **Specification of vertices to be consecutive**: In the same way, the user is able to require two selected vertices to appear consecutively in the linear order of the vertices. Note that this constraint does not restrict the relative order of them. However, this can be easily achieved by combining Restrictions R.1 and R.2.

R.3 **Specification of required and forbidden partial orders.** By clicking on two or more vertices of the graph, while keeping the ctrl button of the keyboard pressed, the user is able to select multiple vertices in a specific order. Then, by right-clicking on one of them the system provides support to restrict the relative order of the selected vertices to be the one (or not to be the one), in which the vertices were clicked on.

Imposition of restrictions on the edge assignments. Besides the restrictions on the linear order of the vertices, the user is also able to assign specific edges to particular pages. This can be achieved through the following functions.

R.4 **Assignment of edges to the same or to different pages**: By selecting two or more edges of the graph and by right-clicking on one of them, the user is able to instruct the system to assign the selected edges to the same or to different pages of the linear layout. Note that the latter option becomes unavailable, when the number of selected edges is greater than the number of available pages of the linear layout.

R.5 **Assignment of edges to specific pages**: In the same way as above, the user is able to assign selected edges of the graph to one of a set of specific

pages of the linear layout. In contrast to Restriction R.4, the user here has to specify the exact pages, to which the selected pages will be assigned.

R.6 **Assignment of edges incident to vertices to specific pages**: The user is also able to assign edges to specific pages through a selection of vertices; in particular, the edges incident to these vertices. In the special case, in which the selected vertices are only two, say u and v, then in the linear order of the vertices, u and v define two intervals; the one between u and v, and the remaining one. In this particular case, the user is also able to apply the constrain only to the edges from u and v that end to only one of these two intervals.

Additional features. The editing mode is equipped with several additional features; in the following, we name few. In order to facilitate the definition of the constraints on the different elements of the graph, the user may restrict the selection mode of the graph editor only to vertices or only to edges, depending on the type of constraints that she wish to introduce. The user is able to save both the graph and its associated constraints in a `GraphML` file for future considerations (see, e.g., `graphml.graphdrawing.org`). There exist also two options that are currently supported for exporting the constructed graph; one in `PDF` and one in `PNG` format. As side features, the editing mode is also equipped with standard layout algorithms (such as, spring-based, orthogonal and radial), while there is also support for querying the graph for standard properties (e.g., connectivity, acyclicity and planarity).

Once the creation of the graph and the definition of the constraints on its linear layout have been completed, the user may request to compute the actual linear layout (if any). At this point, the created graph and its constraints are passed to the server side, which is responsible for the actual computation. Once the computation has been completed, the system enters the view mode, where the computed linear layout is presented to the user. In contrast to the editing mode, in the view mode the user can partially interact with the computed linear layout, e.g., (i) change the common color and the placement (i.e., above or below the line, on which the vertices reside) of the edges assigned to each page, (ii) save or export the final layout to a file, and (iii) navigate, pan or zoom over the layout.

If the user seeks in further editing the graph and its constraints, then she has to return back to the editing mode. In this transition, the user may choose to work either with the original layout that had constructed before or with the computed linear layout (the user's configurations are also restored).

2.2   The Server Side

The server side of the system has been developed in `Python`, while for solving the SAT instances, we used the `Lingeling` solver (`fmv.jku.at/lingeling`); the source code of the server side is also contained in the `github` repository mentioned earlier. All requests that arrive to the server are stored in a database (`SQLite`), which is responsible for associating each of them with a unique id. Once the processing of a request is finished, the computed layout is stored to the
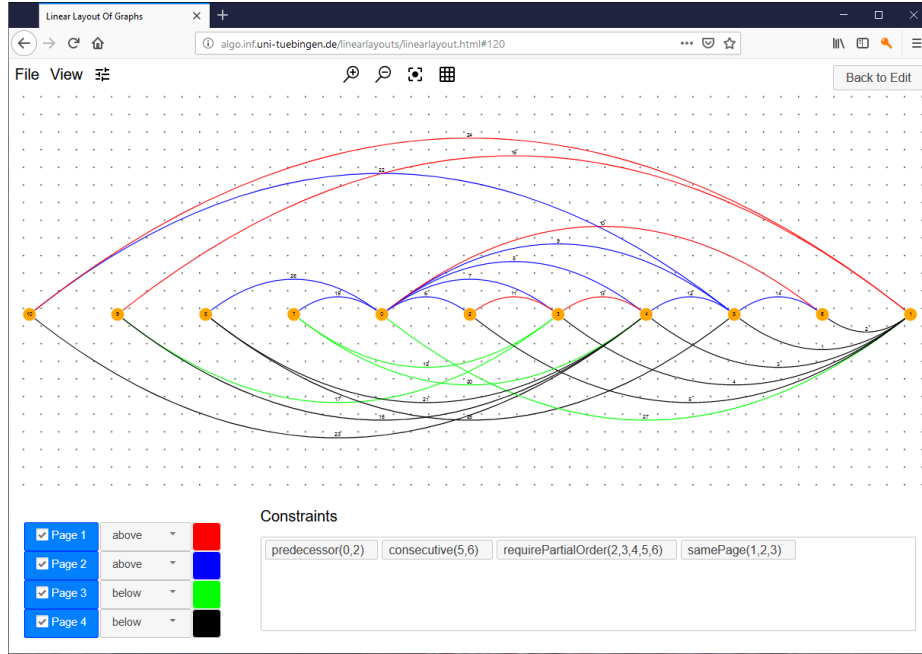
Fig. 4: A screenshot of the system in view mode.

database, so to be available to the client at any time. Note that the server side becomes available to multiple clients through a well-documented API available at the server's interface (`sofa.fsi.uni-tuebingen.de:5555`). In other words, any client, that complies with the developed interface, may communicate with the server side and request a linear layout of a given graph under a set of additional constraints that are currently supported at the server.

The algorithmic core of the server side is an extension of the SAT formulation [5] mentioned in the introduction, featured with additional functions to overcome known limitations; in particular to support different types of linear layouts and different types of user-specific constraints. Even though SAT formulations are of limited applicability, and therefore not so common, in graph drawing (with few notable exceptions; e.g., [7,10,18]), in this particular scenario the formulation is robust enough to solve non-trivial instances in reasonable amount of time (and, as expected, its performance increases when additional constraints are imposed). In the remainder of this section, we give a short overview of the original formulation followed by a high-level description of the extensions that we made.

The original formulation makes use of three types of variables σ, ϕ and χ with the following meanings: (i) for a pair of vertices u and v, variable $\sigma(u,v)$ is `true` if and only if u precedes v in the linear order, (ii) for an edge e and a page ρ, variable $\phi_\rho(e)$ is `true` if and only if edge e is assigned to page ρ of the layout, and (iii) for a pair of edges e and e', variable $\chi(e,e')$ is `true` if and only if e and

e' are assigned to the same page. So, there exist $O(n^2 + m^2 + pm)$ variables, where n denotes the number of vertices of the graph, m its number of edges, and p the number of available pages. A set of $O(n^3 + m^2)$ clauses ensure that the underlying order is linear, and that the layout is valid; for details, refer to [5].

To support different types of layouts, each page ρ is associated with a *type* $\tau(\rho) \in \{\texttt{stack}, \texttt{queue}\}$. If $\tau(\rho) = \texttt{stack}$, then we employ the same set of constraints as in the original formulation to avoid crossings in page ρ. Otherwise, $\tau(\rho) = \texttt{queue}$ holds, in which case we need to guarantee that no two edges of page ρ nest. This can be ensured by introducing the following constraint for every pair of edges (u, v) and (z, w), such that u, v, z and w are pairwise different.

$$
\begin{aligned}
\phi_\rho(u, v) \wedge \phi_\rho(z, w)) \to \ &\neg(\sigma(u, z) \wedge \sigma(z, w) \wedge \sigma(w, v)) \ \wedge \\
&\neg(\sigma(u, w) \wedge \sigma(w, z) \wedge \sigma(z, v)) \ \wedge \\
&\neg(\sigma(v, z) \wedge \sigma(z, w) \wedge \sigma(w, u)) \ \wedge \\
&\neg(\sigma(v, w) \wedge \sigma(w, z) \wedge \sigma(z, u))
\end{aligned}
$$

To ensure Restrictions R.1–R.3, we introduce further constraints. Recall that R.1 and R.2 apply on a pair of vertices, denoted by u and v in the following, while R.3 applies on an ordered set of vertices, denoted by $\{u_1, \ldots, u_k\}$.

R.1: $\sigma(u, v)$, if u must be the predecessor of v
$\quad\quad \sigma(v, u)$, otherwise

R.2: $\sigma(x, u) \vee \sigma(v, x), \ \forall x \notin \{u, v\}$

R.3: $(\sigma(u_1, u_2) \wedge \ldots \wedge \sigma(u_{k-1}, u_k))$, if the order must be preserved
$\quad\quad \neg(\sigma(u_1, u_2) \wedge \ldots \wedge \sigma(u_{k-1}, u_k))$, if the order must be avoided

Restrictions R.4–R.6 are ensured in a similar fashion. Recall that R.4 and R.5 apply on an set of edges, denoted by $\{e_1, \ldots, e_\ell\}$ in the following, while R.6 on an set of vertices, denoted by $\{u_1, \ldots, u_k\}$.

R.4: $\chi(e, e'), \forall e, e' \in \{e_1, \ldots, e_\ell\}$, if the edges must be in the same page
$\quad\quad \neg\chi(e, e'), \forall e, e' \in \{e_1, \ldots, e_\ell\}$, if the edges must be in different pages

R.5: $\neg\phi_\rho(e), \forall e \in \{e_1, \ldots, e_\ell\}$, for each page ρ that is not selected

R.6: $\neg\phi_\rho((u, \cdot)), \forall u \in \{u_1, \ldots, u_k\}$, for each page ρ that is not selected

3 Proof of Concept

In this section, we demonstrate how the system can be used by applying it to the proof-sketch by Yannakakis [31] that we mentioned in the introduction of the paper. At first, we give a short overview of this proof-sketch (Section 3.1), and then we described how we exploited the different functions of the system that we presented in Section 2 to get meaningful insights of this proof-sketch (Section 3.2). We note at this point that the planar graph that Yannakakis claimed not to admit a 3-stack layout is a tremendously large graph, since it is constructed by composing several smaller graphs, each of which has a certain property. Hence, testing via SAT (or via another such approach) whether this final graph admits a 3-stack layout becomes realistically an impossible task.

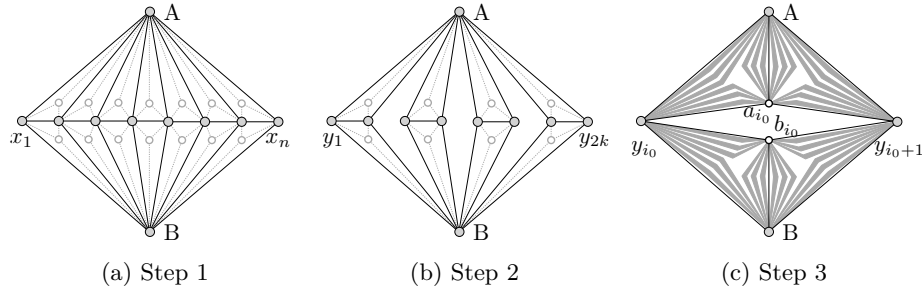(a) Step 1   (b) Step 2   (c) Step 3

Fig. 5: Illustrations for the different steps of proof: (a) the skeleton graph, (b) the stellated-skeleton, and (b) the operation of attaching three copies of graph Q along the edges of the subgraph of the stellated-skeleton induced by A, y_{2i_0-1}, a_{i_0}, b_{i_0}, y_{2i_0} and B.

3.1 An Overview of the Proof-Sketch by Yannakakis

In this section, we give an overview of the aforementioned proof-sketch [31] by Yannakakis that appeared as an extended abstract of [32] at STOC back in 1986. We deem important to stress at this point that since several parts of the proof-sketch have been omitted in [31], the description of the main ideas of it are subject to the best of our understanding of the line of the arguments.

The proof-sketch consists of three steps, each of which assumes the existence of a particular planar graph with a specific property. The graph of the first step, which we call *skeleton*, consists of two designated vertices A and B that are connected to all the vertices of a long path $x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n$, which we call *skeleton path*; for an illustration, refer to Fig. 5a. The property of this graph is essentially the following: if n is chosen large enough (e.g., $n = 1000$), then in any 3-stack layout of the skeleton graph at least one pair of consecutive vertices of the skeleton path will appear between A and B. The observation that gives rise to the second step of the proof-sketch is that if one triples the length of the skeleton path, then one may assume two pairs of consecutive vertices of the skeleton path to lie between A and B, and so on. In other words, if n is chosen large enough, then there exists a particular number of pairs of consecutive vertices of the skeleton path, say $\langle y_1, y_2 \rangle \ldots \langle y_{2k-1}, y_{2k} \rangle$, that lie between A and B in any 3-stack layout of the skeleton graph.

With this observation in mind, in the second step of the proof-sketch, the skeleton graph gets further augmented by *stellating* each of its $2n$ triangular faces, that is, by adding a vertex in its interior and by connecting it to the three vertices of its boundary (refer to the gray-colored vertices in Fig. 5a). Now, focus on the subgraph of the resulting graph, which we call *stellated-skeleton*, that is induced by A, B, $y_1, \ldots, y_{2k}$ and the vertices $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ that stellate the faces $\langle A, y_1, y_2 \rangle \ldots \langle A, y_{2k-1}, y_{2k} \rangle$ and $\langle B, y_1, y_2 \rangle \ldots \langle B, y_{2k-1}, y_{2k} \rangle$, respectively; see Fig. 5b. The property of the stellated-skeleton is essentially the following. If (n is chosen sufficiently large such that) k is large enough, then there exists a 4-tuple $\langle y_{2i_0-1}, a_{i_0}, b_{i_0}, y_{2i_0} \rangle$ of vertices that appear in this order between A and B in any 3-stack layout of the graph constructed so far.

The third step of the proof-sketch yields another augmentation to the graph, which is based on an internally-triangulated planar graph, denoted by Q in [31], whose outer face is bounded by a 4-cycle (s, a, t, b). In particular, three copies of graph Q are "attached" along each edge (u, v) of the graph that has been constructed so far, such that vertices s and t of each of the three copies are identified with the endvertices u and v of the edge (u, v); see Fig. 5c for an illustration of this operation along the edges of the subgraph of the stellated-skeleton induced by A, y_{2i_0-1}, a_{i_0}, b_{i_0}, y_{2i_0} and B.

Note that graph Q is not specified in the proof-sketch; Yannakakis only describes the properties for this graph, which are as follows. Graph Q does not admit a 3-stack layout such that (P.1) a and b lie between s and t in the linear order of the vertices, and (P.2) none of the edges from s and t to the vertices that lie between s and t in the linear order of the vertices (i.e., including a and b) can be assigned to the third stack of the layout.

The three steps described so far formed the difficult part in the proof-sketch; the remainder of it is relatively easier. More precisely, assuming that the 4-tuple $\langle y_{2i_0-1}, a_{i_0}, b_{i_0}, y_{2i_0} \rangle$ of the second step of the proof-sketch is guaranteed, and that the graph Q of the corresponding third step (with the claimed Properties P.1 and P.2) exists, the rest of the proof is sound (and more importantly described in the sketch). Hence, putting all the pieces of this proof together reduces in determining whether the 4-tuple $\langle y_{2i_0-1}, a_{i_0}, b_{i_0}, y_{2i_0} \rangle$ as well as the graph Q exist.

3.2 Our Findings

As already mentioned in the introduction, most of the extensions that we implemented in our system are motivated by our efforts to check the different parts (in particular, the three main steps) of the proof-sketch by Yannakakis. This will become clear in this section, where we report our findings and describe how the developed system helped us to obtain meaningful insights of this proof.

Our findings on the first step of the proof-sketch. The task here was rather clear; we had to check whether two consecutive vertices of the skeleton path will inevitably appear between the two designated vertices A and B of the skeleton graph in any of its 3-stack layouts, when the length of the skeleton path is large enough. To check this, we created an instance of the skeleton graph with approximately 100 vertices. Then, using restriction R.1, we instructed our system to constrain the layout to be computed, such that: (i) vertex A (vertex B) is a predecessor (successor, respectively) of every odd-indexed vertex of the skeleton path, and (ii) vertex A (vertex B) is a successor (predecessor, respectively) of every even-indexed vertex of the skeleton path. If the system could not report a solution under these two constraints, then the claim would follow. However, this was not the case, as the system could easily report a solution (`algo.inf.uni-tuebingen.de/linearlayouts/linearlayout.html#63`), which was so symmetric that we were able to generalize it to all skeleton graphs; see Fig. 6.

Our findings on the second step of the proof-sketch. Even though our findings on the first step of the proof-sketch were a bit discouraging, we found it
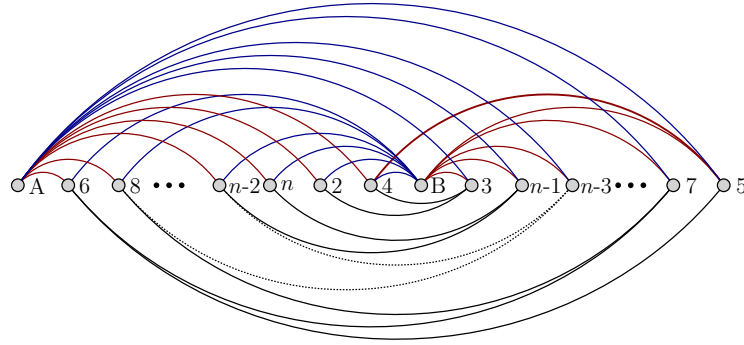
Fig. 6: A 3-stack layout of the skeleton graph in which no two consecutive vertices of the skeleton path lie between A and B.

necessary to continue digging the details of the proof-sketch. Our idea was that if our findings on the second step were more promising, then we could try to modify (e.g., somehow augment) the skeleton graph, and eventually manage to guarantee its property. So, we decided to proceed to check the second step of the proof-sketch, under the assumption that somehow we can guarantee the property of the first step, that is, several pairs of consecutive vertices $\langle y_1, y_2 \rangle, \ldots, \langle y_{2k-1}, y_{2k} \rangle$ of the skeleton path appear between the two designated vertices A and B.

Under this assumption, we had to check whether two stellating vertices a_i and b_i of two faces $\langle A, y_{2i-1}, y_{2i} \rangle$ and $\langle B, y_{2i-1}, y_{2i} \rangle$ of the stellated-skeleton will inevitably appear between y_{2i-1} and y_{2i}, for some $i \in \{1, \ldots, k\}$. To check this, we created an instance of the stellated skeleton with approximately 100 vertices (that is, $k \approx 25$). Then, employed restrictions R.1 and R.3 of our system to constrain the layout to be computed, such that: (i) vertex A is a predecessor of all vertices of the graph, (ii) vertex B is a successor of all vertices of the graph, (iii) for every $i = 1, \ldots, k$, the partial order $\langle y_{2i-1}, a_i, b_i, y_{2i} \rangle$ is forbidden, and (iv) the same holds for the partial order $\langle y_{2i-1}, b_i, a_i, y_{2i} \rangle$. As it was the case with the first step of the proof-sketch, the system could report a solution (algo.inf. uni-tuebingen.de/linearlayouts/linearlayout.html#68), which was again symmetric and we were able to generalize it to all stellated skeletons; see Figure 7.

Our findings on the third step of the proof-sketch. Finding a graph with the properties of the graph Q in [31] was definitely a challenging task. In our efforts, we crafted several internally-triangulated planar graphs with a quadrangular outer face, which contained substructures that are known to require a third stack, such as planar 3-trees [21], the Goldner-Harary graph [20] and planar graphs with several separating triangles [25]. We also tested approximately 4.000 randomly generated planar graphs with 100 to 150 vertices, which we created by first triangulating an evenly distributed point set within a quadrangular region, and then by connecting four vertices at the corners of this region to all vertices on the (internally triangulated) point set without introducing crossings. Each of these graphs had by construction a quadrangular outer face (s, a, t, b) and
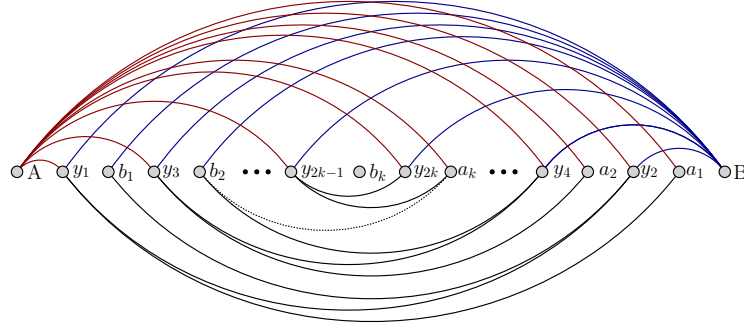
Fig. 7: A 3-stack layout of the stellated-skeleton in which there exist no index $i \in \{1, \ldots, k\}$, such that the vertices y_{2i-1}, a_i, b_i, y_{2i} appear in this order between A and B.

was tested for the property of graph Q in [31] using restriction R.3 and R.6 as follows: (i) vertices s, a and t appear in this order, (ii) vertices s, b and t appear in this order, and (iii) the edges from s and t that end in vertices between s and t are assigned to the first two pages of the layout.

To put it briefly: we did not manage to find a graph with the claimed properties of graph Q in [31]. As a matter of fact, in the layouts that we computed we observed that most of the times very few vertices were eventually between s and t (besides a and b), which is an indication that the restrictions may not be strong enough. It is worth noting at this point that, in the original proof-sketch, it is mentioned that graph Q is the endproduct of a series of constructions of smaller gadget-graphs Q_1, Q_2 and so forth (that are not described; actually, not even their properties are described), which is another indication that if a graph with the properties of graph Q exists, then it might be tremendously large.

4 Conclusions

In this paper, we presented a novel system equipped with several features that automate most of the standard procedures that a domain expert needs for computing different types of (constrained) linear layouts of graphs. As a proof of concept, we used the developed system to check the different parts of a proof-sketch by Yannakakis [31], and we managed to gain valuable insights. Our findings, of course, were mostly negative. The proposed system, however, might be useful in finding a suitable graph with the properties of graph Q in [31] and some other graph (than the one Yannakakis proposed) yielding the 4-tuple describe in Section 3.1. And so we are still hopeful that the proof-sketch can be completed.

More in general, we believe that the developed system is extremely useful for proving lower bounds (e.g., to narrow the current wide gap between the upper and the lower bound on the queue number of planar graphs), which is the main reason to have it available online. We further plan to equip it with additional features, such as adding support for searching and filtering the layouts, highlighting the restrictions and more advanced definitions of restrictions.

References

1. J. M. Alam, M. A. Bekos, M. Gronemann, M. Kaufmann, and S. Pupyrev. On dispersable book embeddings. In A. Brandstädt, E. Köhler, and K. Meer, editors, *WG*, volume 11159 of *LNCS*, pages 1–14. Springer, 2018. `doi:10.1007/978-3-030-00256-5\_1`.

2. J. M. Alam, M. A. Bekos, M. Gronemann, M. Kaufmann, and S. Pupyrev. Queue layouts of planar 3-trees. In T. C. Biedl and A. Kerren, editors, *Graph Drawing and Network Visualization*, volume 11282 of *LNCS*, pages 213–226. Springer, 2018. `doi:10.1007/978-3-030-04414-5\_15`.

3. M. J. Bannister, W. E. Devanny, V. Dujmović, D. Eppstein, and D. R. Wood. Track layouts, layered path decompositions, and leveled planarity. *Algorithmica*, 2018. `doi:10.1007/s00453-018-0487-5`.

4. M. A. Bekos, M. Gronemann, and C. N. Raftopoulou. Two-page book embeddings of 4-planar graphs. *Algorithmica*, 75(1):158–185, 2016. `doi:10.1007/s00453-015-0016-8`.

5. M. A. Bekos, M. Kaufmann, and C. Zielke. The book embedding problem from a SAT-solving perspective. In E. Di Giacomo and A. Lubiw, editors, *Graph Drawing and Network Visualization*, volume 9411 of *LNCS*, pages 125–138. Springer, 2015. `doi:10.1007/978-3-319-27261-0\_11`.

6. F. Bernhart and P. C. Kainen. The book thickness of a graph. *J. Comb. Theory, Ser. B*, 27(3):320–331, 1979. `doi:10.1016/0095-8956(79)90021-2`.

7. T. C. Biedl, T. Bläsius, B. Niedermann, M. Nöllenburg, R. Prutkin, and I. Rutter. Using ILP/SAT to determine pathwidth, visibility representations, and other grid-based graph drawings. In S. K. Wismath and A. Wolff, editors, *Graph Drawing*, volume 8242 of *LNCS*, pages 460–471. Springer, 2013. URL: `https://doi.org/10.1007/978-3-319-03841-4_40`, `doi:10.1007/978-3-319-03841-4\_40`.

8. C. Binucci, E. D. Giacomo, M. I. Hossain, and G. Liotta. 1-page and 2-page drawings with bounded number of crossings per edge. *Eur. J. Comb.*, 68:24–37, 2018. `doi:10.1016/j.ejc.2017.07.009`.

9. J. F. Buss and P. W. Shor. On the pagenumber of planar graphs. In R. A. DeMillo, editor, *ACM Symposium on Theory of Computing*, pages 98–100. ACM, 1984. `doi:10.1145/800057.808670`.

10. M. Chimani and R. Zeranski. Upward planarity testing via SAT. In W. Didimo and M. Patrignani, editors, *Graph Drawing*.

11. G. Cornuéjols, D. Naddef, and W. R. Pulleyblank. Halin graphs and the travelling salesman problem. *Math. Program.*, 26(3):287–294, 1983. `doi:10.1007/BF02591867`.

12. H. de Fraysseix, P. O. de Mendez, and J. Pach. A left-first search algorithm for planar graphs. *Discrete & Computational Geometry*, 13:459–468, 1995. `doi:10.1007/BF02574056`.

13. G. Di Battista, F. Frati, and J. Pach. On the queue number of planar graphs. *SIAM J. Comput.*, 42(6):2243–2285, 2013. `doi:10.1137/130908051`.

14. V. Dujmović and F. Frati. Stack and queue layouts via layered separators. *J. Graph Algorithms Appl.*, 22(1):89–99, 2018. `doi:10.7155/jgaa.00454`.

15. V. Dujmović, P. Morin, G. Joret, P. Micek, D. Wood, and T. Ueckerdt. Planar graphs have bounded queue-number. *CoRR*, 2019. `arXiv:1904.04791`.

16. V. Dujmović and D. R. Wood. On linear layouts of graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2):339–358, 2004.

17. G. Ewald. Hamiltonian circuits in simplicial complexes. *Geometriae Dedicata*, 2(1):115–125, 1973. `doi:10.1007/BF00149287`.

18. G. Gange, P. J. Stuckey, and K. Marriott. Optimal *k*-level planarization and crossing minimization. In U. Brandes and S. Cornelsen, editors, *Graph Drawing*, volume 6502 of *LNCS*, pages 238–249. Springer, 2010. URL: `https://doi.org/10.1007/978-3-642-18469-7_22`, `doi:10.1007/978-3-642-18469-7\_22`.

19. J. L. Ganley. Stack and queue layouts of Halin graphs, 1995. Manuscript.

20. A. Goldner and F. Harary. Note on a smallest nonhamiltonian maximal planar graph. *Bulletin of the Malaysian Math. Sci. Soc.*, 1(6):41–42, 1975.

21. L. S. Heath. Embedding planar graphs in seven pages. In *FOCS*, pages 74–83. IEEE Computer Society, 1984. `doi:10.1109/SFCS.1984.715903`.

22. L. S. Heath, F. T. Leighton, and A. L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Discrete Math.*, 5(3):398–412, 1992. `doi:10.1137/0405031`.

23. L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.*, 21(5):927–958, 1992. `doi:10.1137/0221055`.

24. S. Istrail. An algorithm for embedding planar graphs in six pages. *Iasi University Annals, Mathematics-Computer Science*, 34(4):329–341, 1988.

25. P. C. Kainen and S. Overbay. Extension of a theorem of whitney. *Appl. Math. Lett.*, 20(7):835–837, 2007. `doi:10.1016/j.aml.2006.08.019`.

26. T. Nishizeki and N. Chiba. *Planar Graphs: Theory and Algorithms*, chapter 10. Hamiltonian Cycles, pages 171–184. Dover Books on Mathematics. Courier Dover Publications, 2008.

27. S. Pupyrev. Mixed linear layouts of planar graphs. In F. Frati and K. Ma, editors, *Graph Drawing and Network Visualization*, volume 10692 of *LNCS*, pages 197–209. Springer, 2017. `doi:10.1007/978-3-319-73915-1\_17`.

28. S. Rengarajan and C. E. V. Madhavan. Stack and queue number of 2-trees. In D. Du and M. Li, editors, *COCOON*, volume 959 of *LNCS*, pages 203–212. Springer, 1995. `doi:10.1007/BFb0030834`.

29. U. of Arizona, Dep. of Computer Science. URL: `http://be.cs.arizona.edu`.

30. R. Wiese, M. Eiglsperger, and M. Kaufmann. yFiles: Visualization and automatic layout of graphs. In *Graph Drawing*, pages 453–454, 2001. URL: `https://doi.org/10.1007/3-540-45848-4_42`, `doi:10.1007/3-540-45848-4\_42`.

31. M. Yannakakis. Four pages are necessary and sufficient for planar graphs (extended abstract). In J. Hartmanis, editor, *ACM Symposium on Theory of Computing*, pages 104–108. ACM, 1986. `doi:10.1145/12130.12141`.

32. M. Yannakakis. Embedding planar graphs in four pages. *J. Comput. Syst. Sci.*, 38(1):36–67, 1989. `doi:10.1016/0022-0000(89)90032-9`.