Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 17, no. 3, pp. 329–362 (2013)
DOI: 10.7155/jgaa.00296

Multi-level Verticality Optimization: Concept, Strategies, and Drawing Scheme

Markus Chimani[1] *Philipp Hungerländer*[2]

[1]Theoretical Computer Science, Uni Osnabrück, Germany
[2]Discrete Mathematics and Optimization, Uni Klagenfurt, Austria

Abstract

In traditional multi-level graph drawing—known as Sugiyama's framework—the number of crossings is considered one of the most important goals. Herein, we propose the alternative concept of optimizing the *verticality* of the drawn edges. We formally specify the problem, discuss its relative merits, and show that drawings that are good w.r.t. verticality in fact also have a low number of crossings. We present heuristic and exact approaches to tackle the verticality problem and study them in practice.

Furthermore, we present a new drawing scheme (inherently bundling edges and drawing them monotonously), especially suitable for verticality optimization. It works without the traditional subdivision of edges, i.e., edges may span multiple levels, and therefore potentially allows to tackle larger graphs.

Submitted:	Reviewed:	Revised:	Reviewed:	Revised:
October 2011	April 2012	October 2012	July 2013	July 2013
	Accepted:	Final:	Published:	
	July 2013	July 2013	July 2013	

Article type: Regular paper — Communicated by: M. Kaufmann

Markus Chimani was funded by a Carl-Zeiss-Foundation juniorprofessorship while conducting this research.

E-mail addresses: markus.chimani@uni-osnabrueck.de (Markus Chimani) philipp.hungerlaender@aau.at (Philipp Hungerländer)

1 Introduction

One of the most common drawing paradigms for hierarchical graphs, known as Sugiyama's framework [31], is based on the following idea: First, we place the nodes of a graph on different *levels*, effectively fixing their y-coordinates. Edges spanning multiple levels are subdivided into chains of (sub)edges such that each (sub)edge only spans one level, resulting in a *proper level graph.* The second step is to fix an ordering of the nodes on their levels such that a certain optimization goal (usually the number of crossings) is minimized. As a third step, the nodes are assigned x-coordinates, consistent with the ordering, such that, e.g., the number of bends is minimized or the edges' *verticality* is maximized. (Sub)edges are thereby always drawn as straight lines.

In this paper we discuss a somehow inverse approach to the problem of finding a good node order on the levels, focusing on third step's optimization goal. We observe that when thinking about a drawing where the edges are drawn mostly vertical, we will usually also have a low number of crossings. Furthermore, edges tend to cross only on a very local scale (i.e., edges will usually not cross over a large horizontal distance), increasing the drawing's readability [29]. Hence, perhaps the combination of maximum verticality and low crossing number leads to (qualitatively) better drawings than the traditional minimum crossing number in conjunction with high verticality.

The assumption that high verticality leads to few crossings and good drawings is also supported by the following observation: The *barycenter heuristic* is one of the earliest, and still probably the most common heuristic to quickly solve the layered crossing minimization problem in practice, especially for large-scale graphs. Yet, the heuristic does actually *not* actively try to minimize crossings, but iteratively decides on positions p of nodes on layer i, such that p lies at (or close to) the barycenter of the positions of its adjacent nodes on level $i-1$. So, the heuristic is in fact mainly trying to optimize our verticality problem! Its crossing minimization properties arise only in the wake of this optimization goal.

As we will briefly discuss below, our problem is a special form of an ordering problem, which also arises in areas unrelated to graph drawing. As such, we call the problem *Multi-level Vertical Ordering (MLVO).* When specifically talking about the graph drawing application, i.e., finding orderings of the nodes on their levels such that the edges are drawn *as vertical as possible* (see a precise definition below) we use the term *Multi-level Verticality Optimization*, which, nicely enough, gives rise to the same abbreviation.

As we will see, MLVO is a natural quadratic ordering problem (QOP). We will show that MLVO is NP-hard and closely related to the traditional problem of multi-level crossing minimization (MLCM), where one seeks node orders such that the number of crossings in multi-level drawings is minimized. MLCM has received a lot of attention not only within the graph drawing community, but in combinatorial optimization in general; see, e.g., [2, 6] for overviews on strong heuristics and exact algorithms to tackle the problem. MLVO is also related to the problem of multi-level planarization (MLP) [27, 13], i.e., find node orders which minimize the number of edges that have to be removed in order to obtain a planar (sub)drawing. This problem has been proposed as a possible substitute for MLCM, suggesting that it can result in more aesthetically pleasing drawings.

1.1 Focus and Contribution

The focus of this paper is to present the concept and specification of MLVO in its native graph drawing setting, discuss its relative merits and challenges, show its solvability via various algorithmic strategies, and give an overview on possible further extensions. Therefore, it is beyond the scope of this paper to give in-depth details of inner working of the rather complex exact algorithms to tackle the problem. For such a discussion see [5], where we consider ILP-, QP-, and SDP-based algorithms to tackle the base problem of MLVO. Although graph drawing is the main (and most developed) application area, MLVO can also be interesting in other, very diverse, application fields like scheduling and multiple ranking. Herein, we focus on the graph drawing issue, and refer to [5] for short descriptions of the latter.

In the next section, we will formally discuss the concept of verticality, its application to proper level graphs, and propose the MLVO problem as an alternative to the steps 2&3 in the traditional Sugiyama framework. We will also show that MLVO is already NP-hard for the problem restricted to two levels.

Thereafter (Section 3), we propose a different new drawing style based on the verticality concept; it does not require to subdivide the edges after layering the graph in Sugiyama's first step. It seems that this is one of the first approaches allowing the direct use of non-proper level graphs within the Sugiyama framework, without any (not even implicit) subdivision of long edges. — In [11], an approach to bound the number of nodes per subdivided edge has been presented, and in [2] explicit subdivisions are avoided by modeling the chain of (implicit) subdivision nodes via a single vertical block.

In Section 4 we show how to adopt commonly known MLCM paradigms in order to obtain simple heuristics to solve MLVO in practice and for large graphs. We conclude this section with presenting a sophisticated exact approach based on semi-definite programming (SDP), which dominates any exact approaches based on integer linear programs (ILPs) or quadratic programming (QP)—we refer the reader to the more mathematically oriented companion paper [5] for details on this approach[1]. Herein, we are mainly interested in the SDP's application to the graph drawing scenario. Additionally, the SDP approach can also be used as an exact quadratic compactor for Sugiyama's third stage, i.e., after minimizing crossings.

In Section 5 we present experiments based on all these algorithms and compare the approach to the traditional MLCM concept. We conclude this paper with discussing several extensions to MLVO that can be interesting in practice.

[1]We note that the content of the companion paper is as disjoint from this paper as possible: it virtually only deals with SDP, QP, and ILP approaches from a polyhedral point of view (for the core MLVO problem, without GD specific extensions) and gives a short overview on further application areas (in scheduling and ranking) besides graph drawing.

2 Verticality and Proper Drawings

In the remainder of this paper, we will always consider the following input: Let $G = (V, E)$ with $V = \dot{\bigcup}_{i=1}^{p} V_i$ be a *level graph*, where we draw the nodes V_i on the i-th level. The function $\ell : V \to \{1, \dots, p\}$ gives the level on which a node resides. The edges are directed, i.e., $(v, u) \in E$ induces $\ell(v) < \ell(u)$; v and u are the *source* and *target* node, respectively. Furthermore, let $G' = (V', E')$ with $V' = \dot{\bigcup}_{i=1}^{p} V'_i$, $E' = \dot{\bigcup}_{i=1}^{p-1} E'_i$, and $E'_i \subseteq V'_i \times V'_{i+1}$ for all $1 \leq i < p$, be the corresponding *proper level graph*. Thereby, the original edges E are subdivided into segments such that each edge in E' connects nodes of adjacent levels. Clearly, we have $V_i \subseteq V'_i$ for all levels i. The additional nodes created by this operation are called *long-edge dummy nodes*, or *LEDs* for short.

MLCM and MLP are usually applied to *proper* level graphs, as only the introduction of LEDs (or similar constructions, as in the aforementioned [2]) allows to concisely describe their feasible solutions and objective values. Optimizing these problems means solving $p-1$ dependent, sequentially linked bilevel QOPs (one for each pair of adjacent levels). We will see that MLVO cannot only be applied in such a setting (resulting in *proper drawings*), but also directly to non-proper graphs (resulting in *non-proper drawings*): this gives rise to "true" multi-level QOPs as all levels can directly interact with each other.

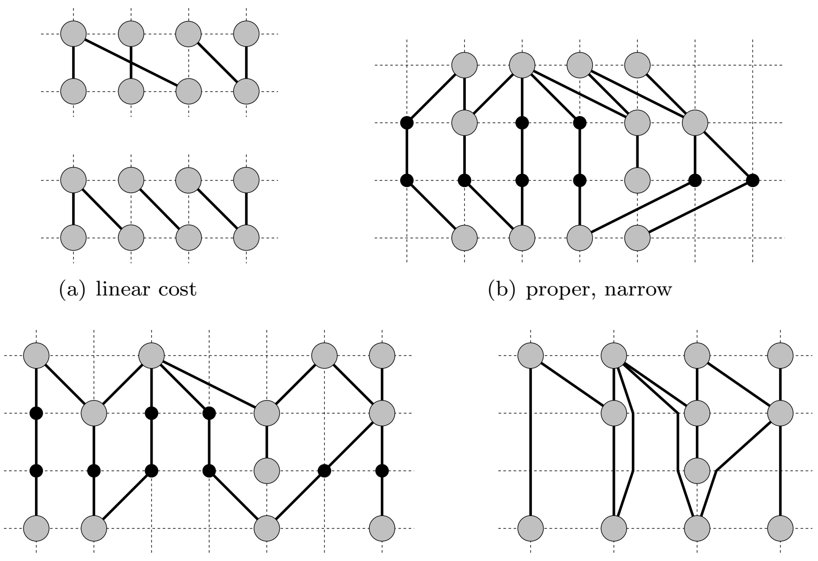
2.1 Verticality

We define the colloquial term *verticality* via its inverse, *non-verticality*: The non-verticality $\mathfrak{d}(e)$ of a straight-line edge e is the square of the difference in the horizontal coordinates of its end nodes. Then, $\mathfrak{d}(E) := \sum_{e \in E} \mathfrak{d}(e)$ denotes the overall non-verticality of a solution. Using only this notion, we could arbitrarily optimize a drawing by scaling the horizontal coordinates. Hence we consider *grid drawings*, i.e., the nodes' positions are mapped to integral coordinates, thereby relating verticality to the drawing's width. Clearly, we only consider *adjacent* integrals for the y-coordinates.

It remains to argue why non-verticality has to be a quadratic term: assume we would only consider a linear function, then even a small example as the one depicted in Fig. 1(a) would result in multiple solutions that are equivalent w.r.t. their objective values, even though the bottom one is clearly preferable from the readability point of view. Intuitively, we prefer multiple slightly non-vertical edges, over few but very non-vertical edges. In fact, this argument brings our model in line with the argument of observing crossings only on a local scale.

2.2 Complexity of Verticality Optimization

Consider the decision variant of MLVO, i.e., given some value M we ask whether there exist node orderings such that the obtained non-verticality is at most M.


(a) linear cost (b) proper, narrow

(c) proper, wide (d) non-prop., wide

Figure 1: Example drawings regarding verticality maximization: (a) equivalent quality with respect to a linear objective function, (b)–(d) different drawing paradigms, cf. text. Original nodes are drawn as large gray circles, LEDs as black small circles, PDs (on the empty grid points) are omitted for readability.

Theorem 1 *The decision variant of MLVO is NP-complete, already when considering only two levels.*

Proof: We reduce from the NP-complete PARTITION problem, i.e., given a set of n numbers $a_1, \ldots, a_n \in \mathbb{N}$ with $\sum_{1 \leq i \leq n} a_i = 2B$, does there exist a partition of these numbers such that the sum in both subsets is exactly B? For details on the complexity of the PARTITION problem see Subsection 3.1.5 of Garey and Johnson [15] or the original paper of Karp [21].

Consider the following MLVO instance with two levels, arising from some PARTITION instance. For each a_i, $1 \leq i \leq n$, we introduce a_i many nodes $U_i = \{v_{i,1}, \ldots, v_{i,a_i}\}$ to V_1 and analogously a_i many nodes $U'_i = \{v'_{i,1}, \ldots, v'_{i,a_i}\}$ to V_2. Then we connect every node of U_i with every node U'_i. Finally, we add two additional nodes $t \in V_1$ and $t' \in V_2$, and connect all nodes of V_2 with t. We now have $2B + 1$ nodes on each level.

An MLVO solution is surely optimal if it achieves the following two properties:

1. The node t is on position $B + 1$, independent of the ordering of the nodes in V_2: since the width of both levels is equal and odd and t is adjacent to all nodes in V_2 any other position would result in strictly larger non-verticalities.

2. Consider the nodes U_i, U'_i corresponding to some number a_i. Their optimal arrangement is to tightly pack all nodes of U_i (U'_i, respectively) horizontally, and U_i being vertically exactly below U'_i. Any other arrangement would result in strictly larger non-verticalities.

Now, we define M as the non-verticality caused by a solution fulfilling both properties.[2] If a solution with non-verticality M is achievable, then the node t partitions the numbers of the PARTITION instance into two groups with equal sum B: for any a_i, all its nodes are either left or right of t (as they are tightly packed), and there are exactly B nodes to the left and to the right of t. Vice versa, if the PARTITION instance is satisfiable, then an MLVO solution with non-verticality M exists. □

Interestingly, and in contrast to MLCM, the variant where the order on one layer is fixed, turns out to be polynomially solvable:

Theorem 2 *2-level MLVO, where the node order on one level is fixed, can be solved in $O(|E| + n^3)$ running time where n is the number of nodes on the free level.*

Proof: Let $V_{free} = \{1, 2, \ldots, n\}$ denote the nodes on the free level. We may attribute the non-verticality of any edge in the graph to its node in V_{free}. Clearly, the non-verticalities attributed to any node $v \in V_{free}$ depend only on its own position within its level and not on the position of any other node in V_{free}. Thus we can independently compute the sum of non-verticalities attributed to a node i when located at position j for all $1 \leq i, j \leq n$ and store the values as the i-j-entries of an $n \times n$ matrix N. Computing this matrix takes $O(|E| + n^2)$ time.

Now, finding an optimal ordering for this variant of MLVO is equivalent to solving the Linear Assignment Problem on matrix N. This can be done in polynomial time, e.g., in $O(n^3)$ with the Hungarian method [9, 23, 26, 32]. □

Notice that both complexity theorems hold true if we compute the non-verticality simply as the sum of the *linear* horizontal distances of all edges.

2.3 Proper Drawing Scheme

We can consider two distinct alignment schemes, due to the fact that the node partitions V'_i have different cardinalities. Let $\omega' := \max_{1 \leq i \leq p} |V'_i|$ denote the width of the widest level. In the *narrow* alignment schemes, we require the nodes on the levels to lie on directly adjacent x-coordinates (Fig. 1(b)). Then, we would usually like to center the distinct levels w.r.t. each other, i.e, a level i may only use the x-coordinates $\{\delta'_i, \ldots, \delta'_i + |V'_i| - 1\}$, with the level's *width offset* $\delta'_i := \lfloor (\omega' - |V'_i|)/2 \rfloor$. The benefit of this alignment scheme is that a simple linear order of the nodes per layer already fully describes the solution. Yet, note that

[2]We can use the quadratic degree and complete-bipartite constraints to state M explicitly.

in most cases such an alignment scheme will not result in aesthetically pleasing drawings.

In the *wide* alignment scheme (Fig. 1(c)), nodes are not restricted to lie on horizontally neighboring grid coordinates. In order to model this, we only need to expand the graph by adding *positional dummy nodes* (PDs) to each level such that all levels have ω' many nodes. All PDs have degree 0. In the following, we will simply consider any (proper) level graph G (G', respectively), which may or may not be augmented with PDs. In graph drawing practice, we will usually only use this wide alignment scheme.

Monotonous drawings. Considering drawings optimal w.r.t. MLVO, we may want to force an additional *monotonicity* property. Within the Sugiyama framework, each edge is drawn using only strongly monotonously increasing y-coordinates. inverting the edge direction). It would be nice to have a similar property along the x-axis over all edge segments corresponding to an original edge. We say a drawing is *monotonous*, if all original edges are weakly monotonous along the x-axis. More formally, let $e = (u, v) \in E$ be any edge in the (non-proper) level graph G, $e_1 = (u = u_0, u_1), e_2 = (u_1, u_2), ..., e_k = (u_{k-1}, u_k = v)$ the corresponding chain of edges in G', and $x : V' \to \mathbb{N}$ the mapping of nodes to x-coordinates in the final drawing. Then a drawing is monotonous, if $x(u) \leq (\geq)\ x(v)$ implies $x(u_i) \leq (\geq)\ x(u_{i+1})$ for all $0 \leq i < k$.

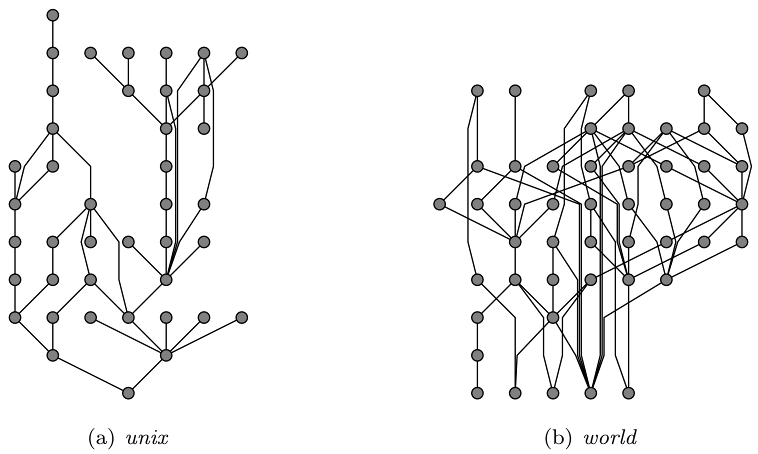
Even though this might be counterintuitive at first, an optimal MLVO solution does not induce this property. We may, however, explicitly ask for this property to hold, giving rise to the *monotonous MLVO* problem.

3 Non-Proper Drawing Scheme

When looking at typical Sugiyama-style drawings, we often observe that LEDs—even though they are never explicitly drawn—are given too much space: Objectively, it is unreasonable for LEDs to require as much horizontal space as a real node. This was also considered, e.g., in [11]. Therefore, current drawing algorithms try hard to "bundle" multiple long edges into one dense channel (whose width is constant, disregarding the number of its elements), to improve overall readability of large, dense graphs; see, e.g., [28]. Yet, such methods usually still use LEDs.

Herein, we show that a drawing scheme can be devised which makes LEDs completely unnecessary, cf. Figs. 1(d) and 2. We will, however, retain PDs as described above to allow a wide alignment scheme on our grid. A particularly interesting side effect of working without LEDs is that the considered graph stays smaller. Thus, this method allows more involved, time-consuming methods (as, e.g., our exact SDP-approach) to be applicable to larger original graphs.

Consider a non-proper level graph G for which we have computed a solution to (non-proper) MLVO, i.e., we have an ordering of the nodes on their layers and non-verticality of an edge is measured simply as the square of the horizontal coordinate difference of its end nodes. We now describe how to generate a


(a) *unix* (b) *world*

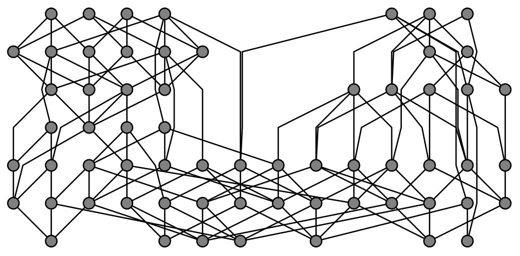

(c) *sm96*

Figure 2: Examples of the non-proper drawing scheme with (near-)optimal verticality, cf. Sect. 5, all with prespecified layering.

drawing realizing such an order and verticality. A working python code of this algorithm is available at `http://www.cs.uos.de/theoinf`.

3.1 Hypothetical and shifted routing

The y- and x-coordinates of the nodes are fixed by the layering (ℓ) and node order per layer, respectively. As a general idea—called the *hypothetical routing*—we want to draw each edge vertically up to the level directly below the target node. Only there, the edge bends to be drawn as a line with the computed non-verticality.

Clearly, there are problems with this simple concept: Firstly, routing edges strictly vertical may require to draw them through other nodes—we may say an edge *overlaps* a node. Secondly, vertical segments of multiple edges would partially coincide—we may say the two edges overlap each other. If two edges have exactly one point (other than a common end point) in common, we say the edges *cross*. Observe that we hence also consider it a crossing if the target node of an edge coincides with an interior point of another edge. By construction, the number of common points between any two edges can only be 0, 1, or infinite.

In order to achieve a readable drawing, we have to avoid any edge-node or edge-edge overlaps. Therefore, we have to relax the hypothetical routing such that we route an edge $e = (u, v)$ vertically "close to" the x-coordinate of the source node (i.e., shifted by some small $s(e)$). More formally, the edge starts at the coordinate $(x(u), \ell(u))$, has a first bend point at $(x(u) \pm s(e), \ell(u) + 1)$ shifting the edge either to the left or to the right (depending on the edge's overall direction), and is routed vertically until the point $(x(u) \pm s(e), \ell(v) - 1)$ where it bends to go straight to the end point $(x(v), \ell(v))$. For short edges or $s(e) = 0$, some bend points may vanish in the obvious way. When $s(e)$ is assumed small enough, the overall non-verticality of this routing is roughly equivalent to the non-verticality achieved by the hypothetical routing. In particular, as the maximum shifting value tends to 0, the drawing's overall non-verticality converges to that of the hypothetical routing for the whole graph.

Observe that *vertical* edges (i.e., $x(u) = x(v)$) are somehow special as it is per se not clear, whether $s(e)$ should be added or subtracted; we will discuss this uncertainty later. Our overall goal is to obtain a *shifted* routing with the following properties:

(P1) there are no edge-node and no edge-edge overlaps,

and there are only two types for crossings:

(P2) two edges cross exactly once if their hypothetical routings cross,

(P3) a vertical edge e_1 *may* cross (at most once) another edge e_2, if e_2's source node is an interior point of e_1's hypothetical routing, but e_2's target node is not.

Observe that this induces that adjacent edges do not cross. In order to achieve these properties, the shift values $s(e)$ for the edges have to be chosen carefully.

Monotonous drawings. Clearly, the hypothetical drawing is monotonous in the x- and y-coordinates, as well as strictly monotonous (i.e., in their general direction). Due to our shifting, this property (necessarily) gets slightly violated for edges drawn strictly vertical in the hypothetical routing. We may say a drawing is *β-monotonous* for some β, if edges with identical source and target x-coordinates deviate from this coordinate by at most β at any point, and all other edges are drawn monotonously.

3.2 Computing Shifts

Let $V^- \subseteq V$ denote the original nodes (in contrast to possible PDs). Larger y (x) coordinates are higher (more right, respectively) in the drawing. We will first compute shift *labels* σ for all edges in the graph. Afterwards, these labels will be transformed into actual shift values (see below): for now it is sufficient to think of a formula of the type $s(e) = \varepsilon \cdot \sigma(e)$ for some small $\varepsilon > 0$.

Computing Shift Labels. See also Algorithm 1. The key idea is to sort all edges with a common source node from the inside (no shift) to the outside (large shifts) such that adjacent edges will not cross. By considering these source nodes from top to bottom, we ensure that all edges starting above the currently considered node already have computed shift labels: the new shift labels can take them into account in order to not cross through these other edges (unless required by the hypothetical routing). Also, we can observe that each edge, at the time of its labeling, is crossing free except possibly for its last segment (from the level directly beneath the target node to the target node itself).

We iteratively consider all nodes $v \in V^-$, in decreasing order of their y-coordinate. For all edges $e \in E_v := \{(v,u) \in E : \ell(u) > \ell(v)\}$ that have v as their source node, we will compute an integer label $\sigma(e)$.

To these ends, we further subpartition E_v into $E_v^<, E_v^=, E_v^>$ depending on whether the target node u_e of an edge is left ($x(u_e) < x(v)$), directly above ($x(u_e) = x(v)$), or right ($x(u_e) > x(v)$) of v, respectively. For all nodes $w \in V$ we store the smallest free label $\sigma^l(w), \sigma^r(w)$ to its left and right side, respectively. Initially, these labels are 1 for $w \in V^-$ and 0 otherwise. Now, let $V_e := \{w \in V : x(w) = x(v) \wedge \ell(v) < \ell(w) < \ell(u_e)\}$ be the set of nodes vertically above v, but below the edge's target node. Then, $\sigma^l_{\max}(e) := \max_{v \in V_e} \sigma^l(v)$ denotes the smallest possible label for e. If $V_e = \emptyset$, we set the value to 0 as we do not require any shift for e. Define $\sigma^r_{\max}(e)$ analogously.

First, consider the *vertical* edges $E_v^=$. Let $E_v^{=,l}, E_v^{=,r}$ be any partition of $E_v^=$—see below for a discussion of a proper choice—into edges that should be shifted to the left or to the right, respectively, if necessary. Now, sort $E_v^{=,l}$ ($E_v^{=,r}$ is analogous) by increasing layer of the target nodes, and iteratively (using the sorted order of the edges) assign integral labels. To label an edge e, set $\sigma(e) := \sigma^l_{max}(e)$ and afterwards $\sigma^l(w) := \sigma(e) + 1$ for all $w \in V_e$.

Now consider the set $E_v^<$ ($E_v^>$ is analogous) and sort it by decreasing $\ell(u_e)$, where u_e is the edge's target node; edges within the same equivalence class w.r.t. this measure are sorted by increasing $|x(u_e) - x(v)|$. We can draw all edges that span only one level as straight lines and remove them from $E_v^<$ for the following discussion. Iterating over the edges e in sorted $E_v^<$, we again set $\sigma(e) := \sigma^l_{max}(e)$ and afterwards $\sigma^l(w) := \sigma(e) + 1$ for all $w \in V_e$. Observe that subsequent edges in sorted $E_v^<$ are always labeled 1 larger than their direct predecessor. In this scheme it may happen that the first edges of sorted $E_v^<$ and $E_v^>$ are labeled 0. If this is the case, we increase all labels of the set where the first edge has the lower target node (breaking ties arbitrarily) by 1, to avoid co-linear lines due to not shifting two edges.

From Shift Labels to Shift Values. The simple notion of computing shift values via $\varepsilon \cdot \sigma(e)$ would require the nodes to be drawn with a width smaller than 2ε. To avoid this, we can simply and consistently offset all shift values, if there is an original node to be passed by the edge. Let $0 < \alpha < \beta < 0.5$ be prespecified parameters describing the distance of minimum and maximum shift, hence, allowing node sizes of width $< 2\alpha$. Let σ^* be the largest overall

Algorithm 1 Computing shift labels

```
Require: level graph $G = (V = \dot{\bigcup}_{i=1}^{p} V_i, E)$, original nodes (not PDs) $V^- \subseteq V$.
Ensure: shifting labels $\sigma$ for the edges
 1: function LabelEdge(edge $e = (v, u)$, side)                    $\triangleright$ side $\in \{l, r\}$
 2:     $V_e := \{w \in V : x(w) = x(v) \wedge \ell(v) < \ell(w) < \ell(u_e)\}$
            $\triangleright$ those are the nodes that $e$ would go through in the hypothetical routing
 3:     $\sigma(e) := \max\{0, \max_{w \in V_e} \sigma^{side}(w)\}$
            $\triangleright$ choose smallest possible shift distance far enough from any of $V_e$
 4:     for all $w \in V_e$ do $\sigma^{side}(w) := \sigma(e) + 1$
            $\triangleright$ set next free shift distance for all nodes where $e$ passes by
 5: procedure ComputeShiftLabels
 6:     for all $v \in V^-$ do
 7:         $\sigma^l(v) := 1$, $\sigma^r(v) := 1$          $\triangleright$ next free shift distance left/right of $v$
 8:     for all $v \in V \setminus V^-$ do
 9:         $\sigma^l(v) := 0$, $\sigma^r(v) := 0$                    $\triangleright$ we may pass through PDs
10:     for all $v \in V^-$ in decreasing order of $\ell(v)$ do
11:         Let $E_v$ be the edges with source $v$, and $(E_v^<, E_v^=, E_v^>)$ their partition
            in whether their target node is left, directly above, or right of $v$,
            respectively.
12:         Let $(E_v^{=,l}, E_v^{=,r})$ be any partition of $E_v^=$ (cf. text).
13:         for all $e = (v, u) \in E_v^{=,l}$ in increasing order of $\ell(u)$ do
14:             LabelEdge$(e, l)$
15:         for all $e = (v, u) \in E_v^{=,r}$ in increasing order of $\ell(u)$ do
16:             LabelEdge$(e, r)$
17:         Sort $E_v^<$: edge $(v, u_1)$ is before $(v, u_2)$ iff $\ell(u_1) > \ell(u_2)$ or $\ell(u_1) =$
            $\ell(u_2) \wedge x(u_1) > x(u_2)$. Remove edges $(v, u)$ with $\ell(v) = \ell(u) - 1$ from
            the list.
18:         for all $e = (v, u) \in E_v^<$ (in sorted order) do
19:             LabelEdge$(e, l)$
20:         Sort $E_v^>$: edge $(v, u_1)$ is before $(v, u_2)$ iff $\ell(u_1) > \ell(u_2)$ or $\ell(u_1) =$
            $\ell(u_2) \wedge x(u_1) < x(u_2)$. Remove edges $(v, u)$ with $\ell(v) = \ell(u) - 1$ from
            the list.
21:         for all $e = (v, u) \in E_v^>$ (in sorted order) do
22:             LabelEdge$(e, r)$
23:         Let $e_l = (v, u_l) \in E_v^<, e_r = (v, u_r) \in E_v^>$ be the first edges in sorted
        $E_v^<, E_v^>$
24:         if $\sigma(e_l) = \sigma(e_r) = 0$ then $\triangleright$ $E_v^= = \emptyset$ and the hypothetical routings
                                        of $e_l, e_r$ do not pass through an original
                                        node
25:             if $\ell(u_l) < \ell(u_r)$ then
26:                 Increase $\sigma(e)$ by 1, for all $e \in E_v^<$
27:             else
28:                 Increase $\sigma(e)$ by 1, for all $e \in E_v^>$
```

label, then $\delta := (\beta - \alpha)/\max\{1, \sigma^* - 1\}$ denotes the shift difference between two neighboring edges. For any node $v \in V^-$ we compute the actual shift value for each emanating edge: If there is an edge e_0 in E_v with label 0, we set $s(e_0) = 0$. If e_0 exists and $e_0 \notin E_v^=$, let $\alpha' := 0$; otherwise $\alpha' := \alpha$. For any other edge $e \in E_v$, we then set $s(e) := (\sigma(e) - 1) \cdot \delta + \alpha'$.

3.3 Analysis of the drawing algorithm

By the fact that the maximum shift distance is bounded by β, and non-vertical edges are only shifted into the direction of its target node, we have:

Observation 1 *The drawing obtained by the above non-proper drawing algorithm is β-monotonous.*

Considering the order and strategy in which the labels are chosen, it is not too hard to see:

Lemma 1 *The drawing obtained by the above non-proper drawing algorithm satisfies the crossing properties (P1)–(P3).*

Proof: Assume two edges would overlap, conflicting (P1). Since the maximum shift value $\beta < 0.5$, both edges would have a common source node. Yet, by construction, any two adjacent edges have different shift values (and/or directions) in the end. Also, by construction, any original node w is initialized with $\sigma^l(w) = \sigma^r(w) = 1$, and hence any edge with a hypothetical routing overlapping w will have a label ≥ 1 and hence not overlap w in the shifted drawing.

Assume there would be a crossing between two edges even though they are not allowed to cross due to (P2),(P3). By construction the drawing is β-monotonous with $\beta < 0.5$ and there can be no crossings between two vertical edges violating (P3). Assume e_a is a non-vertical edge crossing some other edge e_b even though their hypothetical routings did not cross. By definition of shifted routings, the existence of such a crossing induces that both source nodes have the same x-coordinate. If both edges are adjacent on their source node v, they would have gotten distinct labels and by the property of sorting the edges E_v they would not cross. If e_b is vertical, its source node has to be above e_a's source node. But then, it was labeled earlier than e_a, and the smallest feasible label for e_a would have been chosen larger than e_b's label. Analogously, if both edges are non-vertical and non-adjacent, one of them, say e_b starts above e_a's source node. Again, they cannot cross, because e_a's label was chosen to be larger then any other label along its path, in particular, also larger than e_b's label. Observe that the latter argument also shows that non-adjacent edges cross at most once.
□

Hence, the pure orderings of the nodes per layer already induce the required number of crossings, up to crossings due to vertical edges. These are decided by the respective *l,r-partitions* of the vertical edges, i.e., the partitions of $E_v^=$ into $E_v^{=,l}, E_v^{=,r}$, for all $v \in V^-$.

Lemma 2 *Fixing all l,r-partitions, the above drawing algorithm requires the minimum possible number of crossings. Yet, even when given the node orders per level, obtaining l,r-partitions that lead to the overall minimum number of crossings is NP-hard.*

Proof: The first part follows from the algorithmic description and the fact that properties (P1)–(P3) are satisfied (Lemma 1). The NP-hardness follows from the fact that already a single column of vertically arranged nodes constitutes the NP-hard *fixed linear crossing number* problem [24]. □

In practice, the partition problem is usually not critical: the number of crossings between pairs of vertical edges is usually dominated by the crossings involving non-vertical edges. In fact, in our implementation we settle on a very simple, yet seemingly sufficient, heuristic: during the algorithm, we greedily pick the side where the edge attains the smaller label; we break ties by classifying edges whose source node v is on the left (right) half of the drawing as $E_v^{=,l}$ ($E_v^{=,r}$, resp.). This tie breaking can be motivated as follows: Considering a node v on the left side of the drawing, it will usually have more adjacent nodes to its right than to its left side. Our decision will hence usually lead to fewer crossings.

Based on the fact that all sorting is done on integral values, we can conclude:

Theorem 3 *The above drawing algorithm generates a non-proper drawing of a level graph $G = (V, E)$ with specified node orders per level in $O(|V| + |E|)$ time. The edges' routings are monotonous in their y-coordinates, β-monotonous in their x-coordinate, and realize the minimal number of crossings (w.r.t. the given node orders and l,r-partitions).*

Proof: The properties w.r.t. monotonicity and crossings follow from the above Observation 1 and Lemmata 1 and 2. It remains to discuss the running time. Every edge is labeled exactly once, and is contained in precisely one set E_v. The partitioning of a set E_v can be done in linear time. Regarding sorting the partitioned subsets, we observe that both the labels and the x-coordinates are integer values bounded by $O(|V|)$, and hence linear time sorting algorithms are applicable. The lexicographic sorting (of $E_v^<, E_v^>$) can be achieved using radix sort, using any stable linear time sorting algorithm as the inner sorting algorithm. □

4 Solving MLVO

4.1 Barycenter and Median

We already noted in the introduction that traditional MLCM heuristics in fact often optimize the drawings' verticality (in a narrow alignment scheme setting). In particular, we can use the traditional approach of computing the barycenter or median for the nodes, by only looking at fixed positions of the nodes one level below/above (in case of proper drawings), or on any level below/above (in case of non-proper drawings), and sort them accordingly, cf. [22]. Iterating this procedure for all layers both in the upward and downward direction (i.e., alternatingly consider the levels below or above) until no more improvement is possible minimizes the number of crossings only indirectly, but the edges' verticality directly.

When considering the wide alignment scheme, we observe that we cannot compute reasonable values for PDs as they have no incident edges. Therefore, we first only compute the barycenter/median for the original nodes—we call these values the *desired locations* of the nodes—and sort them accordingly. Recall that these positions are (in general) fractional values. Then, we try to disperse the PDs (necessary to achieve the layer width ω') into this list such that the desired locations numerically coincide with their final positions in the list (including the PDs) as good as possible. In other words, the idea is to fill the list with PDs such that large differences between the desired locations are filled up with PDs, whereas there are no PDs between original nodes with similar desired location. In the end, the PDs should be places such that the average location distance between neighboring nodes in the sorted list is as close to 1 as possible, given that the fractional location of original nodes is fixed. We do so heuristically by iteratively putting the PDs between two adjacent desired location values d_1, d_2 with largest gaps, and set the PD's location value to $\min\{(d_1 + d_2)/2, d_1 + 1\}$. Consider this order of the nodes (i.e., sorted by increasing (fractional) location value). We choose integer x-coordinates of a node according to its position in this sorted list.

Observe that, alternatively, Sugiyama's originally proposed idea (see [31, 22]) to find x-coordinates (also based again on the barycenter heuristic) could also be adopted. Preliminary tests showed no significant differences between both methods, with slight benefits for our above described method.

4.2 Local Optima via 2-Opt and Sifting

Consider an initial node order per level, either by random assignment or by applying the above barycenter or median heuristic. We can apply local optimization strategies to further improve the solution. Herein, we describe two implementation-wise simple, yet promising approaches, which are known from various different optimization problems, including MLCM.

The first approach is a 2-Opt strategy, i.e., we iteratively pick all possible pairs of nodes v_1, v_2 on a common layer, where at least one node is not a PD. We then exchange their positions and reevaluate the overall verticality. Clearly, we are only interested in the change of the solution value, and therefore it suffices to compute $\Delta\mathfrak{d} := \mathfrak{d}_{\text{before}}(E_1) + \mathfrak{d}_{\text{before}}(E_2) - \mathfrak{d}_{\text{after}}(E_1) - \mathfrak{d}_{\text{after}}(E_2)$, where E_1, E_2 are the edges having v_1, v_2 as one of their end points, respectively. We finally apply this modification only if $\Delta\mathfrak{d}$ is positive. The process stops when no more improving node pair can be found.

Similarly, we can devise a sifting strategy (cf. [3] for the context of crossing minimization). We pick any two nodes v_1, v_2 (both may be PDs) on a common layer. Let V_{v_1,v_2} be the nodes between these, w.r.t. the current node order on this level. We then shift all nodes $\{v_1\} \cup V_{v_1,v_2}$ by one position towards the old position of v_2, and move v_2 to the former position of v_1. To decide whether this is an improvement, we have to evaluate the non-verticalities of the edges incident to $\{v_1, v_2\} \cup V_{v_1,v_2}$. Again, we only perform improving steps and the process stops when no more such step is possible.

4.3 SDP

Herein, we only very briefly outline an exact approach for MLVO based on semidefinite programming (SDP). We refer to [5] for the companion paper that deals almost exclusively with ILP, QP, and SDP aspects of the problem. (However, see Appendix 7 for a brief overview.)

At the core of our approach, we use variables $y_{uv} \in \{-1, 1\}$, for any pair of nodes on a common layer, to specify a linear order of the nodes per layer; the variable is 1 if u lies left of v, and -1 otherwise. It turns out that SDP approaches are particularly able to solve ordering problems (via the help of several constraint classes which would be beyond the scope of this paper) and allow the direct inclusion of our objective function: Coarsely speaking, we can measure the verticality of an edge (u, v) by counting the nodes which, according to the linear order, lie left of u and v, respectively, and take the square of the countings' difference.

In our experiments, we will alternatingly compute the relaxation of this SDP (i.e., we replace the variable domains by reals $-1 \leq y_{uv} \leq 1$) to obtain a lower bound; based on this solution, we apply a hyperplane rounding strategy to obtain feasible solutions, i.e., upper bounds. We iterate this process until the gaps coincide (after rounding the lower bound to the next integer above), or an iteration limit is reached.

MLVO after MLCM (MLVAC). Our MLVO SDP cannot only be used directly after the Sugiyama's first stage, but we can also apply it after a second stage crossing minimization, i.e., after solving an MLCM problem. By fixing the order of the original nodes (non-PDs), the SDP becomes an exact quadratic compactor for Sugiyama's third stage. Such a fixing can be achieved either by dropping the fixed variables altogether and corresponding modifications to the constraint matrix, or by introducing equality constraints on the respective variables. In our experiments, we used the latter approach due to code simplicity. Implementing the reduction strategy would assumingly lead to further improved running times.

Let $p : V \to \{0, 1, \ldots, \omega\}$ be the relative position function, where $p(u) = 0$ means that the relative position of node u is not fixed. We ask for the following constraint to hold for two nodes $u \dot{<} v \in V_m$ with $p(u) > 0, p(v) > 0$

$$y_{uv} = 1, \text{ if } p(u) < p(v), \qquad y_{uv} = -1, \text{ if } p(u) > p(v). \tag{1}$$

We can further strengthen the semidefinite relaxation by adding $\zeta - 1$ linear-quadratic constraints that we get from multiplying (1) with an arbitrary ordering variable $y_{st}, s \dot{<} t \in V_n$.

We also have to adapt the SDP heuristic. We fix the ordering of the "real" nodes and LEDs before hyperplane rounding and then only allow to flip signs of variables involving PDs.

5 Computational Experience

We compare the relative benefits of the different drawing schemes and solution methods discussed above and showcase their visual results. Therefore we apply the exact SDP approach and the heuristics proposed in the previous section to solve MLVO on a variety of test sets. The aim is to investigate their general applicability and behavior, on a wider range of instances. We refrain from an in-depth merit evaluation between MLCM and MLVO, as this would be beyond the scope of this paper. All computations were conducted on an Intel Xeon E5160 (Dual-Core) with 24 GB RAM, running Debian 5.0 in 32bit mode. The SDP algorithm runs on top of MatLab 7.7, whereas the heuristics are implemented in C++. The SDP approach leaves some room for further incremental improvement as we restrict the number of iterations to control the overall computational effort. For the heuristic, we give the *total* running time and best found solution, considering 500 independent runs. We observe that while (for larger graphs) this is beneficial to fewer runs, there are nearly no more improvements in the solution quality when further increasing this number. All graphs considered in this section (including their optimal solutions, where available), as well as an implementation of the non-planar drawing style, are online at `http://www.cs.uos.de/theoinf`.

5.1 Polytopes and instances from the literature

First we consider input graphs from three different sources, which are often considered in related experimental investigations, e.g., [20, 18, 13, 6]. Table 1 gives the instances' central properties: *Polyt.* are graphs modeling the incidence relation between faces (corner, edge, 2D-face,...) of a polytope. *Gr.viz.* are the largest graphs in the Graphviz gallery [17], a set of diverse real-world graphs from different applications. *Other* collects further graphs, as the *worldcup* instances [1] (historic results up until 1986 and 2002), and the social networks MS88 [25] and SM96 [30] (used in multiple prior MLCM publications).

We conducted the MLVO experiments for proper and non-proper graphs, always using the wide alignment scheme. Table 2 gives an overview of our results. We observe that the SDP relaxation is tight enough to give surprisingly small gaps for all the instances. For the heuristics we observe that the pure median and barycenter heuristic behave very similar but only give weak results. The local search routines are still fast and offer vastly superior solutions. Interestingly, due to the multiple runs performed for each instance, it turns out that it

			Proper				Non-proper			
	Instance	p	$\lvert V'\rvert$	$\lvert E'\rvert$	ω'	ζ	$\lvert V\rvert$	$\lvert E\rvert$	ω	ζ
Polyt.	Octahedron	3	26	48	12	199	*always proper*			
	Dodecahedron	3	62	120	30	1306				
	Cube4	4	80	208	32	1985				
Gr.viz	unix	11	59	66	11	606	41	48	7	232
	world	9	116	137	20	1711	48	69	9	325
	profile	9	92	116	28	3403	61	85	14	820
Other	MS88	3	37	80	15	316	*already proper*			
	worldcup86	4	35	55	19	685	25	45	11	221
	worldcup02	4	50	65	23	1013	31	46	14	365
	SM96-full	7	108	179	26	2276	63	134	14	638

Table 1: Instance properties (wide alignment scheme). Cube4 corresponds to a 4-dimensional cube. p gives the number of levels, $\omega^{(\prime)}$ denotes the width of the levels, ζ gives the dimension of the SDP cost matrix.

is usually most beneficial to start with a random initial order, than one based on the median or barycenter heuristic; this avoids to repetitively find the same weak local optima. Generally, 2-opt gives slightly weaker results than sifting or *both* (i.e., using both methods alternatingly). The latter two variants are virtually indistinguishable w.r.t. solution quality, but *both* (which starts with 2-opt) is usually faster. Comparing the upper bounds obtained by the SDP (based on hyperplane rounding of the SDP relaxation) to our special-purpose local-search heuristics, there is no clear winner. We conclude that simply using both—as they run fast anyhow—may be the best alternative.

Dropping the LEDs and solving the non-proper MLVO based on SDPs of smaller dimension, allows us to go well beyond the graph sizes to which exact approaches to MLCM (which cannot be directly applied to the non-proper setting) are restricted. The general behavior of the heuristic approaches is similar (although faster, of course) to the observations noted above.

5.2 Graphs with varying densities

Motivated by these results, we now consider a synthetic benchmark set where we have control over the density parameter: We generated a set of instances having $p \in \{2, 3, 6, 11, 20\}$ layers and $n \in \{6, \ldots, 22\}$ vertices on each layer. For each combination of p and n, we consider random instances with densities $d \in \{0.1, 0.3, 0.5, 0, 7, 0.9\}$, where each potential edge has equal probability of being selected. Hence we have $\lfloor dn^2 \rfloor$ edges between each pair of layers for the non-proper graphs and $\lfloor dn^2 \rfloor$ edges between each pair of adjacent layers for the proper graphs. For each triple (p, n, d), we report the average over 10 generated instances. We compare the SDP bounds obtained by the SDP relaxation and rounding heuristic and the upper bounds achieved by the local search heuristic *both* with random initial order and 500 independent runs.

		SDP		Heuristic		
	Instance	$\mathfrak{d}^*$	time	$\mathfrak{d}_{50}$	$\mathfrak{d}_{500}$	time$_{500}$
		Proper				
Polyt.	Octahedron	$239^{+}5$	0:03:28	244	244	0.70
	Dodecahedron	$1815^{+}81$	29:55:48	1837	1834	10.78
	Cube4	$5279^{+}121$	80:49:47	5364	5360	33.74
Gr.viz	unix	$58^{+}5$	1:19:41	74	69	1.73
	world	$331^{+}95$	54:30:21	486	479	14.84
	profile	$876^{+}169$	95:45:58	962	959	21.57
Other	MS88	$155^{+}2$	0:52:17	157	157	2.62
	Worldcup86	$349^{+}26$	1:44:46	368	356	2.60
	Worldcup02	$385^{+}15$	7:19:38	405	399	6.44
	SM96-full	$658^{+}36$	137:21:07	809	758	39.78
		Non-proper				
Gr.viz	unix	$30^{+}3$	0:10:11	34	33	0.28
	world	$103^{+}7$	0:43:50	114	109	0.62
	profile	$254^{+}5$	3:11:43	266	260	1.87
Other	Worldcup86	$113^{+}3$	0:31:30	116	116	0.44
	Worldcup02	$150^{+}1$	0:36:42	156	151	0.73
	SM96-full	$408^{+}9$	2:16:06	435	421	3.54

Table 2: Different approaches for proper and non-proper MLVO with wide alignment scheme. The time is suitably given either in seconds or as hh:mm:ss. $\mathfrak{d}^* = X^{+}Y$ gives the final lower bound X and upper bound $X + Y$ of the verticality. The heuristic uses a random initial order and both local optimization schemes (starting with 2-opt) alternatingly. We give the best results after 50 and 500 independent runs; the time is specified as the total for 500 independent runs.

Tables 3 and 4 collect the results for the proper and non-proper graphs, respectively. Thereby, we always choose p and n such that the SDP matrix's size stays roughly the same, i.e., its dimension is 300–500.

For the proper graphs, the SDP relaxation is very tight and yields, in conjunction with the SDP rounding heuristic, surprisingly small gaps and very often even the optimal solution. The number of instances solved to optimality grow with growing density, while the according average absolute gaps dwindle. The local-search heuristic is clearly outperformed, independent of the graph density and especially for instances with many layers.

For the non-proper graphs, the absolute SDP gaps are much larger in general. While we solve some instances to optimality for graphs with $p \leq 6$ layers, hardly any instance could be solved to optimality for $p \geq 11$. Interestingly, in the non-proper case the local search heuristic clearly outperforms the SDP rounding heuristic on graphs with many layers, while the rounding heuristic seems preferable for graphs with few layers. All these observations seem to be quite independent of the graph density.

Non-proper graphs with varying densities												
$d =$			0.1		0.3		0.5		0.7		0.9	
			SDP	Heuristic	SDP	Heuristic	SDP	Heuristic	SDP	Heuristic	SDP	Heuristic
p	n	ζ	✓, t, **gap**	t, **ghs**	✓, t,**gap**	t, **ghs**	✓, t,**gap**	t, **ghs**	✓, t,**gap**	t, **ghs**	✓, t,**gap**	t, **ghs**
2	20	381	**1**, 24:24, **2.0**	1.6, **2.5**	**7**, 17:57, **0.5**	3.3, **0**	**9**, 16:46, **0.1**	4.9, **0**	**7**, 18:36, **0.4**	6.3, **0**	**10**, 15:29, **0**	7.0, **0**
	21	421	**0**, 29:37, **3.7**	1.9, **2.1**	**1**, 31:05, **2.5**	4.2, **0.1**	**6**, 23:46, **0.7**	6.1, **0**	**7**, 24:55, **0.3**	8.2, **0**	**9**, 20:42, **0.1**	8.3, **0**
	22	463	**1**, 37:45, **3.9**	2.3, **0.5**	**4**, 35:54, **1.5**	5.0, **0.1**	**3**, 35:49, **1.2**	8.1, **0**	**6**, 33:53, **1.4**	9.5, **0**	**9**, 24:55, **0.2**	10.0, **0.1**
3	16	361	**1**, 22:02, **2.7**	1.5, **0.8**	**6**, 17:06, **0.5**	3.1, **0.1**	**6**, 17:33, **0.8**	4.5, **0**	**7**, 17:35, **0.8**	6.1, **0**	**10**, 12:16, **0**	6.5, **0**
	17	409	**1**, 27:23, **3.3**	2.0, **2.2**	**4**, 26:49, **1.8**	3.9, **0.1**	**1**, 28.43, **1.9**	6.2, **0**	**8**, 22:34, **0.8**	8.0, **0**	**10**, 19:45, **0**	8.6, **0**
	18	460	**0**, 38:25, **3.7**	2.6, **2.0**	**3**, 35:46, **1.7**	4.9, **0**	**4**, 35:08, **2.2**	8.4, **0.1**	**4**, 33:29, **2.3**	9.8, **0.1**	**7**, 28:57, **0.3**	11.2, **0**
6	11	331	**0**, 19:05, **3.4**	1.0, **7.5**	**8**, 12:33, **0.5**	2.2, **1.5**	**9**, 14:03, **0.1**	3.1, **0.5**	**10**, 11:24, **0**	4.1, **0**	**10**, 10:52, **0**	4.4, **0**
	12	397	**0**, 26:56, **4.4**	1.4, **10.9**	**5**, 23:46, **1.1**	2.9, **2.2**	**8**, 20:11, **0.3**	4.3, **0.9**	**5**, 24:24, **0.8**	5.7, **0.1**	**10**, 17:00, **0**	6.0, **0**
	13	469	**0**, 40:26, **5.5**	2.0, **10.1**	**3**, 39:09, **2**	4.2, **3.3**	**5**, 35:20, **1.1**	6.3, **1.5**	**8**, 31:32, **0.5**	8.0, **0.4**	**7**, 33:56, **0.3**	8.6, **0.9**
11	8	309	**0**, 28:33, **6.7**	0.6, **4.5**	**10**, 12:23, **0**	1.4, **6.3**	**10**, 11:51, **0**	2.1, **2.7**	**10**, 12:49, **0**	2.5, **1.5**	**10**, 12:50, **0**	2.6, **2.4**
	9	397	**0**, 37:22, **6.6**	1.0, **14.1**	**10**, 23:21, **0**	2.2, **8.9**	**8**, 25:32, **0.5**	3.4, **6.1**	**10**, 23:57, **0**	4.3, **4.1**	**10**, 24:32, **0**	4.3, **3.0**
	10	495	**0**, 55:47, **9.5**	1.6, **20.2**	**6**, 43:24, **0.7**	3.5, **12.5**	**6**, 46:29, **0.4**	5.1, **6.8**	**8**, 49:32, **0.2**	6.5, **8.1**	**10**, 43:39, **0**	6.6, **2.7**
20	6	301	**0**, 30:01, **8.3**	0.5, **4.0**	**10**, 8:05, **0**	1.0, **12.7**	**10**, 9:45, **0**	1.5, **9.4**	**10**, 12:03, **0**	1.9, **3.3**	**10**, 14:18, **0**	1.8, **4.6**
	7	421	**0**, 49:53,**12.3**	0.8, **6.9**	**10**, 23:17, **0**	1.9, **21.9**	**10**, 25:01, **0**	2.8, **12.5**	**10**, 27:15, **0**	3.4, **13.6**	**9**, 32:11, **0.1**	3.5, **11.0**
	8	561	**0**,1:27:17,**23.2**	1.3, **10.1**	**10**,1:06:12, **0**	3.3, **38.6**	**10**,1:00:41, **0**	4.6, **24.3**	**10**,1:13:03, **0**	6.0, **19.1**	**10**,1:06:45, **0**	5.9, **14.8**

Table 3: Different approaches on random proper graphs with representatively chosen values for d, n and p. As before, ζ denotes the resulting dimension of the SDP matrix. "✓" denotes the number of instances solved to optimality (out of 10),"t" gives the average time (either in seconds or as hh:mm:ss) over the solved instances. "ghs" gives the gap between the local-search heuristic solution and the SDP upper bound; here, this value is always positive, i.e., the SDP upper bound is stronger than the local-search heuristic.

Non-proper graphs with varying densities																											
$d =$			0.1					0.3					0.5					0.7					0.9				
			SDP			Heuristic		SDP			Heuristic		SDP			Heuristic		SDP			Heuristic		SDP			Heuristic	
p	n	ζ	✓,	t,	**gap**	t,	**ghs**	✓,	t,	**gap**	t,	**ghs**	✓,	t,	**gap**	t,	**ghs**	✓,	t,	**gap**	t,	**ghs**	✓,	t,	**gap**	t,	**ghs**
3	16	361	**1**,	26:52,	**4.1**	1.9,	**0.7**	**7**,	23:15,	**0.5**	4.2,	**0.2**	**6**,	22:06,	**0.4**	6.2,	**0**	**9**,	21:56,	**0.2**	9.3,	**0**	**8**,	20:28,	**0.4**	10.4,	**0**
	17	409	**0**,	36:49,	**3.5**	2.5,	**0.3**	**3**,	32:47,	**1.5**	5.5,	**0**	**7**,	29:34,	**0.4**	8.7,	**0**	**5**,	32:25,	**1.0**	11.3,	**0**	**6**,	28:47,	**0.9**	13.4,	**0**
	18	460	**0**,	45:14,	**5.9**	3.1,	**2.1**	**4**,	42:08,	**1.7**	7.0,	**0**	**1**,	45:53,	**3.2**	10.5,	**0.1**	**1**,	46:44,	**2.1**	14.3,	**0**	**5**,	39:15,	**2.2**	16.8,	**0**
6	11	331	**3**,	21:00,	**1.4**	2.2,	**0.2**	**3**,	22:15,	**4.7**	5.2,	**0**	**5**,	21:49,	**3.9**	9.1,	**-0.6**	**2**,	23:20,	**1.8**	12.7,	**0**	**4**,	21:37,	**2.1**	15.4,	**0**
	12	397	**1**,	33:26,	**6.5**	2.9,	**1.3**	**1**,	35:03,	**8.0**	7.9,	**0.5**	**0**,	35:14,	**11.2**	13.1,	**-0.2**	**0**,	35:09,	**15.6**	17.3,	**-1.9**	**3**,	34:38,	**3.7**	22.0,	**0.1**
	13	469	**0**,	51:02,	**10.4**	4.1,	**2.8**	**0**,	50:58,	**18.9**	11.0,	**-2.4**	**0**,	52:15,	**20.9**	18.5,	**-1.5**	**0**,	53:16,	**13.8**	25.0,	**0**	**0**,	52:16,	**9.6**	30.2,	**-0.1**
11	8	309	**1**,	22:07,	**6.5**	2.3,	**0.3**	**1**,	21:12,	**12.3**	6.6,	**-0.4**	**0**,	23:08,	**18.7**	11.3,	**-0.8**	**0**,	23:28,	**22.8**	16.2,	**-8.0**	**0**,	22:27,	**5.4**	20.3,	**0.2**
	9	397	**0**,	34:30,	**15.4**	3.8,	**1.2**	**0**,	37:06,	**36.3**	10.7,	**-6.6**	**0**,	36:30,	**64.7**	19.6,	**-27.8**	**0**,	38:38,	**28.1**	26.3,	**0.3**	**0**,	37:00,	**28.4**	31.9,	**-1.5**
	10	496	**0**,	55:45,	**47.7**	5.9,	**-6.5**	**0**,	1:00:28,	**79.5**	17.7,	**-20.6**	**0**,	1:02:15,	**65.5**	29.6,	**-11.8**	**0**,	1:02:00,	**67.0**	43.1,	**-7.0**	**0**,	1:00:26,	**94.5**	47.5,	**-28.2**
20	6	301	**0**,	21:55,	**17.8**	3.5,	**0.3**	**0**,	23:31,	**48.6**	9.3,	**-4.3**	**0**,	22:57,	**59.3**	16.4,	**-14.6**	**0**,	22:24,	**73.8**	23.6,	**-18.3**	**0**,	21:53,	**57.2**	30.4,	**-20.4**
	7	421	**0**,	40:37,	**89.4**	5.9,	**-25.8**	**0**,	45:32,	**147.9**	18.1,	**-41.4**	**0**,	46:39,	**149.5**	33.1,	**-42.3**	**0**,	46:37,	**182.4**	45.7,	**-70.5**	**0**,	43:17,	**109.3**	60.0,	**-34.8**
	8	561	**0**,	1:26:48,	**110.8**	9.8,	**-21.7**	**0**,	1:42:55,	**264.3**	31.1,	**-67.1**	**0**,	1:42:54,	**292.7**	56.0,	**-84.4**	**0**,	1:38:15,	**227.2**	82.8,	**-34.7**	**0**,	1:42:42,	**259.5**	106.5,	**-109.5**

Table 4: Different approaches on random non-proper graphs with representatively chosen values for d, n and p. As before, ζ denotes the resulting dimension of the SDP matrix. "✓" denotes the number of instances solved to optimality (out of 10),"t" gives the average time (either in seconds or as hh:mm:ss.) over the solved instances. "ghs" gives the gap between the local-search heuristic solution and the SDP upper bound; this value is negative (red) if local search is stronger than the SDP bound.

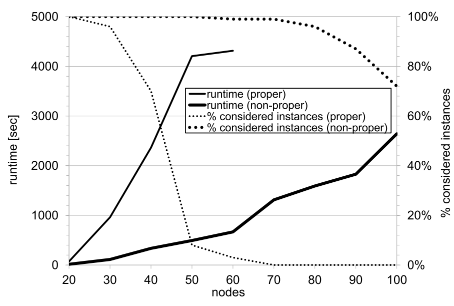
5.3 Real-world graphs

Next, we investigate our algorithms on two well-known larger benchmark sets, which were also considered in the context of (exact) multi-level crossing minimization [6]. The *Rome graphs* [8] contain 11,528 instances with 10–100 nodes and, although originally undirected, can be unambiguously interpreted as directed acyclic graphs, as proposed in [10]. The *North DAGs* [7] contain 1,158 DAGs, with 10–99 arcs. Consistent with [6], we consider two different ways of layering the graphs of both benchmark sets: the optimal LP-based algorithm by [14] and the layering resulting from compacting an upward planarization [4]. Both yield similar results in terms of MLVO running time and solvability. Our benchmark instances, except for very small graphs, are all sparse ($d \approx 0.1$) and have many layers ($p \geq 10$). Non-proper instances with narrow alignment scheme are quite trivial for nearly all Rome and North instances. Therefore we only consider the more challenging wide alignment scheme in the following.

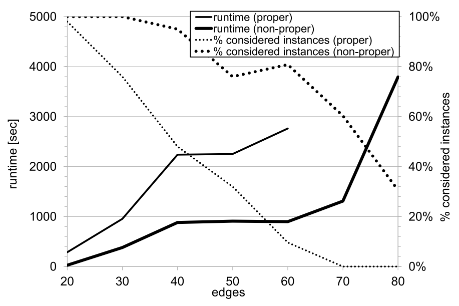
Our main finding is that the overall observation w.r.t. the heuristic variants hold. Yet, as the layerings introduce many more LEDs than the graphs considered before, the advantage of not requiring LEDs becomes even more pronounced: considering the largest graphs of the North DAGs (Rome graphs) with originally more than 90 edges (nodes), a single run of the heuristic requires only 6ms (2ms) on average, whereas the proper graphs require 1.8sec (0.8sec, resp.). Similarly, the SDP approach is applicable to all Rome and nearly all North graphs (98%) in the non-proper setting, as the approach works well up to $\zeta \approx 5000$. Using the proper drawing scheme, the SDP approach is applicable to 80% of the graphs with originally up to 60 nodes (Rome graphs) or 40 edges (North DAGs).

We again compare the upper bounds obtained by the SDP rounding heuristic with the local search heuristic *both* with random initial order and 500 independent runs in detail. Thereby, we disregard graphs with SDP dimension $\zeta > 1000$—considering larger dimensions would have been prohibitive in the context of the sheer amount of instances in the benchmark sets. However, observe that previous experiments (Table 2) considered instances with substantially larger ζ. Figures 3(a),3(b) give the average running time of the exact SDP algorithm dependent on graph sizes and drawing scheme. Figures 4(a)–4(d) show the gaps of the heuristics to the SDP lower bound, again considering different graph sizes and drawing schemes.

The running times and number of discarded instances grow strongly with the instance size for both Rome and North graphs. Considering the non-proper drawing scheme the local-search heuristic is clearly advantageous for both Rome and North graphs whereas SDP rounding heuristic performs better on proper graphs. These findings perfectly match the results for random graphs with $d = 0.1$ and $p \geq 10$ of the previous subsection.



(a) Rome graphs



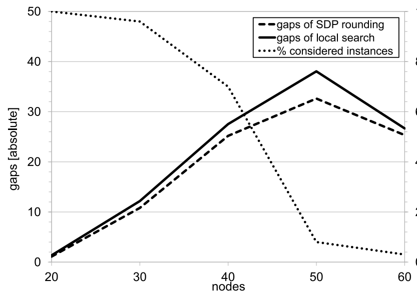
(b) North DAGs

Figure 3: Running times for Rome graphs and North DAGs, both in the proper and the non-proper setting. The right axes and dotted lines give the number of instances (relative to the original full instance set) not discarded due to too large constraint matrices. Thus the higher the percentage, the higher is the significance of the obtained average runtimes. Observe that the larger graphs are too large for the proper scheme, but are still applicable in the non-proper setting.

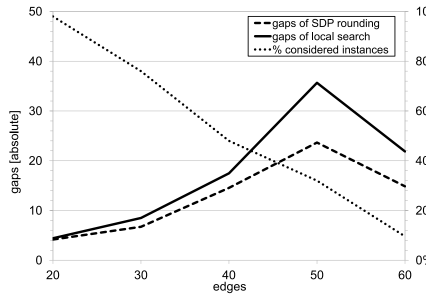
5.4 Combining MLVO with MLCM

Finally we compare the SDP approach for MLCM to its closest relative in the MLVO setting: proper MLVO with narrow alignment scheme; cf. Table 5. We observe that MLVO is harder than MLCM from an optimization point of view: in general, the MLVO-SDP requires more computation time and cannot close the optimality gap as often. Additionally we apply the SDP to MLVAC (MLVO after MLCM), i.e., proper MLVO with wide alignment scheme and fixed crossing minimal orderings for the non-PDs. MLVAC also yields surprisingly small gaps, but is essentially harder to solve than the other two problem types as using the wide alignment scheme increases the SDP's dimension. Comparing the results for MLVAC with the ones for proper MLVO from Table 2 shows that the optimal solutions of MLCM and MLVO are closely related but always different.

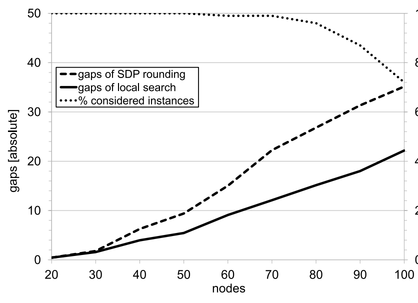
We conclude with some example figures. Figure 5 visually compares MLVO to MLVAC within the proper drawing style, showcasing the potential merit of verticality optimization over focusing on the crossing number. Already Figure 2 showcased the non-proper drawing style; Figure 6 shows the applicability of this drawing style, in conjunction with a near-optimal MLVO solution, for a large Rome graph.



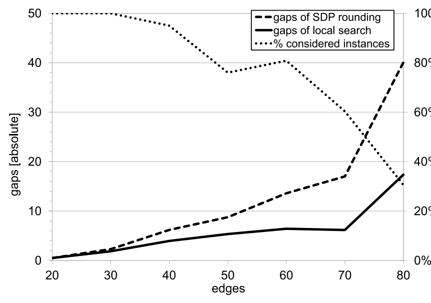
(a) Rome graphs, proper



(b) North DAGs, proper



(c) Rome graphs, non-proper



(d) North DAGs, non-proper

Figure 4: Absolute gaps between the heuristic solutions and the lower bound obtained by the SDP. Again, the right axes and dotted lines give the percentage of non-discarded instances.

		MLCM			MLVO			MLVAC	
	Instance	z^*	$\mathfrak{d}$	time	z	$\mathfrak{d}^*$	time	$\mathfrak{d}^*(z^*)$	time
Polyt.	Octahedron	80	264	10.66	81	$261^{+}1$	0:02:37	$243^{+}1$	0:11:49
	Dodecahedron	$393^{+}1$	3096	4:40:09	399	$3051^{+}27$	3:31:58	$1851^{+}15$	7:57:00
	Cube4	$1192^{+}3$	6594	7:10:19	1247	$6336^{+}86$	7:57:46	$5414^{+}32$	36:23:34
Gr.viz	unix	0	141	0.25	7	111	0:04:27	$86^{+}5$	1:01:46
	world	46	847	1:13:49	83	$620^{+}41$	6:33:10	$459^{+}29$	17:46:48
	profile	37	2767	0:53:34	75	$1303^{+}9$	7:09:51	$1363^{+}38$	178:54:16
Other	MS88	91	300	2.79	109	249	0:01:27	$154^{+}4$	0:48:26
	Worldcup86	49	762	25.3	72	559	0:05:43	$506^{+}5$	2:01:36
	Worldcup02	45	790	0:01:33	63	$501^{+}1$	1:24:56	$520^{+}9$	3:17:16
	SM96-full	162	1491	0:53:29	222	$1212^{+}13$	8:47:37	$711^{+}67$	61:30:45

Table 5: Comparing Proper MLVO with narrow alignment scheme with MLCM, and combining them to MLVAC. The columns z^* and $\mathfrak{d}^*$ give the optimal solutions (or final bounds) of MLCM and MLVO, respectively. The columns z, $\mathfrak{d}$ give the crossing number and non-verticality of the found solution. $\mathfrak{d}^*(z^*)$ gives bounds on the optimal non-verticality with assured minimal crossing number. The time is suitably given either in seconds or as hh:mm:ss.

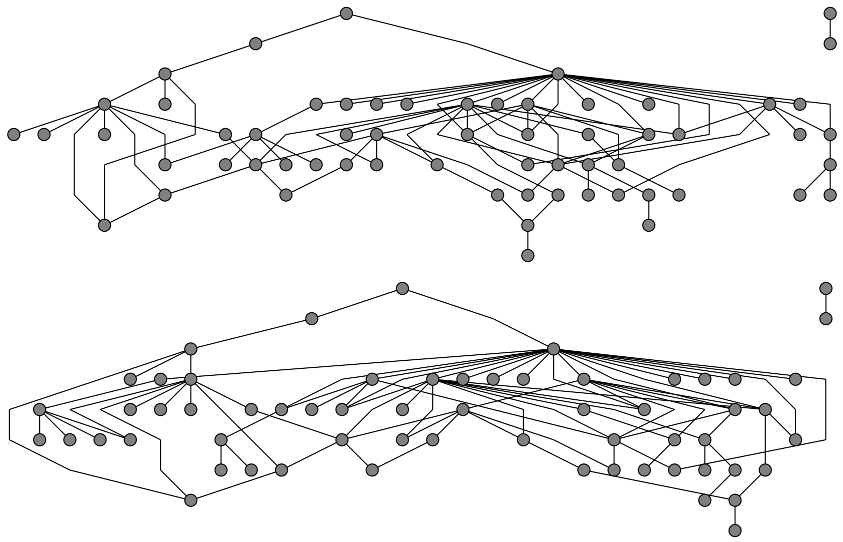


Figure 5: Instance *profile* drawn as a proper level graph (SDP upper bounds, not necessarily optimal). The top drawing optimizes the verticality (MLVO), whereas the bottom drawing optimizes the number of crossings (in fact, the latter solves MLVAC, i.e., it optimizes verticality within a crossing optimal solution).

6 Extensions

6.1 Edge-weights and drawing areas

In all the above approaches, including the SDP, it is straight forward to allow edge-weights. These can be used to model edges which are more important to be drawn relatively vertical than others, or to penalize non-verticalities for long edges more than for short ones (or vice versa) in the non-proper drawing scheme.

In practice, it can be interesting to consider other outer shape drawings than the rectangular array dominated by the width of the largest layer. Clearly, it is trivial to allow wider drawings, potentially resulting in less overall non-verticality by adding more PDs to the layers. Similarly, we can approximate any convex shape (e.g., a circular disc) by adding fewer or more PDs to the layers and shifting the first x-coordinate per layer via an offset, as suitable. We can model more general drawing shapes, including holes, by occupying any forbidden position p with a fixed-position PD u (yet note that edges may still

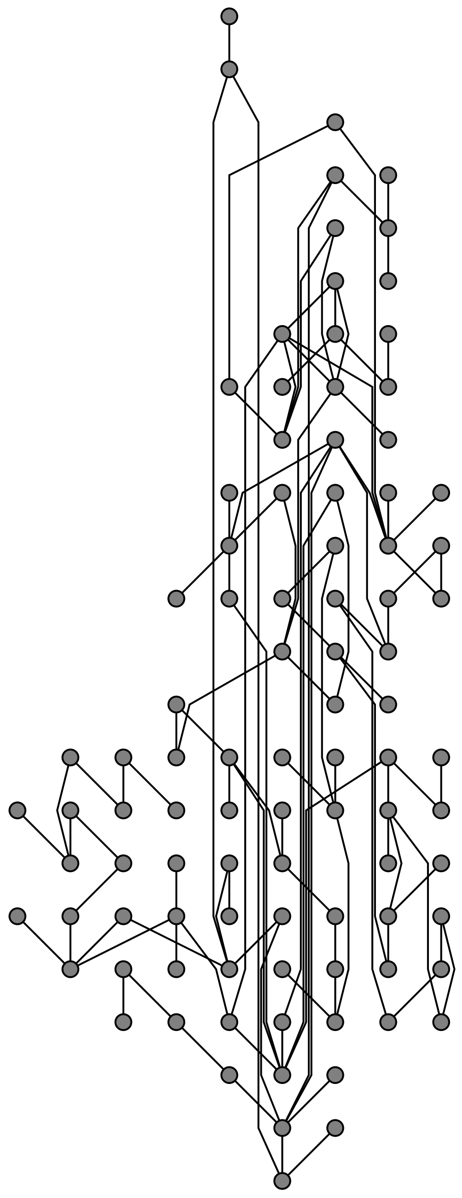


Figure 6: Rome graph No. 8861 (94 nodes) drawn with non-proper drawing style and near-optimal verticality (achieved by the SDP).

be routed close to these positions) by asking

$$\sum_{\substack{v \in V_{\ell(u)} \\ v \neq u}} y_{uv} = \omega + 1 - 2p + g_{\ell(u)}.$$

Thereby (and also in the following) we use the function $g_{\ell(u)} := (\omega - |V_{\ell(u)}|) \bmod 2$ to distinguish even and odd cases.

6.2 Monotonicity

In the non-proper drawing style we already observed that all edges are drawn (β-)monotonously along the x-coordinate, but this is not necessarily the case for proper drawings. While such a requirement is complicated to efficiently implement within our heuristic schemes, it is simple to include in the SDP approach. Conceptually, we require that, for all pairs of consecutive edge segments, their horizontal differences Δ_i, Δ_{i+1} do not have different signs, i.e., $\Delta_i \cdot \Delta_{i+1} \geq 0$.

More formally, let $e = (u, v) \in E$ be any original edge in the non-proper level graph G spanning k levels with the corresponding edge chain along the nodes $\langle u = u_0, u_1, u_2, \ldots, u_k = v\rangle$ in G'. Monotonicity of the edge is equivalent with feasibility of the following system of inequalities

$$[x(u_{i+1}) - x(u_i)][x(u_{i+2}) - x(u_{i+1})] \geq 0,\ i \in \{0, \ldots, k-2\}.$$

Substituting the x-terms therein via

$$x\,(u_i) = -\frac{1}{2} \sum_{\substack{v \in V_{m+i} \\ v \neq u_i}} y_{u_i v} + \frac{\omega + 1 - g_{\ell(u_i)}}{2},$$

yields feasible constraints on Z. We can further strengthen the semidefinite relaxation by additionally generating the analogous constraints for nodes on non-adjacent layers:

$$[x(u_i) - x(u_h)][x(u_l) - x(u_j)] \geq 0,\ h < i, j < l \in \{0, \ldots, k\}.$$

6.3 Node sizes

In many real-world scenarios, it can be interesting to consider nodes of varying size. Before, any node required exactly one grid point; generally, we may introduce nodes requiring $d_x \times d_y$ grid points. A horizontal stretch is easy to incorporate: when considering the absolute grid position of a node we not only compute the number of nodes to its left, but the sum of their horizontal stretches.

To incorporate vertical stretches, we copy the node on all its respective levels and connect them from level to level with dummy edges. Now, we only generate solutions where these dummy edges are strictly vertical and not crossed, both of which can be achieved in the SDP straight-forwardly: For the former, we simply add corresponding equalities. For the latter, let u be a node vertically stretched between layers ℓ_0 and ℓ_1, $x(u)$ its x-coordinate, and (w_0, w_1) any other edge with $\ell_0 < \ell(w_1) \leq \ell_1$ (i.e., a potentially crossing edge). We require

$$[x(w_0) - x(u)][x(w_1) - x(u)] \geq 0.$$

Interestingly, the introduction of node sizes itself makes the problem NP-hard. Recall from Theorem 2 that verticality optimization is polynomial time solvable for two layers, if the order on one of the layers is fixed. However we have:

Theorem 4 *The decision variant of MLVO with given node sizes is strongly NP-complete. This holds even when restricted to two levels, where the node order on one level is fixed but the nodes on the other level have given widths.*

Proof: We reduce from the strongly NP-complete 3-Partition problem, i.e., given a set A of $3m$ numbers $a_1, \ldots, a_{3m} \in \mathbb{N}$ with $\sum_{1\leq i\leq 3m} a_i = mB$, do there exist m disjoint subsets $S_1, \ldots, S_m$ of A such that the sum of the numbers of each subset is exactly B?

Consider the following MLVO instance with two levels, arising from some 3-Partition instance. For each a_i, $1 \leq i \leq n$, we introduce a node with horizontal stretch a_i to V_2. Then we add $2m-2$ further nodes $t_i \in V_1$, $u_i \in V_2$, $1 \leq i \leq m-1$, with horizontal stretch 1 and connect t_i with u_i. Finally we introduce mB PDs to V_1 such that the width of both levels is $(B+1)m-1$ and fix t_i, $1 \leq i \leq m-1$, at position $iB+1$.

An MLVO solution is clearly optimal if the nodes u_i, $1 \leq i \leq m-1$, are located at the positions $iB+1$ as the non-verticality of such a solution is zero. Hence we set M to zero. Now, if a solution with non-verticality zero is achievable, then the nodes u_i, $1 \leq i \leq m-1$, partition the numbers of the 3-Partition instance into m groups with equal sum B. Vice versa, if the 3-Partition instance is satisfiable, then an MLVO solution with non-verticality zero exists. $\square$

7 Conclusions and Further Thoughts

We suggested the concept of *verticality* as an explicit quadratic optimization goal. We showed first approaches to tackle the problem in practice and derived a new drawing style based on this concept; the latter allows to meaningfully abandon the graph-enlarging edge subdivision intrinsic to the traditional Sugiyama scheme. Our concept offers interesting further topics for research:

- In our test set, verticality-wise optimal drawings are typically very good w.r.t. the crossing number, and vice versa. Yet, it is an open (graph theoretic) question, how much these two measures can deviate in their respectively optimal drawings. In other words, how bad (in terms of crossing number) can a verticality-wise optimal drawing become, and vice versa?

- It seems worthwhile to investigate further, more involved, algorithms that close the gap between our simple heuristics and the computationally expensive SDP approach. Can, e.g., sifting-based algorithms like [2] be adopted to efficiently work for verticality optimization? Quite generally, MLVO seems to be an interesting playground to study how to adopt the extended research on MLCM algorithms to a new but related field.

- Intentionally, this article leaves one central question unanswered: Is a verticality-optimal drawing "better" in terms of perception than a crossing minimum drawing. Answering this question goes well beyond the scope of this paper: on the one hand it would require a well-constructed user study. On the other hand, such a study is not yet feasible: As noted above, we are still missing algorithms to obtain practically near-optimal solutions for graphs too large for our SDP. Only then, we can compare such results to (near-)optimal MLCM solutions in a meaningful way.

References

[1] A. Ahmed, X. Fu, S.-H. Hong, Q. H. Nguyen, and K. Xu. Visual analysis of history of world cup: A dynamic network with dynamic hierarchy and geographic clustering. In *VINCI'2009*, pages 25–39. Springer, 2010. doi:10.1007/978-1-4419-0312-9_2.

[2] C. Bachmaier, F. J. Brandenburg, W. Brunner, and F. Hübner. Global k-level crossing reduction. *JGAA*, 15(5):631–659, 2011. doi:10.7155/jgaa.00242.

[3] R. S. C. Matuszewski and P. Molitor. Using sifting for k-layer straightline crossing minimization. In *GD'1999*, LNCS 1731, pages 217–224, 1999. doi:10.1007/3-540-46648-7_22.

[4] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Layer-free upward crossing minimization. *ACM J. Experimental Algorithmics*, 15, 2010. doi:10.1145/1671970.1671975.

[5] M. Chimani and P. Hungerländer. Exact approaches to multi-level vertical orderings. *INFORMS Journal on Computing*, online-before-print id. 1120.0525, March 2013. doi:10.1287/ijoc.1120.0525.

[6] M. Chimani, P. Hungerländer, M. Jünger, and P. Mutzel. An SDP approach to multi-level crossing minimization. *Journal of Experimental Algorithmics*, 17:3.3:3.1–3.3:3.26, Sept. 2012. doi:10.1145/2133803.2330084.

[7] G. Di Battista, A. Garg, G. Liotta, A. Parise, R. Tamassia, E. Tassinari, F. Vargiu, and L. Vismara. Drawing directed acyclic graphs: An experimental study. *Int. J. Comp. Geom. and App.*, 10(6):623–648, 2000. doi:10.1142/S0218195900000358.

[8] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comp. Geom.*, 7(5–6):303–325, 1997. doi:10.1016/S0925-7721(96)00005-3.

[9] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972. doi:10.1145/321694.321699.

[10] M. Eiglsperger, M. Kaufmann, and F. Eppinger. An approach for mixed upward planarization. *Journal of Graph Algorithms and Applications*, 7(2):203–220, 2003. doi:10.7155/jgaa.00067.

[11] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of sugiyama's algorithm for layered graph drawing. *JGAA*, 9(3):305–325, 2005. doi:10.7155/jgaa.00111.

[12] I. Fischer, G. Gruber, F. Rendl, and R. Sotirov. Computational experience with a bundle method for semidefinite cutting plane relaxations of max-cut and equipartition. *Math.Prog.*, 105:451–469, 2006. doi: 10.1007/s10107-005-0661-9.

[13] G. Gange, P. J. Stuckey, and K. Marriott. Optimal k-level planarization and crossing minimization. In *GD'10*, LNCS 6502, pages 238–249, 2010. doi:10.1007/978-3-642-18469-7_22.

[14] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng.*, 19(3):214–230, 1993. doi:10.1109/32.221135.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[16] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:1115–1145, 1995. doi:10.1145/227683.227684.

[17] Graphviz gallery, http://www.graphviz.org/Gallery.php, Nov. 2010.

[18] P. Healy and A. Kuusik. The vertex-exchange graph: A new concept for multilevel crossing minimisation. In *GD'99*, LNCS 1731, pages 205–216, 1999. Extended version as TR UL-CSIS-99-1, Univ. Limerick, Ireland, 1999. doi:10.1007/3-540-46648-7_21.

[19] J.-B. Hiriart-Urruty and C. Lemarechal. *Convex Analysis and minimization algorithms (vol. 1 and 2).* Springer, 1993.

[20] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In *GD'97*, LNCS 1353, pages 13–24, 1997. doi:10.1007/3-540-63938-1_46.

[21] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. doi:10.1007/978-3-540-68279-0_8.

[22] M. Kaufmann and D. Wagner. *Drawing Graphs.* LNCS 2025, Springer, 2001.

[23] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955. doi:10.1002/nav.3800020109.

[24] S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Trans. Comput.*, 39:124–127, 1990. doi:10.1109/12.46286.

[25] M. May and K. Szkatula. On the bipartite crossing number. *Control and Cybernetics*, 72:85–97, 1988.

[26] J. Munkres. Algorithms for the assignment and transportation problems. *SIAM Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. doi:10.1137/0105003.

[27] P. Mutzel and R. Weiskircher. Two-layer planarization in graph drawing. In *ISAAC'1998*, LNCS 1533, pages 69–78, 1998. doi:10.1007/3-540-49381-6_9.

[28] L. Nachmanson, S. Pupyrev, and M. Kaufmann. Improving layered graph layouts with edge bundling. In *GD'2010*, LNCS 6502, 2010. doi:10.1007/978-3-642-18469-7_30.

[29] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *GD'1997*, LNCS 1353, pages 248–259, 1997. doi:10.1007/3-540-63938-1_67.

[30] F. Shieh and C. McCreary. Directed graphs drawing by clan-based decomposition. In *GD'95*, LNCS 1027, pages 472–482, 1996. doi:10.1007/BFb0021831.

[31] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Sys., Man, Cyb.*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.

[32] N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971. doi:10.1002/net.3230010206.

APPENDIX

An overview on the SDP approach

We sketch an exact method for (non-)proper MLVO[3] by analyzing matrix liftings of ordering problems. Let us introduce bivalent linear ordering variables assuming some fixed total order $\dot{<}$ of the nodes (e.g., based on their indices)

$$y_{uv} \in \{-1, 1\}, \qquad \forall u, v \in V_i,\ 1 \leq i \leq p,\ u \dot{<} v.$$

The variables shall be 1 if u is left of v and -1 otherwise. For notational simplicity, we also use the shorthand $y_{vu} := 1 - y_{uv}$ for $u \dot{<} v$. It is well-known that feasible orderings can be described via *3-cycle inequalities*

$$-1 \leq y_{uv} + y_{vw} - y_{uw} \leq 1, \qquad \forall u, v, w \in V_i, 1 \leq i \leq p,\ u \dot{<} v \dot{<} w. \tag{2}$$

Taking the vector y collecting the y_{uv}, we can define the multi-level quadratic ordering (MQO) polytope

$$\mathcal{P}_{MQO} := \operatorname{conv} \left\{ \begin{pmatrix} 1 \\ y \end{pmatrix} \begin{pmatrix} 1 \\ y \end{pmatrix}^{\top} \ :\ y \in \{-1, 1\},\ y \text{ satisfies (2)} \right\}.$$

Now the non-convex equation $Y = yy^{\top}$ can be relaxed to the constraint $Y - yy^T \succcurlyeq 0$, which is convex due to the Schur-complement lemma. Moreover, the main diagonal entries of Y correspond to y_{uv}^2, and hence $\operatorname{diag}(Y) = e$, the vector of all ones. To simplify our notation, we introduce

$$Z = Z(y, Y) := \begin{pmatrix} 1 & y^T \\ y & Y \end{pmatrix}, \tag{3}$$

where $\zeta := \dim(Z) = 1 + \sum_{i=1}^{p} \binom{|V_i|}{2}$ and $Z = (z_{ij})$. We have $Y - yy^T \succcurlyeq 0 \Leftrightarrow Z \succcurlyeq 0$. Hence, $\mathcal{P}_{MQO}$ is contained in the elliptope $\mathcal{E} := \{\ Z :\ \operatorname{diag}(Z) = e, Z \succcurlyeq 0\ \}$. In order to express constraints on y in terms of Y, we reformulate them as quadratic conditions in y. For (2) this gives

$$y_{uv}y_{vw} - y_{uv}y_{uw} - y_{uw}y_{vw} = -1, \qquad \forall u, v, w \in V_i,\ 1 \leq i \leq p,\ u \dot{<} v \dot{<} w. \tag{4}$$

We can assign a semidefinite cost matrix C to give $\mathfrak{d}(E)$ for any given feasible ordering y and can compute MLVO by solving $\mathfrak{d}^* = \min \{\ \langle C, Z \rangle\ :\ Z \in \mathcal{I}_{MQO}\ \}$, where

$$\mathcal{I}_{MQO} := \{\ Z\ :\ Z \text{ partitioned as in (3), satisfies (4)},\ Z \in \mathcal{E},\ y \in \{-1, 1\}\ \}.$$

By dropping the integrality of y, we get a basic semidefinite relaxation for MLVO that can be tightened in multiple ways, e.g. via metric- and Lovász-Schreiver cuts. See [5] for details.

[3]Both cases are virtually identical for the SDP approach.

To make the SDP computationally tractable, we only maintain the constraints $Z \succcurlyeq 0$ and $\operatorname{diag}(Z) = e$ explicitly, and deal with the other constraints via Lagrangian duality using subgradient optimization techniques (in particular, the bundle method [19, 12]). We obtain upper bounds via the hyperplane rounding method [16], supplemented by a repair strategy. Again, we refer to [5] for details. Therein, it is also shown that this approach clearly dominates—both theoretically and practically—other approaches based on linear or quadratic programs.