Cyclic Leveling of Directed Graphs

Christian Bachmaier, Franz J. Brandenburg,
Wolfgang Brunner, and Gergö Lovász

University of Passau, Germany
{bachmaier|brandenb|brunner|lovasz}@fim.uni-passau.de

Abstract. The Sugiyama framework is the most commonly used concept for visualizing directed graphs. It draws them in a hierarchical way and operates in four phases: cycle removal, leveling, crossing reduction, and coordinate assignment.
However, there are situations where cycles must be displayed as such, e. g., distinguished cycles in the biosciences and processes that repeat in a daily or weekly turn. This forbids the removal of cycles. In their seminal paper Sugiyama et al. also introduced recurrent hierarchies as a concept to draw graphs with cycles. However, this concept has not received much attention since then.
In this paper we investigate the leveling problem for cyclic graphs. We show that minimizing the sum of the length of all edges is $\mathcal{NP}$-hard for a given number of levels and present three different heuristics for the leveling problem. This sharply contrasts the situation in the hierarchical style of drawing directed graphs, where this problem is solvable in polynomial time.

1 Introduction

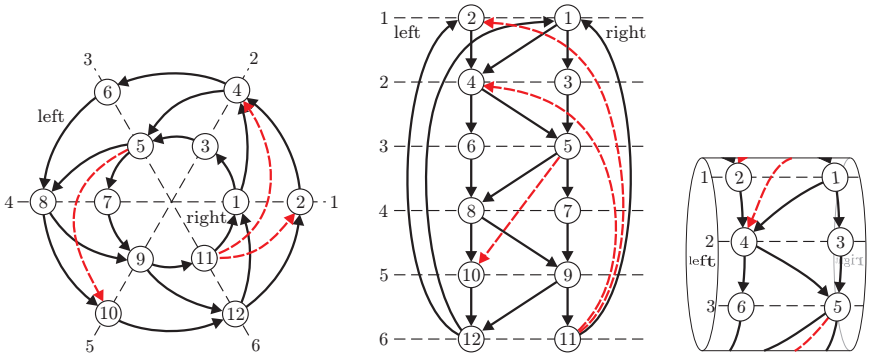
The Sugiyama framework [8] is among the most intensively investigated algorithms in graph drawing. It is the standard technique to draw directed graphs, and displays them in an hierarchical manner. This is well-suited particularly for directed acyclic graphs, which are drawn top-down (or left to right) and level by level. These drawings reflect the underlying graph as a partial order. Typical applications are schedules, UML diagrams and flow charts.

In the general case, the Sugiyama framework first destroys cycles. In the decycling phase it removes or redirects some edges until the resulting graph is acyclic. However, there are many situations, where this procedure is inacceptable. For example, there are well-known cycles in the biosciences, and it is a common standard there to display these cycles as such. These cycles often serve as a landmark [7]. Another application for cycles are repeating processes, such as daily, weekly or monthly schedules with almost the same tasks. Here again it is important that these cycles are clearly visible in a "nice" drawing.

In their original paper from 1981 [8], Sugiyama et al. have proposed a solution for both the hierarchical and the cyclic style. The latter is called *recurrent hierarchy*. A recurrent hierarchy is a level graph with additional edges from the last

to the first level. Here, two drawings are natural: The first is a 2D drawing, where the levels are rays from a common center, and are sorted counterclockwise by their number, see Fig. 1(a). All nodes of one level are placed at different positions on their ray and an edge $e = (u, v)$ is drawn as a monotone counterclockwise curve from u to v wrapping around the center at most once. The second is a 3D drawing on a cylinder, see Fig. 1(c). A combination of the two drawing methods would be the best of both worlds: An interactive 2D view which shows horizontal levels. This view can be scrolled upwards and downwards infinitely and always shows a different part of the cylinder, e. g., the front view of Fig. 1(c).

Recurrent hierarchies are known to most graph drawers – but unnoticed. A planar recurrent hierarchy is shown on the cover of the book by Kaufmann and Wagner [6]. There it is stated that recurrent hierarchies are "unfortunately [...] still not well studied". The reason is that they are much harder. Intuitively, there is no start and no end, there are no top and bottom levels. Formally, we pinpoint a problem which is tractable in the hierarchical style and is intractable in the cyclic style.



(a) Cyclic (b) Hierarchic (c) Cyclic 3D

Fig. 1. Example drawings

In cyclic drawings edges are irreversible and cycles are represented in a direct way. Thus, the cycle removal phase disappears from the common Sugiyama framework. This saves much effort, since the underlying problem is the $\mathcal{NP}$-hard feedback arc set problem [5]. Another advantage are short edges. The sum of the edge length can be smaller than in the hierarchical case: Consider a cycle consisting of three nodes. The only way to draw this graph in the Sugiyama framework is to reverse one edge which will then span two levels. Therefore, the sum of the edge length will be four. In the cyclic case this graph can be drawn on three levels s. t. each edge has span one. Moreover, the cyclic style reduces the number of crossings in general. See Fig. 1(a) and (b) as an example. At the

threshold with no crossings [2], there are cyclic level planar graphs which are not level planar. Here, consider Fig. 1 with the solid edges only.

Note that any Sugiyama drawing is a cyclic Sugiyama drawing which discards the option to draw edges between the last and first level. Therefore, all benefits of such drawings exist in the cyclic case as well. However, the sum of the edge length and the number of crossings will often be smaller.

In this paper we consider the leveling phase for the cyclic Sugiyama framework. In the hierarchical version this phase is generally solved by topological sorting, or more advanced, by the Coffman-Graham algorithm [3, 6]. As our main result we show that minimizing the sum of the length of all edges is $\mathcal{NP}$-hard for a given number of levels. This sharply contrasts the hierarchical case. Then we introduce three heuristics for the cyclic leveling problem, and evaluate them experimentally within the Gravisto system [1].

2 Preliminaries

Let $G = (V, E)$ be a directed graph. For a given $k \in \mathbb{N}$ we call $\phi\colon V \to \{1, 2, \ldots, k\}$ a level assignment and $G = (V, E, \phi)$ a cyclic k-level graph. We denote with $\deg(v)$ the degree of a node $v \in V$. For two nodes $u, v \in V$ let $\operatorname{span}(u, v) = \phi(v) - \phi(u)$ if $\phi(u) < \phi(v)$, and $\operatorname{span}(u, v) = \phi(v) - \phi(u) + k$ otherwise. For an edge $e = (u, v) \in E$ we define $\operatorname{span}(e) = \operatorname{span}(u, v)$ and $\operatorname{span}(G) = \sum_{e \in E} \operatorname{span}(e)$. For a set of edges $E' \subseteq E$ we define $\operatorname{span}(E') = \sum_{e \in E'} \operatorname{span}(e)$. $\operatorname{next}(l) = (l \mod k) + 1$ denotes the level after l. For a node $v \in V$ and a subset $V' \subset V$ we set $E(v, V') = \{\, (u, v) \in E \mid u \in V' \,\} \cup \{\, (v, w) \in E \mid w \in V' \,\}$.

3 Complexity of Cyclic Leveling

In this section we consider different leveling problems and compare their complexity in the hierarchical and the cyclic style. The graphs $G = (V, E)$ are directed in both cases and are acyclic in the hierarchical case.

Definition 1 (Height and Width). *Let G be a directed graph which is drawn s. t. the edges connect vertices on different levels and are uni-directed from the start level to a successive level. Let the* height *be the number of levels and let the* width *be the maximal number of nodes on a level.*

We can now state our leveling problems, both for the common hierarchical style and for the cyclic style of recurrent hierarchies.

Problem 1. Let $k \in \mathbb{N}$. Does there exist a leveling of G with height at most k?

Problem 2. Let $\omega \in \mathbb{N}$. Does there exist a leveling of G with width at most ω?

Problem 3. Let $k, \omega \in \mathbb{N}$. Does there exist a leveling of G with height at most k and width at most ω?

In the hierarchical case problems 1 and 2 are easy: The former can be solved in linear time by the longest path search algorithm [6], whereas, the latter is trivial as each graph has a leveling with width 1 by placing each node on its own level according to a topological sorting of G. Problem 3 is $\mathcal{NP}$-hard as this corresponds to precedence constrained scheduling [5].

In the cyclic case all these problems are easy. Note that an edge $e = (u, v)$ does not impose any constraint on the leveling of the nodes u and v. u can have a smaller level, a larger level, and even the same level as v. Therefore, the answer to Problems 1 and 2 is yes (if $k, \omega > 0$). For Problem 3 there is a cyclic leveling if $|V| \leq k \cdot \omega$ by arbitrarily placing vertices in a $k \times \omega$ grid.

Problem 4. Let $l \in \mathbb{N}$. Does there exist a leveling of G with $\text{span}(G) \leq l$?

In the hierarchical case minimizing the span can be formulated as an ILP:

$$\min \sum_{(u,v) \in E} (\phi(v) - \phi(u)) \tag{1}$$

$$\forall v \in V : \phi(v) \in \mathbb{N} \tag{2}$$

$$\forall e = (u, v) \in E : \phi(v) - \phi(u) \geq 1 \tag{3}$$

This ILP can be solved in polynomial time, since the constraint matrix is totally unimodular [6]. Therefore, Problem 4 has a polynomial time complexity in the hierarchical case as well. In the cyclic case the span can no longer be formulated by a system of linear equations, as a case differentiation or the modulo operation is needed.

As a degenerated case we may place all nodes on a single level. Then all edges have span 1 which is obviously minimal. Therefore, we sharpen Problem 4:

Problem 5. Let $l, k \in \mathbb{N}$. Does there exist a leveling of G with exactly k levels with $\text{span}(G) \leq l$?

Problem 5 is simple for $k = 1$, as such a leveling exists if $l \geq |E|$. For $k > 1$ we now show that the problem is $\mathcal{NP}$-hard. We use two different reductions for $k = 2$ and $k > 2$. For $k = 2$ we use the $\mathcal{NP}$-hard bipartite subgraph problem [5]:

Problem 6 (Bipartite subgraph). Let $G = (V, E)$ be an undirected graph and $k \in \mathbb{N}$. Does there exist a bipartite subgraph G' of G with at least k edges?

Lemma 1. *Let $G = (V, E)$ be an undirected graph and $l \in \mathbb{N}$. Let $G^* = (V, E^*)$ be a directed version of G with an arbitrary direction for each edge. G contains a bipartite subgraph G' with at least l edges if and only if there exists a leveling of G^* on two levels with* $\text{span}(G^*) \leq 2|E| - l$.

Proof. "⇒": Let $G' = (V', E')$ be a bipartite subgraph of G with at least l edges. Let $V_1 \dot{\cup} V_2 = V'$ be the partition of the node set with all edges of E' between V_1 and V_2. We construct the following leveling for G^*: For each node $v \in V_1$ we set $\phi(v) = 1$, for each node $v \in V_2$ we set $\phi(v) = 2$, and for all nodes $v \in V \setminus (V_1 \cup V_2)$

we set $\phi(v) = 1$. Then each edge in E' has span 1 and all other edges have span 1 or 2. Thus, $\text{span}(G^*) \leq |E'| + 2(|E| - |E'|) = 2|E| - |E'| \leq 2|E| - l$.

"$\Leftarrow$": Let ϕ be a leveling of G^* with $\text{span}(G^*) \leq 2|E| - l$. Let V_1 and V_2 be the nodes of V on level 1 and 2, respectively. Let $E' \subseteq E$ be the set of edges e s. t. one end node is in V_1 and the other is in V_2. Then, $G' = (V, E')$ is bipartite. All edges in E' have span 1 in the leveling ϕ and all other edges have span 2. As $\text{span}(G^*) \leq 2|E| - l = 1 \cdot l + 2(|E| - l)$, there are at least l edges with span 1. As these edges are in E', G' is a bipartite subgraph of G with at least l edges. □

For $k > 2$ we use graph k-colorability, which is $\mathcal{NP}$-hard for a fixed $k > 2$ [5]:

Problem 7 (Graph k-colorability). Let $G = (V, E)$ be an undirected graph and let $k \in \mathbb{N}$. Does there exist a coloring $c : V \to \{1, \ldots, k\}$, s. t. $c(u) \neq c(v)$ for every edge $e = \{u, v\} \in E$?

Lemma 2. *Let $G = (V, E)$ be an undirected graph and let $k \in \mathbb{N}$. Let $G' = (V, E')$ with E' containing the edges (u, v) and (v, u) for each edge $\{u, v\} \in E$. G is k-colorable if and only if G' has a leveling on k levels with* $\text{span}(G') \leq k \cdot |E|$.

Proof. Let $e = \{u, v\} \in E$. Note that for each leveling ϕ of G' and each edge $e = (u, v) \in E'$ the sum of the spans of (u, v) and (v, u) is either k (if $\phi(u) \neq \phi(v)$) or $2k$ (if $\phi(u) = \phi(v)$). Thus, $\text{span}(G') \geq k \cdot \frac{|E'|}{2} = k \cdot |E|$.

"$\Rightarrow$": Let c be a coloring of G. Set $\phi = c$. Then, for each edge with end nodes u and v in G (and G') $\phi(u) \neq \phi(v)$ holds. Thus, each pair of edges (u, v) and (v, u) in sum has span k and $\text{span}(G') = k \cdot |E|$ holds.

"$\Leftarrow$": Let ϕ be a leveling of G' with $\text{span}(G') \leq k \cdot |E|$. Then, $\text{span}(G') = k \cdot |E|$ and for each edge $(u, v) \in E'$ $\phi(u) \neq \phi(v)$ holds. Thus, $c = \phi$ is a correct coloring. □

Theorem 1. *Let $G = (V, E)$ be a directed graph and $l, k \in \mathbb{N}$ ($k \geq 2$). The problem whether there exists a leveling of G on k levels with* $\text{span}(G) \leq l$ *is $\mathcal{NP}$-complete.*

Proof. Lemma 1 and Lemma 2 show that the problem is $\mathcal{NP}$-hard for $k = 2$ and $k > 2$, respectively. The problem is obviously in $\mathcal{NP}$. □

4 Heuristics

As minimizing the span of a graph in a cyclic leveling with k levels is $\mathcal{NP}$-complete, we have to use heuristics. Known approaches from the hierarchical case as the longest path method [6] or the Coffman-Graham algorithm [3] cannot be easily adapted to the cyclic case. They heavily rely on the fact that the graph is acyclic and start the leveling process at nodes with no incoming edges. As it is not guaranteed that such nodes exist in the cyclic case at all, we introduce three new heuristics. They are evaluated experimentally in Sect. 5.

The input to the algorithms are the number of levels k and the maximum number of nodes on a level ω. The output is the leveling $\phi : V \to \{1, \ldots, k\}$. The parameter k is either given by the user or it is pre-computed, e. g., as the average length of simple cycles detected by a depth first search of the graph.

Table 1. Complexity of leveling (k as height and ω as width)

	hierarchical	cyclic
Minimizing k	$\mathcal{O}(\|V\| + \|E\|)$, by longest path	Set $\phi : V \to \{1\}$
Minimizing ω	$\mathcal{O}(\|V\| + \|E\|)$, by ϕ = topsort	Choose injective ϕ
Leveling with k and ω given	$\mathcal{NP}$-hard, precedence constrained scheduling	Test $k \cdot \omega \geq \|V\|$
Minimizing k with ω given	$\mathcal{NP}$-hard for arbitrary $\omega > 2$	Set $k = \lceil \frac{\|V\|}{\omega} \rceil$
Minimizing ω with k given	$\mathcal{NP}$-hard for $k > 2$	Set $\omega = \lceil \frac{\|V\|}{k} \rceil$
Minimizing span(G) with k given	$\mathcal{P}$, by LP	$\mathcal{NP}$-hard for $k > 1$

4.1 Breadth First Search

The breadth first search (BFS) heuristic (Algorithm 1) is rather simple: We choose an arbitrary start node v, set $\phi(v) = 1$ and perform a directed BFS from v. When we reach a node w for the first time using an edge (u, w), we set $\phi(w) = \text{next}(\phi(u))$ if this level does not contain ω nodes already. Otherwise, we move w to the first non-full level.

Using this heuristic the tree edges will have a rather short span. But the back edges are not taken into account for the leveling at all. Thus, these edges can be arbitrarily long.

Lemma 3. *The BFS leveling heuristic needs $\mathcal{O}(|V| + |E| + k^2)$ time.*

Proof. BFS runs in $\mathcal{O}(|V| + |E|)$ time. In addition we must keep and update an array N of size k. $N[i]$ denotes the first non-full level from level i. At most all k levels can get full which costs $\mathcal{O}(k)$ time for each. □

4.2 Minimum Spanning Tree

This heuristic has similarities to the algorithm of Prim [4], which computes the minimum spanning tree (MST) of a graph. We sequentially level the nodes by a greedy algorithm. Let $V' \subset V$ be the set of already leveled nodes. When we level a node v, all edges in $E(v, V')$ get a fixed span. Therefore, we set $\phi(v)$ s. t. span($E(v, V')$) is minimized. Note that there are possibly more edges incident to v which are also incident to not yet leveled nodes. These edges will be considered when the second end node is leveled.

We decide in which order to add the nodes by using a distance function $\delta(v)$. We discuss four options:

Algorithm 1: breadthFirstSearchLeveling

Input: G: a directed graph, k: the number of levels,
ω: the maximum number of nodes on each level
Output: ϕ: a cyclic leveling of G

```
1  Queue Q ← ∅
2  Leveling φ ← ∅
3  foreach u ∈ V do u.marked ← false
4  foreach l ∈ {1, ..., k} do N[l] ← l
5  foreach u ∈ V do
6      if ¬u.marked then
7          Q.append(u)
8          u.marked ← true
9          φ(u) ← N[1]
10         updateN(N[1])
11         while ¬Q.isEmpty() do
12             v ← Q.removeFirst()
13             foreach neighbor w of v do
14                 if ¬w.marked then
15                     w.marked ← true
16                     φ(w) ← N[next(φ(v))]
17                     updateN(φ(w))
18                     Q.append(w)
19 return φ
```

Minimum Increase in Span (MST_MIN) We choose the node which will create the minimum increase in span in the already leveled graph:

$$\delta_{\text{MIN}}(v) = \min_{\phi(v)\in\{1,\ldots,k\}} \text{span}(E(v, V')) \tag{4}$$

Minimum Average Increase in Span (MST_MIN_AVG) Using the distance function δ_{MIN} will place nodes with a low degree first, as nodes with a higher degree will almost always cause a higher increase in span. Therefore, considering the increase in span per edge is reasonable:

$$\delta_{\text{MIN\_AVG}}(v) = \min_{\phi(v)\in\{1,\ldots,k\}} \frac{\text{span}(E(v, V'))}{|E(v, V')|} \tag{5}$$

We distribute isolated nodes evenly on the non-full levels in the end.

Maximum (Average) Increase in Span (MST_MAX(_AVG)) Choose the node which causes the maximum (average) increase in span per edge:

$$\delta_{\text{MAX}}(v) = \frac{1}{\delta_{\text{MIN}}(v)}, \qquad \delta_{\text{MAX\_AVG}}(v) = \frac{1}{\delta_{\text{MIN\_AVG}}(v)} \tag{6}$$

The idea behind this is the following: A node which causes a high increase in span will cause this increase when leveled later as well. But if we level this node now, we can possibly level other adjacent, not yet leveled nodes in a better way.

Note that we only use the distance function $\delta(v)$ to determine which node to level next. When we level a node v, we set $\phi(v)$ s. t. the increase in span will be minimized. In some cases several levels for v will create the same increase in span. We will then choose a level for v which minimizes $\sum_{e \in E(v,V')} \text{span}(e)^2$ as well. Thus, we assign v a level which is more centered between its leveled adjacent nodes. In each case we can only use a level which has not yet ω nodes on it. Nodes with already leveled neighbors block a place on their optimal level s. t. they can later be placed on the level. Algorithm 2 shows the complete heuristic.

Algorithm 2: minimumSpanningTreeLeveling

Input: G: a directed graph, k: the number of levels,
ω: the maximum number of nodes on each level
Output: ϕ: a cyclic leveling of G

```
1  Heap H ← ∅
2  Leveling φ ← ∅
3  foreach u ∈ V do
4  |  u.status ← white
5  |  δ(u) ← ∞
6  foreach u ∈ V do
7  |  if u.status = white then
8  |  |  δ(u) ← 0
9  |  |  H.insert(u)
10 |  |  while ¬H.isempty() do
11 |  |  |  v ← H.removeMin()
12 |  |  |  v.status ← black
13 |  |  |  φ(v) ← getOptimalLevel(v)
14 |  |  |  foreach neighbor w of v with w.status ≠ black do
15 |  |  |  |  δ(w) ← computeDistance(w)
16 |  |  |  |  φ(w) ← getOptimalLevel(w)
17 |  |  |  |  if w.status = gray then
18 |  |  |  |  |  H.update(w)
19 |  |  |  |  else
20 |  |  |  |  |  w.status ← gray
21 |  |  |  |  |  H.insert(w)
22 return φ
```

Lemma 4. *The MST heuristic needs* $\mathcal{O}(|V| \log |V| + k \cdot \deg(G) \cdot |E|)$ *time.*

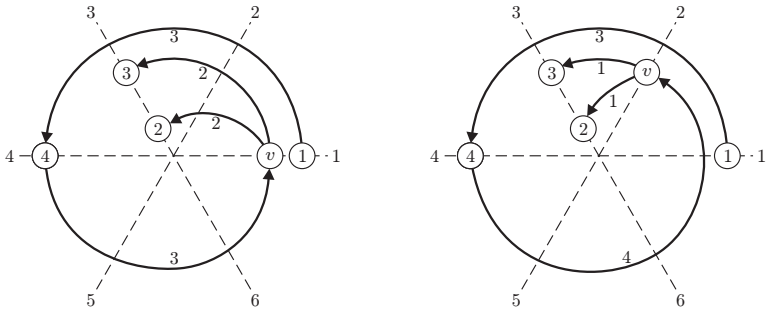
Proof. The time complexity is dominated by the while loop. Here, removing each node from the heap costs $\mathcal{O}(|V| \log |V|)$. Each edge $e = (w, z) \in E$ may change

its span whenever a neighbor v of w (or z) is fixed on a level. In this case each of the k levels is tested for w (or z). Thus, we get $\mathcal{O}(k \cdot (\deg(w) + \deg(z)))$ for e and $\mathcal{O}(k \cdot \deg(G) \cdot |E|)$ for all edges. Finally, updating all neighbors in the heap costs $\mathcal{O}(|E| \log |V|)$ (or $\mathcal{O}(|E|)$ using a Fibonacci heap). □

4.3 Force Based

Spring embedders use a physical model to simulate the edges as springs [6]. Forces between nodes are computed and the nodes are moved accordingly. Transferring this idea to the cyclic leveling problem, we could use a force function similar to conventional energy based placement algorithms as follows:

$$\text{force}(v) = \sum_{(v,w)\in E} (\text{span}(v,w) - 1)^2 - \sum_{(u,v)\in E} (\text{span}(u,v) - 1)^2 \tag{7}$$



(a) v in energy minimum of (7) (b) v in energy minimum of (8)

Fig. 2. Force based placement of node v

However, moving a node to its energy minimum using this force will not minimize the span of the graph, i. e., (7) minimizes the deviation between the edge lengths, e. g., see Fig. 2. Furthermore, the span may increase when moving a node towards its energy minimum, as some edges can flip from span 1 to span k. We solve this problem by using directly the span as the (undirected) force which is minimized:

$$\text{force}(v) = \text{span}(E(v, V)) \tag{8}$$

We move the node with the maximum impacting force. And we directly move the node to its energy minimum, which is the level s. t. the span is minimized. For this, we test all possible (non-full) levels. Note that moving all nodes at once would not decrease time complexity here. Algorithm 3 shows the pseudo code.

As an initial leveling we either use a random leveling (SE_RND) or the result of the best minimum spanning tree heuristic MST_MIN_AVG (SE_MST).

Algorithm 3: forceBasedLeveling

Input: G: a directed graph, k: the number of levels,
ω: the maximum number of nodes on each level
Output: ϕ: a cyclic leveling of G

```
1  Heap H ← ∅
2  φ ← computeInitialLeveling()
3  foreach v ∈ V do
4      computeForce(v)
5  while improvement ∧ iterations < limit do
6      foreach v ∈ V do
7          H.insert(v)
8      while ¬H.isEmpty() do
9          v ← H.removeMax()
10         φ(v) ← energyMinimalLevel(v)
11         foreach neighbor w of v do
12             updateForce(w, v)
13 return φ
```

Lemma 5. *In the force based heuristic $\mathcal{O}(|V| \log |V| + k \cdot |E|)$ time is needed for each iteration.*

Proof. Inserting all nodes in the heap can be implemented in $\mathcal{O}(|V|)$ time. Removing each node from the heap has time complexity $\mathcal{O}(|V| \log |V|)$. Computing the energy minimal level for v costs $\mathcal{O}(k \cdot \deg(v))$, which is $\mathcal{O}(k \cdot |E|)$ for all nodes. Computing the new force is possible in time $\mathcal{O}(1)$ for each neighbor of v, in $\mathcal{O}(\deg(v))$ for all neighbors and $\mathcal{O}(|E|)$ in total. The $\mathcal{O}(|E|)$ updates in the heap cost $\mathcal{O}(|E| \log |V|)$ (or $\mathcal{O}(|E|)$ using a Fibonacci heap). □

5 Empirical Results

In this section we evaluate and compare the heuristics with each other and with an optimal leveling. The optimal leveling is computed by a branch and bound algorithm which can be used for graphs up to 18 nodes.

In Fig. 3 the running times of the algorithms are shown. Figure 4 compares the calculated spans of the heuristics with the optimal span. For a better pairwise comparison of the heuristics, Fig. 5 only shows their results.

For Fig. 3 and 5 the number of nodes $|V|$ was increased by steps of 50 each time. For each size 10 graphs with $|E| = 5|V|$ were created randomly. For Fig. 4 10 graphs for each size $|V|$ and $|E| = 2|V|$ were used. In all three diagrams k and ω were set to $\sqrt{2|V|}$, s. t. there were $2|V|$ possible node positions. Each heuristic was applied to each graph $\min(|V|, 30)$ times using different start nodes resp. initial levelings and choosing the average.

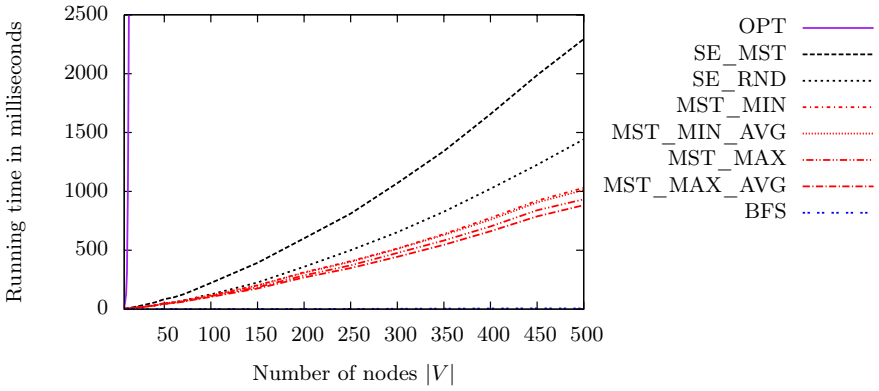


Fig. 3. Running times of the algorithms

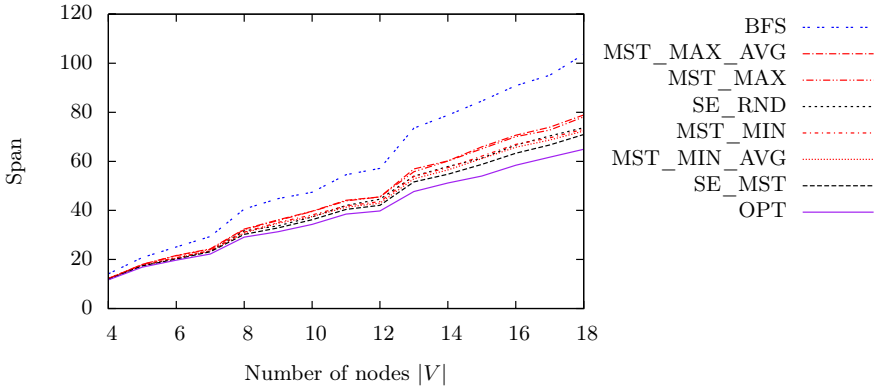


Fig. 4. Average spans of small graphs

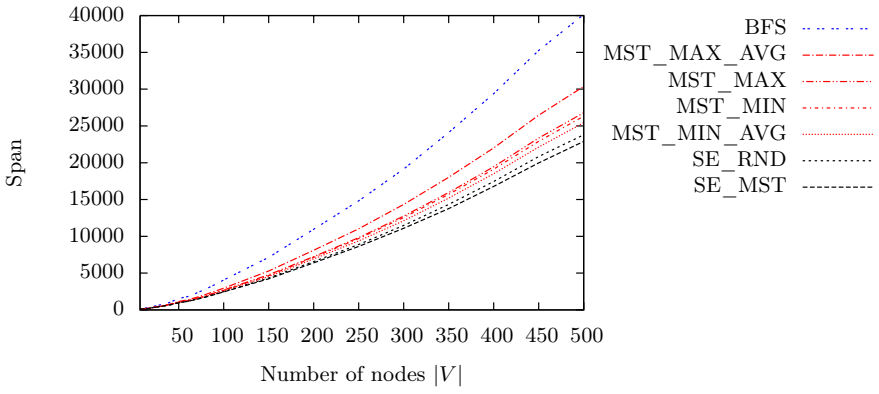


Fig. 5. Average spans of large graphs

The benchmarks show the practical performance of the algorithms. All tests were run on a 2.8 GHz Celeron PC under the Java 6.0 platform from Sun Microsystems, Inc. within the Gravisto framework [1].

As expected the force based heuristics with MST initialization computes the best leveling and the results are close to the optimum. All MST variants do not differ very much, but MST_MIN_AVG seems to be the best. The results can be improved by applying the heuristics i times to the same graph with i different start nodes or different initial levelings, respectively, and choosing the best result. However, the price is an i times higher running time.

6 Summary and Open Problems

The leveling problem has turned out to be essentially different in the hierarchical and cyclic style. We have shown different optimization goals for the cyclic leveling compared to the goals of the hierarchic leveling. For the reasonable minimization of the sum of the edge lengths we have shown the $\mathcal{NP}$-hardness and presented three practical heuristics for the problem.

Open problems are the approximation ratios of our heuristics, other quality measures for cyclic drawings, the best number of levels, and the completion of the cyclic style to a cyclic Sugiyama framework.

References

1. Bachmaier, C., Brandenburg, F.J., Forster, M., Holleis, P., Raitner, M.: Gravisto: Graph visualization toolkit. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 502–503. Springer, Heidelberg (2005)
2. Bachmaier, C., Brunner, W.: Linear time planarity testing and embedding of strongly connected cyclic level graphs. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 136–147. Springer, Heidelberg (2008)
3. Coffman, E.G., Graham, R.L.: Optimal scheduling for two processor systems. Acta Informatica 1(3), 200–213 (1972)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge, 2nd edn. (2001)
5. Garey, M.R., Johnson, D.S.: A Guide to the Theory of NP-Completeness. W. H. Freemann, New York (1979)
6. Kaufmann, M., Wagner, D.: Drawing Graphs, LNCS, vol. 2025. Springer, Heidelberg (2001)
7. Michal, G. (ed.): Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology. Wiley, New York (1999)
8. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics 11(2), 109–125 (1981)