ALGORITHMS FOR NECKLACE MAPS

BETTINA SPECKMANN
Department of Mathematics and Computer Science, TU Eindhoven, Den Dolech 2
Eindhoven, 5612 AZ, The Netherlands
b.speckmann@tue.nl

KEVIN VERBEEK
Department of Computer Science, University of California, Santa Barbara
Santa Barbara, 93106, United States of America
kverbeek@cs.ucsb.edu

Received (received date)
Revised (revised date)
Communicated by (Name)

Necklace maps visualize quantitative data associated with regions by placing scaled symbols, usually disks, without overlap on a closed curve (the necklace) surrounding the map regions. Each region is projected onto an interval on the necklace that contains its symbol. In this paper we address the algorithmic question how to maximize symbol sizes while keeping symbols disjoint and inside their intervals. For that we reduce the problem to a one-dimensional problem which we solve efficiently. Solutions to the one-dimensional problem provide a very good approximation for the original necklace map problem.

We consider two variants: Fixed-Order, where an order for the symbols on the necklace is given, and Any-Order where any symbol order is possible. The Fixed-Order problem can be solved in \(O(n \log n)\) time. We show that the Any-Order problem is NP-hard for certain types of intervals and give an exact algorithm for the decision version. This algorithm is fixed-parameter tractable in the thickness \(K\) of the input. Our algorithm runs in \(O(n \log n + n2^K)\) time which can be improved to \(O(n \log n + nK^2)\) time using a heuristic. We implemented our algorithm and evaluated it experimentally.

Keywords: Necklace maps; scheduling; automated cartography.

1. Introduction

Necklace maps are a novel type of quantitative map which visualizes numerical data associated with regions. The regions of the underlying map are projected onto intervals on a one-dimensional closed curve (the necklace) that surrounds them. Symbols are scaled such that their area corresponds to the data of their region and placed without overlap inside the region’s interval on the necklace (Figure 1). Typical data sets for necklace maps include population, gross domestic product, or energy consumption.

Related work. There are several well-established thematic map types for quanti-
tative data associated with regions, specifically, choropleth maps, cartograms, and proportional symbol maps. However, all these maps suffer from limitations. Choropleth maps tend to overemphasize large regions and can generally only be used for data that is uniformly distributed within each region. Cartograms deform the underlying regions according to the data, which can make the map virtually unrecognizable if the data value differs greatly from the original area of a region or if data is not available at all for a particular region. Finally, proportional symbol maps can appear very cluttered with many overlapping symbols if large data values are associated with small regions. Necklace maps alleviate this latter problem by placing the symbols on a curve surrounding the regions. Necklace maps combine elements from proportional symbol maps and boundary labeling. They appear uncluttered and allow for comparatively large symbols sizes. In our InfoVis paper we discuss necklace maps from a visualization point of view and outline a framework to compute them fully automatically. In this paper we address the algorithmic question how to maximize symbol sizes while keeping symbols disjoint and inside their intervals on the necklace.

Proportional symbol maps place scaled symbols or diagrams directly on the input map, often on the centroid of the regions. The symbol, most commonly a disk or a square, is scaled such that its area corresponds to the data value of the region. Algorithms for proportional symbol maps have been studied by Cabello et al.\textsuperscript{5}.

Boundary labeling\textsuperscript{2,3,4} is frequently used in medical and technical drawings.Comparatively large labels are placed around an axis-parallel rectangle that contains the points or areas to be labeled. More recently this setting has been extended to place labels around a circle instead of a rectangle\textsuperscript{8}, with several different optimization criteria. Each label is connected to its site with a polygonal line, a leader, and no two leaders intersect. Labels tend to have the same (fixed) height and the main algorithmic problem is to find the optimal order of labels around the rectangle. Also necklace maps place symbols close to the boundary of the map. However, symbol sizes are neither equal, nor fixed, and geographic constraints for symbol
placements have to be obeyed. Furthermore, necklace maps do not use leaders, since the association between region and symbol is created via spatial proximity and color coding.

**Definitions and notation.** Before we can formally state the problem we first need to introduce some definitions and notation. To simplify the presentation we give all definitions for circular symbols, they naturally extend to other shapes. Furthermore, we consider only necklace maps with a single, circular necklace. All definitions and algorithms extend to multiple, star-shaped necklaces.

Our input is a set of polygons $P = \{P_1, \ldots, P_n\}$ (the geographic regions) and a set of data values $Z = \{z_1, \ldots, z_n\}$, where value $z_i$ ($z_i > 0$) is associated with polygon $P_i$. The data is normalized, that is, $\sum_{i=1}^{n} z_i = 1$. Furthermore, we are given a circular necklace $C$ with center $v = (x, y)$ and radius $R$. We denote the location of the center of a circular symbol $S_i$ on $C$ by an angle $\alpha_i$ relative to $v$, measured against the positive $x$-axis through $v$. We further denote the global scaling factor for all symbols by $\rho$.

We use two different methods to project a polygon $P_i$ onto a single contiguous interval $I_i = [a_i, b_i]$ on the necklace. $I_i$ represents all acceptable angles to place the symbol $S_i$ associated with $P_i$, counter-clockwise from $a_i$ to $b_i$. For a centroid interval we trace a ray from the center of the necklace through the centroid of $P_i$ which intersects $C$ at angle $\beta_i$. Then $I_i$ is defined as a constant-sized interval around $\beta_i$ for an appropriately chosen constant. For a wedge interval we consider an angle $\alpha$ to be acceptable for $S_i$ if the ray from $v$ at angle $\alpha$ passes through $P_i$. Then $I_i$ is the smallest interval containing all acceptable angles for $S_i$ (Figure 2). An exception is the polygon that contains $v$, for this polygon we always use a centroid interval. Although these interval definitions are quite simple, they do ensure a very good spatial relation between symbols and regions in practice. In our InfoVis paper we describe an additional type of intervals called density-dependent intervals. Density-dependent intervals depend on the data values. By enlarging the intervals on parts of the necklace that need to contain many (large) symbols, we can obtain larger symbols sizes. In this paper we focus only on centroid and wedge intervals. Figure 3 shows the effects of the three types of intervals. Motivated by our application, we assume that $|I| < \pi$ for all intervals $I$.

The above methods to obtain intervals result in essentially two types of intervals relevant for this paper. As will be described in Section 4.2, wedge intervals can be considered as arbitrary intervals, since every possible interval can occur as a wedge interval. On the other hand, centroid intervals always have a fixed size. Hence, in the remainder of this paper, wedge intervals are arbitrary intervals, whereas centroid
intervals are intervals with a fixed size.

A placement for a set of symbols $S_i$ is a set of angles $\alpha_i$ and a scaling factor $\rho$. A placement is feasible if (i) all $S_i$ are disjoint and (ii) $\alpha_i \in I_i$ for all $1 \leq i \leq n$. Since the area of each symbol is proportional to the data value of its region, symbol $S_i$ has radius $\rho \sqrt{z_i}$. The part of $C$ which is covered by $S_i$ is twice the covering radius $z'_i = z'_i(\rho)$ of $S_i$, which can be computed as $z'_i(\rho) = \arcsin(\frac{\rho \sqrt{z_i}}{R})$.

With the covering radii, we can describe our problem as a 1-dimensional problem. In the 1-dimensional problem, a placement is feasible if:

- $\alpha_i \in I_i$ for $1 \leq i \leq n$
- The (circular) intervals $[\alpha_i - z'_i, \alpha_i + z'_i]$ are disjoint.

Note that a feasible placement in the original problem is not necessarily a feasible placement in the 1-dimensional problem. However, an optimal solution to the 1-dimensional problem is generally a very good approximation of the optimal solution to the original problem. We make this more precise in Section 2. In the remainder of this paper, with exception of Section 2, we consider only the 1-dimensional problem. When we consider the problem on a circle, we actually mean this in the topological sense, rather than the geometric sense.

To simplify the description we assume in the following that for each symbol $S_i$ we are given a radius $r_i$ such that, for a given $\rho$, the covering radius is simply $z'_i(\rho) = \rho r_i$. All algorithms extend to the “real” covering radius at small additional cost.

**Formal problem statement.** A necklace map communicates its message via the sizes of its symbols. By definition these symbols are not allowed to overlap. The accuracy with which the map can be interpreted depends on the symbol sizes as well as the symbol locations. Acceptable symbol locations are captured by the intervals, hence each symbol center must be located inside its interval. This leads us to the following optimization problem.
Max-Size: Find the maximal global scaling factor $\rho$ such that there exists a feasible placement for the symbols.

We consider two variants of this problem:

**Fixed-Order:** We are given a fixed order in which the symbols must be placed on the necklace (for example determined by the centers of the intervals).

**Any-Order:** We can place the symbols in any order as long as the placement is feasible.

**Results.** In Section 3 we give an $O(n \log n)$ time algorithm to solve the Fixed-Order problem. The Any-Order problem can be cast as a scheduling problem. In Section 4 we first review related work, before proving that the Any-Order problem is NP-hard for wedge intervals. The status of Any-Order for centroid intervals is open. However, our algorithm for the Fixed-Order problem, when used with the order induced by the region centroids, results in a factor $\frac{1}{2}$-approximation for the Any-Order problem and centroid intervals. In Section 5 we present an exact algorithm to solve the decision version of the Any-Order problem for a given $\rho$. This algorithm is fixed-parameter tractable in the thickness $K$ of the input, that is, the maximal number of intervals any point on the necklace is contained in. Our algorithm runs in $O(n \log n + n^2K^4)$ time. For geographic maps $K$ rarely exceeds 10, making thickness a quite natural parameter choice. We also present a heuristic variant which runs in $O(n \log n + nK^2K^2)$ time. This heuristic found a feasible solution (if one existed) in all our experiments. Our InfoVis paper discusses the visual quality of our maps \cite{18}. To complement this, in Section 6 we experimentally evaluate some measurable properties of our algorithm, and compare the results and performance of different variants of our algorithm.

2. Reduction to 1-dimensional problem

In this section we prove how well the one-dimensional version of the problem approximates the original problem. Here we make the simplifying assumption that no symbol is allowed to cover the center of the necklace $v$.

**Theorem 1.** Let $\rho$ be the optimal scaling factor for some instance of the original problem, and let $\rho'$ be the optimal scaling factor for the same instance of the 1-dimensional problem. Then $\rho' \geq \frac{\sqrt{3}}{2} \rho$.

**Proof.** Consider the optimal solution of the original problem. Assume without loss of generality that the radius of $C$ is 1. A symbol with radius $r$ intersects the necklace $C$ in a circular interval of size $2 \arccos(1 - r^2/2)$. We can scale this symbol down until its covering radius matches up with $\arccos(1 - r^2/2)$, so that the space occupied by the symbol in the 1-dimensional problem matches the above circular interval. This requires a scaling factor of $\sqrt{3} - r^2/2$. This function is decreasing and, since
\( r \leq 1 \), the scaling factor must be at least \( \sqrt{3}/2 \). After scaling we obtain a feasible placement for the 1-dimensional problem, and hence \( \rho' \geq \sqrt{3}/2 \).

3. Fixed-Order

In this section we are given a fixed order in which to place the symbols on the necklace. We assume that the symbols are numbered in this particular order, with an arbitrary symbol as \( S_1 \). We first consider the problem on a line. Consider a subset of symbols from \( S_i \) to \( S_j \). The centers of all symbols \( S_k (i \leq k \leq j) \) must be placed between \( a_i \) and \( b_j \) (see Figure 4). Note that half of \( S_i \) and half of \( S_j \) can be outside of \([a_i, b_j]\). Let \( r_{ij} = \sum_{k=i}^{j} r_k \), then the scale factor for these symbols can be at most \( \rho_{ij} = \frac{b_j - a_i}{2r_{ij} - r_i - r_j} \). Let \( \rho^* = \min_{i<j} \rho_{ij} \), then, based on the arguments above, \( \rho^* \) is an upper bound for the optimal scale factor. In fact, it is the optimal solution.

Lemma 1. The optimal scale factor for the Fixed-Order problem on a line is \( \rho^* \).

Proof. Consider an optimal placement with scale factor \( \rho \) and move all symbols as far to the left as possible. There must be one symbol \( S_j \) that is placed on the rightmost position of its interval, namely \( b_j \). If this is not the case, \( \rho \) is not optimal, since we can slightly grow the symbols such that they stay in their intervals. Choose the largest \( i < j \) such that \( S_i \) is placed on the leftmost position of its interval, namely \( a_i \). If there is no empty space in the interval \([a_i, b_j]\), then \( \rho = \rho_{ij} \geq \rho^* \) and hence \( \rho^* = \rho \), since \( \rho^* \) is an upper bound for the optimal scale factor. Otherwise there is some empty space between a symbol \( S_k \) and \( S_{k+1} \). But then, since all symbols are moved to the left as far as possible, \( S_{k+1} \) must be placed on \( a_k \). This contradicts the choice of \( S_i \). \( \square \)

We compute \( \rho^* \) using the following divide and conquer algorithm. We first split the problem into two subproblems of roughly half the size, and recursively compute the optimal scale factor \( \rho_1 \) for the symbols \( S_1, \ldots, S_k \) and \( \rho_2 \) for the symbols \( S_{k+1}, \ldots, S_n \), where \( k = \lfloor n/2 \rfloor \). Note that \( \rho_1 \) is the minimum of all \( \rho_{ij} \) with \( i < j \leq k \) and \( \rho_2 \) is the minimum of all \( \rho_{ij} \) with \( k < i < j \). For the conquer step, we need to

Fig. 4. All centers of symbols \( S_2 \ldots S_6 \) must lie between \( a_2 \) and \( b_6 \).
compute the minimum of all $\rho_{ij}$ with $i \leq k < j$. Let $\rho$ be this minimum, then the optimal scale factor is given by $\rho^* = \min(\rho_1, \rho_2, \rho)$.

For a particular $\rho_{ij}$ with $i \leq k < j$ we can consider two different parts: the symbols $S_i$ to $S_k$, and the symbols $S_{k+1}$ to $S_j$. We define a set $\mathcal{L}$ of lines $\ell_i : \frac{x-a_i}{2r_i} - a_i$ for $i \leq k$, and a set $\mathcal{R}$ of lines $\ell_j : \frac{b_j-x}{2r_{j+1}} - b_j$ for $j > k$. The $x$-coordinates represent the location of the right side of symbol $S_k$ and the $y$-coordinates represent scale factors (see Figure 5). The $y$-coordinate of the intersection between a line $y = (x - a)/b$ and a line $y = (c-x)/d$ is given by $y = (c-a)/(b+d)$. Hence the $y$-coordinate of the intersection between $\ell_i \in \mathcal{L}$ and $\ell_j \in \mathcal{R}$ is given by $y = \frac{b_j-a_i}{2r_{j+1} - 2r_{j+1}/r_j} = \rho_{ij}$. Thus, we can compute $\rho$ by computing the lowest intersection between a line in $\mathcal{L}$ and a line in $\mathcal{R}$.

To compute the lowest intersection, we consider the dual problem. Let the dual of a point $p = (x, y)$ be the line $p^* : px - py$, and let the dual of a line $\ell : ax + b$ be the point $\ell^* = (a, -b)$. In the dual plane, the intersection between lines $\ell_i \in \mathcal{L}$ and $\ell_j \in \mathcal{R}$ corresponds to a line $p^*$ that passes through the points $\ell_i^*$ and $\ell_j^*$. The $y$-coordinate of this intersection is equal to minus the $y$-coordinate of the intersection between $p^*$ and the $y$-axis. So the problem in the dual plane is to find the highest intersection between the $y$-axis and a line through $\ell_i^*$ and $\ell_j^*$. For $\ell_i \in \mathcal{L}$ and $\ell_j \in \mathcal{R}$. Note that the $x$-coordinates of $\ell^*$ are positive for $\ell \in \mathcal{L}$ and negative for $\ell \in \mathcal{R}$. Now consider the upper hull of the points $\ell^*$ for $\ell \in \mathcal{L} \cup \mathcal{R}$. The unique line segment $\ell^\top$ of the upper hull that crosses the $y$-axis must have the highest intersection with the $y$-axis (no points can be above $\ell^\top$) and the endpoints of $\ell^\top$ correspond to a line $\ell_i \in \mathcal{L}$ and a line $\ell_j \in \mathcal{R}$. Finally note that the dual points $\ell^*$ for $\ell \in \mathcal{L} \cup \mathcal{R}$ are already sorted on $x$-coordinates, so the upper hull can be computed in $O(n)$ time. Thus, $\rho$ can be computed in $O(n)$ time.

**Lemma 2.** We can compute the optimal scale factor $\rho^*$ for the Fixed-Order problem on a line in $O(n \log n)$ time.

Next we consider the Fixed-Order problem on a circle. Unfortunately, a circular order does not give as much information as a linear order on a line. Consider a set of intervals restricted to one half of the circle, such that $I_n \subseteq I_{n-1} \subseteq \ldots \subseteq I_1$. For the corresponding problem on the line, the order must be $S_1, \ldots, S_n$. However, on the
Lemma 4. The optimal scale factor for \( \rho \) find a feasible placement given the optimal scale factor \( \rho \).

Proof. Consider an optimal placement for \( I \) this corresponds to a valid placement for \( \rho \) \( \geq \rho \) which means that \( i \) If \( a \) also be placed on \( S \) and \( \rho \) factor \( \rho \) \( S \) and \( a \). Let \( \rho^* \) be the optimal scale factor for \( I \) 2n symbols, where \( r_{i+n} = r_i \) and \( [a_{i+n}, b_{i+n}] = [a_i + 2\pi, b_i + 2\pi] \) for \( 1 \leq i \leq n \).

Lemma 3. Every valid placement for \( I_C \) corresponds to a valid placement for \( I_{\ell} \).

Proof. Consider a valid placement for \( I_C \) with angles \( \alpha_i \) for \( 1 \leq i \leq n \) and scale factor \( \rho \). This directly implies a placement for \( I_{\ell} \) with the same scale factor \( \rho \) and \( \alpha_{i+n} = \alpha_i + 2\pi \) for \( 1 \leq i \leq n \). This placement is valid if \( \alpha_i \leq \alpha_{i+1} \) for \( 1 \leq i < 2n \). Assume that \( \alpha_{i+1} < \alpha_i \) for some \( i \). This means that \( a_{i+1} \in I_i \), but also, since the placement for \( I_C \) is valid, that \( a_i \in I_{i+1} \). As \( |I_i| < \pi \) and \( |I_{i+1}| < \pi \), this is not possible.

Let \( \rho^* \) be the optimal scale factor for \( I_{\ell} \). For \( I_C \) we get one more upper bound, namely if the entire circle is filled with symbols: \( \rho_C = \pi/r_{1n} \).

Lemma 4. The optimal scale factor for \( I_C \) is the minimum of \( \rho^* \) and \( \rho_C \).

Proof. Consider an optimal placement for \( I_C \) with scale factor \( \rho \). By Lemma 3 this corresponds to a valid placement for \( I_{\ell} \), hence \( \rho \leq \rho^* \). We need to show that \( \rho \geq \min(\rho^*, \rho_C) \). To that end we show that any valid placement for \( I_{\ell} \) with scale factor \( \rho \leq \rho_C \) corresponds to a valid placement for \( I_C \). Move all symbols as far left as possible and let \( S \) be the last symbol that is placed on \( a_n \). If \( i > n \), then \( S_{i-n} \) must also be placed on \( a_{i-n} \), by construction of \( I_{\ell} \). Then the placement of the symbols \( S_{i-1} \) to \( S_{i-n} \) corresponds in a straightforward manner to a valid placement on \( I_C \). If \( i \leq n \), then \( S_{i+n} \) is not placed on \( a_{i+n} \) and there is no empty space between \( a_i \) and \( a_{i+n} \). But then \( \rho > \pi/r_{i(i+n-1)} = \pi/r_{1n} = \rho_C \). Hence we must have \( i > n \), which means that \( \rho = \min(\rho^*, \rho_C) \).

Theorem 2. Given the above assumptions, the Fixed-Order problem on a circle can be solved in \( O(n \log n) \) time.

Proof. By Lemma 4 the optimal scale factor is the minimum of \( \rho^* \) and \( \rho_C \), and by Lemma 2 we can compute this minimum in \( O(n \log n) \) time. All that remains is to find a feasible placement given the optimal scale factor \( \rho \). There must be a feasible
placement such that some symbol, say $S_i$, is placed on $a_i$. If we know $i$, then we can place $S_i$ on $a_i$ and then, in counter-clockwise order, place every next symbol as far clockwise as possible. This must result in a feasible placement. To determine $i$, we simply start by placing $S_1$ on $a_1$. Then, all symbols $S_1, \ldots, S_{i-1}$ must be placed further clockwise (or at the same angle) than in the feasible placement with $S_i$ on $a_i$, and hence we can place $S_i$ on $a_i$. Thus, after going through the circle twice, we must have found a feasible placement, which takes $O(n)$ time.

If we use the real covering radius $z'_i(\rho) = \arcsin(\frac{\sqrt{\rho R}}{R})$ instead of $z'_i(\rho) = \rho r_i$, then we cannot obtain the same running time as described above. The lines $\ell_i (i \leq k)$ and $\ell_j (j > k)$ are replaced by monotonically increasing and decreasing curves, respectively. In this case we cannot compute the intersection of two curves exactly. This justifies the use of a bisection search to obtain the optimal scale factor $\rho$. For a given scale factor, we can easily determine if there exists a feasible placement with that scale factor in $O(n)$ time, using the approach outlined in the proof of Theorem 2. Thus, we can compute the optimal scale factor with relative error $\epsilon$ in $O(n \log \epsilon^{-1})$ time.

4. Any-Order: NP-hardness and approximation

As mentioned before, the Any-Order problem can be cast as a scheduling problem. We hence first review related work in Section 4.1, before arguing that Any-Order is NP-hard for wedge intervals in Section 4.2. Furthermore, we show in Section 4.3 that our algorithm for the Fixed-Order problem, when used with the order induced by the region centroids, results in a factor $\frac{1}{2}$-approximation for the Any-Order problem and centroid intervals.

4.1. Scheduling

The Any-Order problem can be modeled as a scheduling problem as follows. Every symbol $S_i$ corresponds to a task $\tau_i$, the necklace corresponds to a single processor. The size of each symbol is modeled by the computation time $c_i$ of task $\tau_i$, while the intervals are represented by a release time $r_i$ and a deadline $d_i$ for $\tau_i$. That is, task $\tau_i$ cannot run before $r_i$ and must be finished before $d_i$ (Figure 6). Tasks might be interrupted (preemptive scheduling) but since symbols have to be placed contiguously we have to insist on non-preemptive scheduling. Finally, since we place symbols on a circle and not on a line we must consider periodic scheduling.

Not unexpectedly this scheduling problem is NP-hard in its full generality, even without the periodicity (which implies that periodic scheduling is NP-hard.
as well). Efficient algorithms do exist for special cases with restrictions on the release times and deadlines \(13,16\), restrictions on the computation times \(11\), or restrictions on the tasks themselves \(12\). Furthermore, there do exist heuristics \(7\) and exact branch-and-bound algorithms \(1\) that solve this problem. Our exact algorithm (see Section 5) is significantly simpler than the branch-and-bound approach and has the advantage of a provable running time. More recent results in the area of non-preemptive scheduling are focused more on multiprocessors and real-time systems and hence are no longer related to our problem.

4.2. Any-Order with wedge intervals is NP-hard

Given the fact that the scheduling problem described above is NP-hard and that wedge intervals allow us to create all possible intervals, it is not surprising that the Any-Order problem with wedge intervals is NP-hard as well. Assume we are given a task scheduling instance with tasks \(\tau_i\), \(r_i\), \(d_i\), and \(c_i\) as described above.

We can now compute intervals for our problem as \(I_i = [r_i + c_i/2, d_i - c_i/2]\) (we can assume that \(c_i \leq d_i - r_i\), otherwise the instance is clearly not feasible). The covering radius of a symbol \(S_i\) is then \(c_i/2\). Now we need to compute polygons \(P_1 \ldots P_n\) that generate the intervals \(I_1 \ldots I_n\) as wedge intervals. For that we scale the set of intervals to the first quadrant of \(C\) and use the construction depicted in Figure 7.

Corollary 1. The Any-Order problem with wedge intervals is NP-hard.

We should note one subtlety of this construction. We cannot generally compute the exact coordinates of the vertices of the polygons in polynomial time such that we get the required intervals exactly. However, as we can assume the values \(r_i\), \(d_i\), and \(c_i\) to be integers, the intervals (and also the covering radii) do not need to be exact. Hence, using appropriate rounding, we can still claim that the Any-Order problem with wedge intervals is NP-hard.

4.3. Any-Order with centroid intervals

The Any-Order problem with wedge intervals is NP-hard, but the status of the same problem with centroid intervals is open. Intuitively, one could assume that in an optimal solution all symbols should be ordered by the angles of the corresponding region centroids. However, this is not generally the case (Figure 8). We show below, though, that our algorithm for the Fixed-Order problem, when used with the order induced by the region centroids, results in a factor \(\frac{1}{2}\)-approximation for the Any-Order problem and centroid intervals. Note that this case is not the same as the scheduling problem studied by Kise et al. \(13\), because only the center of a symbol
must be in its interval, whereas in the problem of Kise et al. the whole “symbol”
must be in its interval, significantly changing the problem.

First consider the problem on a line instead of on a circle. Let the symbols $S_i$
be ordered as induced by the region centroids. Consider a subset of symbols from
$S_i$ to $S_j$. For both the Fixed-Order and the Any-Order problem, the centers of all
symbols $S_k$ ($i \leq k \leq j$) must be placed between $a_i$ and $b_j$, since $I_k \subseteq [a_i, b_j]$.
Recall that, for the Fixed-Order problem, half of $S_i$ and half of $S_j$ can be outside
of the interval $[a_i, b_j]$. Let $s_{ij} = 2r_{ij} - r_i - r_j$, then $\rho_{ij} = \frac{b_j - a_i}{s_{ij}}$. For the Any-Order
problem, these outer symbols could be any two symbols $S_k$ and $S_l$ ($i \leq k < l \leq j$).
Let $s'_{ij} = 2r_{ij} - r_k - r_l$, where $k$ and $l$ are chosen to minimize $s'_{ij}$, and let $\rho'_{ij} = \frac{b_j - a_i}{s'_{ij}}$.
Note that $\rho'_{ij}$ is an upper bound for the Any-Order problem.

Theorem 3. The Fixed-Order algorithm, used with the order induced by the region
centroids, computes a $1/2$-approximation of the Any-Order problem with centroid
intervals. The factor $1/2$ is tight.

Proof. Consider an instance $I_C$ of the Any-Order problem on the circle and
run the Fixed-Order algorithm on $I_C$ (note that the order of the $a_i$ is the
same as the order induced by the region centroids). If the resulting scale factor
is $\rho_C$, then this is clearly the optimal solution. Otherwise, by Lemma 4, the
resulting scale factor is $\rho^*$: the optimal scale factor for $I_L$ (with fixed order).
Every solution of the Any-Order problem for $I_C$ corresponds to a solution for $I_L$, so
it is sufficient to consider $I_L$. Note that, for any choice of $k$ and $l$, $s'_{ij} \geq r_{ij} \geq s_{ij}/2$.
Let $\rho_{opt}$ be the optimal scale factor for $I_L$ with any order.

$$\rho_{opt} \leq \min_{i,j|i<j} \frac{b_j - a_i}{s'_{ij}} \leq 2 \min_{i,j|i<j} \frac{b_j - a_i}{s_{ij}} = 2\rho^*,$$
and hence $\rho^* \geq \rho_{opt}/2$. To show the tightness of this factor, consider 4 symbols with
the same intervals $I_1 = [0, 1]$ and with $r_1 = r_4 = 0$ and $r_2 = r_3 = 1$. Then $\rho^* = 1/4$
(Figure 8 top), whereas $\rho_{opt} = 1/2$ (Figure 8 bottom).

5. Any-Order: Exact algorithm

In this section we describe an exact algorithm for the decision version of the Any-
Order problem for arbitrary intervals. That is, our algorithm decides for a given $\rho$
if there is a feasible placement for this $\rho$. We can use this decision algorithm in a
binary search for the optimal $\rho$ (which stops when a required precision is obtained).
The decision algorithm is fixed-parameter tractable in the thickness $K$ of the input,
that is, the maximal number of intervals any point on the necklace is contained in. For wedge intervals, the thickness $K$ is simply the maximum number of polygons that can be crossed by a ray originating from the center $v$ of the necklace $C$.

For geographic maps $K$ rarely exceeds 10, making thickness indeed a quite natural parameter. This parameter has been used before in the context of scheduling problems, where it is defined as the width of a partially ordered set $6,19$. The global approach by Colbourn and Pulleyblank $6$ can be adapted to solve our problem in $O(n^KK^2)$ time, but we present a more efficient algorithm for our specific case. The algorithm by Steiner $19$ cannot be used to solve our problem.

In the remainder of this section we first explain how to solve the Any-Order problem on a line, and then extend our algorithm to also work on a circle.

5.1. Any-Order on a line

Our algorithm starts by partitioning the intervals $I_i$ and their corresponding symbols $S_i$ into $K$ layers $L_j$ ($1 \leq j \leq K$), such that each layer contains intervals that are mutually disjoint. A simple algorithm that does this in $O(n \log n + nK)$ time is described in a book by Kleinberg and Tardos $14$, pp. 122–125. Note that this problem is closely related to the problem of coloring interval graphs. As interval graphs are perfect graphs, we need exactly $K$ layers.

**Dynamic programming.** After we have partitioned the intervals into $K$ layers, we can use dynamic programming to decide if a feasible solution exists. First we divide the space up into slices $d_j$, by slicing along all endpoints $a_i$ and $b_i$ of all intervals $I_i$. Note that every slice $d_j$ is of the form $[a_{i_1}, a_{i_2}]$, $[b_{i_1}, b_{i_2}]$, $[a_{i_1}, b_{i_2}]$, or $[b_{i_1}, a_{i_2}]$. If two endpoints $a_{i_1}$ and $b_{i_2}$ $(i_1 \neq i_2)$ coincide, then we place $b_{i_2}$ before $a_{i_1}$ and slice accordingly (possibly resulting in degenerate slices). In other cases the endpoints can be ordered arbitrarily. There are exactly $2n - 1$ slices (plus 2 unbounded slices).

Within a slice the set of intervals does not change. Furthermore, this set contains at most $K$ intervals which are all from different layers.

Now assume we are given a feasible solution. We take a subset $S$ of the symbols from the leftmost symbol up to some symbol $S_r$. Let the center of $S_r$, say $x_r$, be in the slice $d_j$. It is easy to see that for every interval $I_i$ that is interior disjoint from $d_j$, the symbol $S_i$ must be in $S$ if $I_i$ is to the left of $d_j$ and $S_i$ cannot be in $S$ if $I_i$ is to the right of $d_j$ (Figure 10 left). For all intervals $I_i$ with $d_j \subseteq I_i$, we cannot say anything, but there are at most $K$ of those. That means that all possible subsets $S$ can be represented by a slice $d_j$ and a bitstring $B$ with $B_k = 1$ if the interval from layer $L_k$ that intersects slice $d_j$ is in $S$ and $B_k = 0$ otherwise. We write this set as $S(j, B)$. Note that $d_j$ might not contain an interval from layer $L_k$, in which case $B_k$ must be zero. A simple computation now shows that there are at most $O(n2^K)$ of
these subsets $S$.

We now perform dynamic programming on these subsets $S(j, B)$. We define $F(j, B)$ as the rightmost point of the leftmost solution for the subset $S(j, B)$ (the right side of the rightmost symbol must be to the left as far as possible). If a feasible solution is not possible for a subset $S(j, B)$, then $F(j, B) = \infty$. We go through every slice from left to right and then, for every slice, we go through all (valid) bitstrings $B$ of size $K$, ordered on the number of ones in $B$.

For the first value, $F(1, (0, \ldots, 0)) = -\infty$. Another special case is the first value of every slice, namely $F(j, (0, \ldots, 0))$ ($j > 1$). In that case, there must be a bitstring $B^*$ such that $S(j, (0, \ldots, 0)) = S(j - 1, B^*)$. There are two cases (Figure 10 right). If $d_j$ is of the form $[a_{i_1}, a_{i_2}]$ or $[a_{i_1}, b_{i_2}]$, then $B_k^* = 0$ for all $k$. Otherwise, $d_j$ is of the form $[b_{i_1}, b_{i_2}]$ or $[b_{i_1}, a_{i_2}]$. We then need to make sure that $S(j - 1, B^*)$ contains symbol $S_{i_1}$. To achieve that, set $B_k^* = 0$ for all $k$ except for when $k$ is the layer of $I_{i_1}$. This way we can simply say that $F(j, (0, \ldots, 0)) = F(j - 1, B^*)$.

For the general case, i.e. for $F(j, B)$, we first try the same approach as for the first value of every slice. That is, we try to find a $B^*$ such that $S(j, B) = S(j - 1, B^*)$. The only case for which $B^*$ does not exist, is if $d_j$ is of the form $[a_{i_1}, a_{i_2}]$ or $[a_{i_1}, b_{i_2}]$, and $B_k = 1$, where $k$ is the layer of $I_{i_1}$. That is because a subset for slice $d_{j-1}$ cannot contain symbol $S_{i_1}$, because its interval $I_{i_1}$ is to the right of $d_{j-1}$. In this case, we make the following observation. The center of the rightmost symbol of any placement of $S(j, B)$ must be in $d_j$ or after it. That means that the last symbol must belong to one of the at most $K$ intervals of $d_j$. Also, if the interval of this last symbol is in layer $k$, then $B_k = 1$. Now we try the following for each value of $k$ for which $B_k = 1$. Let $S_i$ be the corresponding symbol on layer $k$. We set $B_k$ to zero, call the result $B^*$, and try to place $S_i$ after $F(j, B^*)$. We let $F(j, B)$ be the
rightmost point of the leftmost solution of the at most \( K \) options (Figure 11). If, for none of the options, we can place \( S_i \) in \( I_i \), then \( F(j, B) = \infty \).

If the final value \( F(2n, \langle 0, \ldots, 0 \rangle) < \infty \), then a feasible solution exists. It is easy to extend the algorithm to output such a feasible solution.

**Lemma 5.** The above algorithm correctly decides the existence of a feasible solution in \( O(n \log n + nK2^K) \) time.

**Proof.** If our algorithm finds a feasible solution, then clearly a feasible solution exists, because our algorithm makes sure all symbols are placed in their intervals and all symbols are disjoint. We show by induction on the number of ones in \( B \) that, if \( F(j, B) = \infty \), no feasible solution exists for subset \( S(j, B) \). The only interesting case is when we have to make a choice. As described above, we consider all possible subsets \( B^* \) excluding the last symbol. If \( F(j, B^*) = \infty \), then no feasible solution for \( S(j, B^*) \) exists by induction. Otherwise, the leftmost solution of \( S(j, B^*) \) is such that the center of the last symbol \( S_i \) must be placed after \( I_i \). Because \( F(j, B^*) \) is given by the leftmost solution, every placement of subset \( S(j, B^*) \) will cause the center of \( S_i \) to be outside of its interval. Because we try all options and all options fail, there is no feasible solution for subset \( S(j, B) \). The correctness of our algorithm now follows by induction.

The partitioning and slicing takes \( O(n \log n + n2^K) \) entries in the dynamic programming table that take \( O(K) \) time per entry to compute.

**Including partial orders.** Our dynamic programming algorithm allows for a partial order \( O \) to be imposed on the symbols. Say we require \( S_{i_1} \) to be left of \( S_{i_2} \). Clearly, a subset \( S \) that contains \( S_{i_2} \) but not \( S_{i_1} \) is not allowed. First assume that \( I_{i_2} \) has a part to the left of \( I_{i_1} \cap I_{i_2} \). Let \( d_j \) be a slice that is contained in this part and let \( L_k \) be the layer of \( I_{i_2} \). If \( B_k = 1 \) then \( S(j, B) \) contains \( S_{i_2} \) but not \( S_{i_1} \), so this option cannot be possible. To achieve this, we initially set all values \( F(j, B) = \infty \) for which \( B_k = 1 \). Now let \( d_j \) be a slice that is contained in \( I_{i_1} \cap I_{i_2} \) and let \( L_k \) be the layer of \( I_{i_1} \) and \( L_{k_2} \) be the layer of \( I_{i_2} \). Whenever \( B_{k_1} = 0 \) and \( B_{k_2} = 1 \), \( S(j, B) \) contains \( S_{i_2} \) but not \( S_{i_1} \), so this again cannot be possible. As before, we initially set all values \( F(j, B) = \infty \) for which \( B_{k_1} = 0 \) and \( B_{k_2} = 1 \). So the only feasible solutions we obtain cannot contain a subset that contains \( S_{i_2} \) but not \( S_{i_1} \) and hence the solution obeys the partial order.

To set all relevant values \( F(j, B) \) to \( \infty \) efficiently, we add each pair \((i_1, i_2)\) in the partial order to all slices for which the pair is relevant. We first compute, for each interval \( I_i \), which slices it covers. This can be computed in \( O(n \log n) \) time by performing a binary search for each interval. For each pair \((i_1, i_2)\) in \( O \) we can then easily determine the slices to which this pair must be added in \( O(1) \) time. Hence we can add all pairs to the relevant slices in \( O(|O| + m) \) time, where \(|O|\) is the number of pairs in the partial order and \( m \) is the total number of pairs added to all slices.
Note that $m = O(nK^2)$, as every slice contains at most $K$ intervals. For every slice, we can construct a lookup table in $O(K^2)$ time that stores which combinations in the bitstring $B$ are valid. Then we can go through all possible bitstrings recursively (one bit at a time) and, for every bit, check if there is a conflict with any of the preceding bits in $O(K)$ time. For all bitstrings $B$ we set $F(j, B) = \infty$ if a conflict was detected. This takes $O(K^2)$ time per slice, which results in a total running time of $O(|O| + n \log n + nK^2)$.

5.2. Any-Order on a circle

Extending the algorithm from the previous section to work on a circle is not trivial. The first problem arises when we want to partition a set of circular intervals into layers. In some cases we might need more layers than the thickness $K$ (Figure 12). Furthermore, finding the minimal number of layers for circular intervals is actually NP-hard. There is, however, a simple greedy algorithm that uses no more than $\ell - 1\ell - 2K$ layers, where $\ell$ is the size of the smallest subset of circular intervals that spans the entire circle. In general the number of layers found by the greedy algorithm is between $K$ and $2K - 1$, as shown by Tucker.

Fortunately we do not have to use either of these approaches. Below we describe two options that transform the Any-Order problem on a circle to the same problem on a line. The first is an exact algorithm whose running time is unfortunately much slower than the original algorithm on the line. The second is a heuristic variation on the same approach which is significantly faster. This heuristic found a feasible solution (if one existed) in all our experiments.

**Exact algorithm.** The basic idea is to cut the circle in one point, transform it into a line, and use the algorithm of Section 5.1 to find a solution if one exists. We then need to check if the whole solution fits on the circle (check rightmost symbol with leftmost symbol). However, we cannot just cut anywhere. The dynamic programming algorithm pushes all symbols to the left as far as possible, which is optimal on a line, but does not help on a circle. Hence we must cut the circle at the center of a symbol in a feasible solution. Consider a feasible solution and move all symbols in clockwise direction along the circle until the solution is no longer feasible. At this point, the center of one of the symbols $S_i$ must be placed on $a_i$. We do not know which symbol will have this property, so we have to try all of them. Now assume that we cut the circle at $a_i$ and place $S_i$ on $a_i$ (Figure 13). Every interval $I_j$ that contains $a_i$ is cut into two pieces. To run the dynamic programming algorithm, we need to choose which of these two pieces is used per interval $I_j$. At most $K$ of such intervals exist, so we have at most $O(2^K)$ choices per cut. Note that, after a cut and a choice is made, we can use the algorithm of Section 5.1 to
partition the intervals into at most $K$ layers (and we need to sort the intervals only once). Thus, the total running time is $O(n \log n + n^2 K 4^K)$.

**Heuristic variant.** A faster approach than the one described above is the following. We simply start at $a_1$ and ignore all intervals $I_j$ that contain $a_1$. Then we move $c$ times in counterclockwise direction around the circle and add intervals when we encounter them. Hence we are creating a problem instance on a line with $c$ copies of each of the original intervals (and symbols). Then we compute a feasible solution (if one exists) for this problem instance using the algorithm of Section 5.1. If no feasible solution exists for this constructed instance, then no feasible solution exists on the circle. Otherwise, we still need to find a feasible solution for the circle. We slide a window of size $2\pi$ over the solution of the constructed instance. If we find a window that contains all symbols, then this is a feasible solution for the circle (Figure 14). This last step runs in $O(cn)$ time. The total running time is $O(n \log n + cnK 2^K)$. In practice we find that a small value like $c = 5$ is more than sufficient to always find a feasible solution if one exists. Unfortunately we cannot guarantee that this heuristic always finds a feasible solution if it exists (for a given $\rho$), a counterexample for $c = 2$ is depicted in Figure 15.

![Figure 13. The circle is cut at $a_i$. We pick one part of the interval cut by the dashed line.](image)

![Figure 14. The heuristic variant (with $c = 2$).](image)

![Figure 15. The heuristic variant does not find a feasible solution for $c = 2$.](image)
6. Experimental evaluation

Proportional symbol maps (see Figure 16 right) can appear very cluttered, with many overlapping symbols that cover the underlying map, making it difficult to associate symbols with the correct regions. To judge this notion objectively, we performed experiments that measured the length of region boundaries covered. We measured this both for necklace maps (computed with centroid intervals) and for proportional symbol maps with symbols of the same size placed at region centroids. We also considered the amount of overlap among symbols of the proportional symbol maps: the overlap ratio. The overlap ratio is defined as the visible area (area of the union) of all symbols divided by the total area of all the symbols. Since the symbols on necklace maps are moved towards the outside of the map, the final map often has a larger area than a symbol map showing the same symbols. Hence we also measured the relative map area of necklace maps when compared with the area of proportional symbol maps. Finally, we computed the number of regions for which at least 75% of the boundary is covered by symbols. Such regions will likely be hard to recognize. We tested with two maps, East Asia & Pacific (Figure 17) and Western Europe (Figure 16), and several data sets for each map. We constructed the necklaces manually. The results are shown in Table 1.

Not surprisingly, the length of region boundaries covered by proportional symbol maps is much larger than for necklace maps. The overlap ratios of proportional symbol maps are generally not high, with only about 15% of the symbols covered. The increase in map area for necklace maps is comparatively small for the East Asia & Pacific map, thanks to the open necklace. But also for the “classic” Western Europe map the increase is still around 30%, which could be considered a small price to pay for the added readability of the map. There is a large difference between necklace maps and symbol maps if we consider the number of regions for which at least 75% of the border is covered by symbols. For necklace maps this number is nearly always zero, while for proportional symbol maps it is quite large compared to the number of symbols.
We implemented our algorithms in Java and evaluated the performance of our implementation. All necklace maps for the data sets in Table 1 and for all figures in this paper were computed in a few milliseconds. To be able to actually measure performance, we created a somewhat artificial data set, namely a world map with only one necklace. As we explain in the InfoVis paper, such data sets are unsuitable visually and should in fact always use several necklaces.

We used three different data sets and used density dependent intervals to obtain three different values for the thickness $K$ for each data set. The experiments were performed on a Pentium D 3 GHz processor (dual-core) with 1 GB of RAM. We computed the necklace maps using the heuristic variant described in Section 5.2 (with $c = 5$), and we used 30 steps in the binary search to obtain the optimal scaling factor. We also measured the time required for computing the intervals and performing the post-processing. The results are shown in Table 2, all running times are in seconds.

Optimizing the symbol sizes is by far the slowest step. The results follow the asymptotic bounds given in Section 5.2 well. This is problematic only for large $K$ and all maps we have encountered in practice have $K \leq 10$.

Finally we compared the results and performance of different variants of our algorithm. Specifically, we compared the Fixed-Order algorithm, the exact Any-Order problem, and the heuristic for different values of $c$. To compare the algorithms, we constructed synthetic data sets that bypass the underlying map as input. That is, we generate only sets of circular intervals with associated symbol sizes as input, for different values of $n$ and $K$. Symbol sizes (covering radii) are picked uniformly at random between 1 and 5. To generate a set of $n$ circular intervals with thickness $K$ (where $n$ is an integer multiple of $K$), we use the following approach. We
construct $K$ “layers”, and for each layer, we cut the circle into $n/K$ intervals. We then scale the intervals to obtain a certain coverage of the circle (per layer). In the experiments we use a coverage of 75%. This does not necessarily result in a set of circular intervals with thickness $K$, but generated sets with the incorrect thickness are omitted during the experiments. As above, we use 30 steps in the binary search to compute the optimal scaling factor. For each combination of $n$ and $K$ we perform 100 experiments. The average running times are shown in Table 3.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$n$</th>
<th>Fixed</th>
<th>Exact</th>
<th>$c = 1$</th>
<th>$c = 2$</th>
<th>$c = 3$</th>
<th>$c = 4$</th>
<th>$c = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>25</td>
<td>0.0010</td>
<td>0.2603</td>
<td>0.0049</td>
<td>0.0093</td>
<td>0.0149</td>
<td>0.0191</td>
<td>0.0234</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>0.0010</td>
<td>0.3259</td>
<td>0.0058</td>
<td>0.0114</td>
<td>0.0174</td>
<td>0.0235</td>
<td>0.0304</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>0.0012</td>
<td>0.4816</td>
<td>0.0069</td>
<td>0.0135</td>
<td>0.0206</td>
<td>0.0291</td>
<td>0.0374</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>0.0012</td>
<td>0.6414</td>
<td>0.0077</td>
<td>0.0153</td>
<td>0.0232</td>
<td>0.0335</td>
<td>0.0441</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>0.0011</td>
<td>0.9471</td>
<td>0.0105</td>
<td>0.0218</td>
<td>0.0326</td>
<td>0.0512</td>
<td>0.0734</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>0.0013</td>
<td>1.5322</td>
<td>0.0135</td>
<td>0.0273</td>
<td>0.0458</td>
<td>0.0720</td>
<td>0.1070</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>0.0015</td>
<td>2.0198</td>
<td>0.0154</td>
<td>0.0313</td>
<td>0.0543</td>
<td>0.0948</td>
<td>0.1485</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>0.0016</td>
<td>2.7870</td>
<td>0.0183</td>
<td>0.0400</td>
<td>0.0787</td>
<td>0.1214</td>
<td>0.2215</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>0.0013</td>
<td>3.7926</td>
<td>0.0247</td>
<td>0.0533</td>
<td>0.0966</td>
<td>0.1667</td>
<td>0.2808</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>0.0015</td>
<td>6.1335</td>
<td>0.0300</td>
<td>0.0701</td>
<td>0.1306</td>
<td>0.2568</td>
<td>0.3981</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>0.0018</td>
<td>8.5584</td>
<td>0.0337</td>
<td>0.0833</td>
<td>0.1534</td>
<td>0.3042</td>
<td>0.4503</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>0.0019</td>
<td>11.5538</td>
<td>0.0407</td>
<td>0.1024</td>
<td>0.2426</td>
<td>0.4178</td>
<td>0.6409</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>0.0016</td>
<td>14.4825</td>
<td>0.0573</td>
<td>0.1614</td>
<td>0.3713</td>
<td>0.6209</td>
<td>1.1754</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>0.0017</td>
<td>24.6669</td>
<td>0.0741</td>
<td>0.2186</td>
<td>0.5195</td>
<td>0.8611</td>
<td>1.4390</td>
</tr>
<tr>
<td>8</td>
<td>56</td>
<td>0.0019</td>
<td>34.0133</td>
<td>0.0874</td>
<td>0.2697</td>
<td>0.6364</td>
<td>1.1617</td>
<td>1.7459</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>0.0022</td>
<td>46.0455</td>
<td>0.1001</td>
<td>0.3239</td>
<td>0.7333</td>
<td>1.4334</td>
<td>1.8954</td>
</tr>
</tbody>
</table>

Table 3. Average running times (in seconds).
We should note that the running times have a fairly high variance, since the running time heavily depends on the input. If the set of intervals has thickness $K$ in only a small part of the circle, then the Any-Order algorithms run much faster than when the set of intervals has thickness $K$ everywhere. Nonetheless, the global picture should be apparent from the results of Table 3. The heuristic is much faster than the exact algorithm (at least for $c \leq 5$), and this difference in performance grows fast as $K$ increases.

Next we consider the quality of the results of the different algorithms. Here we simply look at the resulting scaling factor. Clearly the exact algorithm always finds the optimal scaling factor. In Table 4 we show how often the other algorithms find this optimal scaling factor. In Table 5 we consider the ratio between the obtained scale factor and the optimal scale factor, both the smallest ratio over all experiments and the average ratio (here we exclude algorithms that found the optimal scaling factor in each of our experiments).

| $K$ | $n$ | Fixed | $c = 1$ | $c = 2$ | $c = 3$ | $c = 4$ | $c = 5$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>25</td>
<td>4%</td>
<td>53%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>8%</td>
<td>62%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>9%</td>
<td>59%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>17%</td>
<td>67%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>11%</td>
<td>45%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>15%</td>
<td>50%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>7%</td>
<td>59%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>19%</td>
<td>60%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>8%</td>
<td>37%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>12%</td>
<td>49%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>15%</td>
<td>45%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>21%</td>
<td>64%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>8%</td>
<td>39%</td>
<td>97%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>15%</td>
<td>56%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>8</td>
<td>56</td>
<td>16%</td>
<td>49%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>11%</td>
<td>53%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 4. Percentage of experiments in which the optimal scaling factor has been found.

These experiments show that the heuristic performs very well if $c \geq 2$. For $c = 2$, the heuristic did not always find the optimal scaling factor, but the ratio is almost negligible. For $c > 2$, the heuristic always found the optimal scaling factor in all our experiments. For $c = 1$, the heuristic still found the optimal scaling factor in about 50% of the cases, but in some cases the ratio was very small. The Fixed-Order algorithm rarely found the optimal scaling factor, but the smallest ratio was
7. Conclusions and open problems

We studied the algorithmic question of optimizing symbol sizes for necklace maps. We presented several algorithms, both for the Fixed-Order and the Any-Order version of the problem. We implemented our algorithms and evaluated them experimentally. Our implementation also heuristically supports star-shaped necklaces (both open and closed) and multiple, disjoint or nested necklaces. These extensions are discussed in our InfoVis paper. Clearly the shape and location of the necklaces is very important for a necklace map. Currently we create our necklaces by hand and add them to the input map. We would like to be able to automatically generate suitable necklaces for given input maps and data sets, however, this seems to be a quite challenging algorithmic problem on its own. Finally, it would be interesting to explore animated necklace maps for time-varying data.

Acknowledgements. B. Speckmann and K. Verbeek were supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707 and VICI 639.023.208.

References

1. P. Baptiste, L. Peridy and E. Pinson. A branch and bound to minimize the number of late jobs on a single machine with release time constraints. European Journal of...

Bettina Speckmann and Kevin Verbeek


Bettina Speckmann and Kevin Verbeek