Fast and Flexible Overlap Detection for Chart Labeling with Occupancy Bitmap

Chanwut Kittivorawong* — University of Washington
Dominik Moritz† — Apple Inc.
Kanit Wongsuphasawat† — Apple Inc.
Jeffrey Heer* — University of Washington

Abstract

Legible labels should not overlap with other marks in a chart. The state-of-the-art labeling algorithm detects overlaps using a set of points to approximate each mark's shape. This approach is inefficient for large marks or many marks as it requires too many points to detect overlaps. In response, we present a *Bitmap-Based* label placement algorithm, which leverages *occupancy bitmap* to accelerate overlap detection. To create an occupancy bitmap, we rasterize marks onto a bitmap based on the area they occupy in the chart. With the bitmap, we can efficiently place labels without overlapping existing marks, regardless of the number and geometric complexity of the marks. This Bitmap-Based algorithm offers significant performance improvements over the state-of-the-art approach while placing a similar number of labels.

Index Terms: Human-centered computing—Visualization—Visualization techniques; Human-centered computing—Visualization—Information Visualization

1 Introduction

Text labels are important for annotating charts with details of specific data points. To be legible, labels should not overlap with other graphical marks in the chart. Since manual label placement can be tedious, prior work proposed automatic label placement algorithms (e.g., [7, 8, 11–13]). As the placement of each label can be arbitrary and depend on the placement of other labels in the chart, perfectly maximizing the number of placements is an NP-hard problem with respect to the number of labels to be placed. In practice, label placement algorithms need to strike a balance between achieving better performance (especially for interactive applications) and maximizing the number of labels placed.

To achieve interactive performance, many label placement algorithms (e.g., [7, 8]) use a greedy approach, instead of examining *all* combinations of label placements. To place each label, these algorithms first determine a list of preferred positions. They then place each label at a preferred position that is unoccupied. If all possible placements lead to overlaps, they omit the particular label. This greedy approach greatly reduces the search space to be linear with respect to the number of labels. However, detecting overlapping elements remains the bottleneck. A naïve overlap detection by comparing each position of a label with all placed labels yields an $O(n^2)$ runtime in a chart with n labels [2], which can be problematic for charts with many marks.

Particle-Based Labeling [7], the state-of-the-art fast labeling algorithm, accelerates overlap detection by simulating shapes as particles (collections of points) and comparing each label position only to particles in its neighborhood. This approach works well for charts that contain small shapes like scatter plots. However, for larger shapes, the algorithm needs to sample many points to simulate the shape's form, significantly increasing required computations for overlap detection. As the number of points to check with depends on the number of marks and their sizes, the required computation increases significantly for plots with many large marks.

In this paper, we aim to improve the performance of label placement algorithms with a more efficient way to detect overlapping elements. In addition, we aim to generalize the overlap detection technique so that it can be used with different types of charts. To achieve these goals, we make three contributions.

First, we present *occupancy bitmap*, which record if pixels on a particular chart are occupied, as a new data structure for fast label overlap detection. All graphical marks are rasterized to a bitmap to record the pixels that they occupy. This bitmap structure can leverage bitwise operators to quickly detect if a new label overlaps with any existing elements in the chart and update occupancy information after a new label is placed on the chart. With this approach, the cost to detect overlaps for a new label is fixed based on the chart size and size of the label, regardless of the number and the size of other graphical marks in the chart.

Second, we apply occupancy bitmaps to label various charts with different placement strategies including scatter plots, connected scatter plots, line charts, and cartographic maps.

Third, to evaluate our approach, we compare it to Particle-Based Labeling [7]. Our approach requires over *22% less* time to label a map of 3320 airports in the US and reachable airports from SEA-TAC airport, while placing a comparable number of labels. To facilitate this evaluation and the adoption of our method, we implement it as an extension to the Vega visualization tool [9].

2 Related Work

Prior work on automatic label placement has investigated different aspects of labeling including the optimization goal of the labeling algorithm, the method to detect overlapping marks, label positioning, priority of each label, and orientation of each label.

Existing approaches for placing labels often either prioritize visual quality or runtime performance. Several projects aimed to improve the visual quality of certain chart types by defining and optimizing certain quality metrics [1, 3, 6, 11, 14]. However, these approaches are not generalizable as these quality metrics are typically specific to the chart types. As the number of labels placed is important for giving more information to readers, the number of labels placed is often used as a proxy for visual quality. Some has applied several techniques such as simulated annealing [13] and 0-1 integer programming [12] to increase the number of labels placed. However, these approaches are slow as they iteratively adjust label layouts for better ones. To achieve interactive runtime performance, prior works use a greedy approach [7, 8]. These algorithms can place 10,000 labels within the order of milliseconds. Therefore, they are suitable for visualizing large data sets or interactive charts.

In general, a greedy label placement algorithm has two inputs: (1) a set of data points D to label, (2) a set of existing marks M that labels need to avoid. From these inputs, it takes the following steps to determine label placements:

1. Include all the marks M in a data structure O that stores occupancy information.

*e-mail: {chanwutk,jheer}@cs.washington.edu
†e-mail: {domoritz,kanitw}@apple.com. The work was done at the University of Washington.

2. For each data point in D:
   (a) Determines a list of candidate positions P nearby its corresponding marks, ordered by their preferences.
   (b) Find the most preferable position $p \in P$ that does not overlap with any mark as recorded in O.
   (c) If a non-overlapping position p exists, place the label at the position p and update O to include the label placed.

To determine candidate label positions for a mark, labeling algorithms often use 8-position model [5], generating candidate positions based on the four corners (e.g., top-left) and sides (top, bottom, left, and right) of the mark's axis-aligned bounding rectangle. Hirsch [4] extends this discrete positioning approach as a more generalized ”slider model”. This paper applies the standard 8-position model to generate candidate positions for different chart types and focus on accelerating overlap detection.

Since detecting overlapping marks is the bottleneck for label placement algorithms, prior work has investigated data structures to speed up overlap detection. The *trellis strategy* by Mote et al. [8] subdivides a chart into a two dimensional grid. To check if a label can be placed at a position, it checks the positions of other data points and their labels in neighboring grid boxes.

To generalize trellis strategy for arbitrary marks, Luboschik et al. presents Particle-Based Labeling [7], which represents a mark as a set of *virtual particles* that are sampled to cover the areas occupied by the mark. It then applies the trellis strategy to check for overlaps between the virtual particles instead of the actual marks. To sample particles from a mark, they propose two approaches. First, image-based sampling rasterizes all the marks in M onto an image and then samples particles from occupied pixels. Alternatively, the vector-based approach samples points to represent the contours of vector graphics of marks.

Particle-Based Labeling works for any kind of marks, but it is more efficient for detecting overlaps between labels and small marks. For large filled marks (such as an area in area chart), Particle-Based Labeling can be inefficient because it needs to represent a filled mark with many particles densely placed inside the mark's occupied area. Thus, checking whether the position of a label is occupied by any mark in a particular grid box is expensive. This paper presents a Bitmap-Based algorithm, which improves upon Particle-Based Labeling and can efficiently detect overlaps in charts with large filled graphical marks.

3 Fast Overlap Detection with Occupancy Bitmap

We now present an *occupancy bitmap* as a data structure to accelerate overlap detection, which is the bottleneck of label placement.

To accelerate overlap detection, an occupancy bitmap allows a label placement algorithm to efficiently check if a candidate position for a new label is previously occupied. Once a new label is placed, the labeling algorithm can also quickly update the occupancy bitmap to include the newly occupied area.

An occupancy bitmap is a two-dimensional bitmap of the same resolution as the screen-space (in pixel area) of the chart. Building on well-known bitmap (or bit arrays) structures [10], each bit in the occupancy bitmap records the occupancy of its corresponding pixel in the chart as shown in Fig. 1. A bit is set to one if its corresponding pixel is occupied and zero otherwise.

Occupancy bitmap provides two key benefits over the data structure used in Particle-Based Labeling. First, by using a bitmap to store occupancy information, the time required to check if placing a label at a certain position overlaps with any existing elements depends only on the chart size and label size, but does not depend on the complexity and the number of existing elements. Second, the bitmap structure leverages bitwise operators to accelerate two key operations for overlap detection: (1) The *lookup* operation checks if the area is partly occupied to decide whether the area is available for placing labels; (2) The *update* operation marks all bits in the area taken up by a new label placed as occupied.

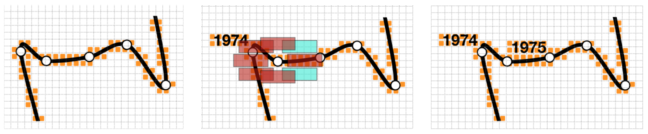

Figure 1: (Left) We rasterize connected scatter plot onto the bitmap to mark occupied pixels, shown in orange. (Middle) We use the 8-position model [5] to generate candidate positions for label placements. The cyan positions are available, while the red ones are not. (Right) After placing the label “1975”, the pixels under the label need to be marked as occupied.

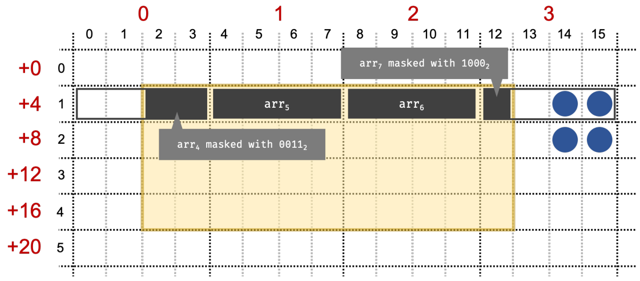

Figure 2: The black indices indicate the x/y coordinate of pixels in the chart. The red indices indicate the indices of the underlying array of the bitmap. For the purpose of demonstration, the bitmap is implemented on an array of 4-bit integers each representing a bit-string of length 4. The blue circles are marking occupied pixels. The yellow box is the area to lookup or update.

We implement the bitmap using a one-dimensional array of n-bits integers, in which each integer represents the bits of a contiguous subset of a row in the bit matrix. Thus, an integer in the array encodes the occupancy of n horizontally consecutive pixels in the chart.[1] For a chart with width w and height h, the occupancy of the pixel (x,y) is the bit at the position $((y \times w)+x) \bmod n$ of the integer at the array index $\lfloor \frac{(y \times w)+x}{n} \rfloor$. This bitmap layout is efficient because it supports looking up and updating a vector of bits simultaneously, instead of one bit at a time.

In the underlying array of the bitmap, there are two sets of integer entries that interact with the areas. First, I_f is the set of integer entries that are fully covered by the area, shown in the red column number 1 and 2 in Fig. 2. Second, I_p is the set of integer entries that are partly covered by the area, in the red column 0 and 4 in Fig. 2.

For lookup, the algorithm can check if each integer entry in I_f is zero. For each entry in I_p, the algorithm masks the entry with a bitwise-and operation to include only the bits inside the area, before checking if the masking result is zero. For example, arr_5 and arr_6 in Fig. 2 are in I_f. The integer value of each entry is 0000_2, meaning that the four pixels it represents are all unoccupied. arr_4 and arr_7 are in I_p. The integer value of arr_7 is 0011_2. The masking value is 1000_2 because only the leftmost bit is in the area. The masking result is $0011_2 \& 1000_2 = 0000_2$, meaning that the leftmost bit, which is inside the area, is unoccupied. The same process with different masking value is applied for the integer value of arr_4. Then, we can conclude that the bits from coordinate $(2,1)$ to $(12,1)$ are all unoccupied (zero). The process is repeated for row 1 to row 4 to check the whole rectangular area for the potential label position.

All the bits represented by each integer entry in I_f can be set as occupied simultaneously by setting the integer value of each entry to $11...11_2$. For each entry in I_p, the algorithm masks the entry with a bitwise-or operation to retain previous values of the

[1] In our JavaScript implementation, we use 32-bit integer as it is the largest available integer size in JavaScript.

bits outside of the area. For the example shown in Fig. 2, arr_5 and arr_6 are in I_f, each entry is set to 1111_2, meaning that four bits that it represents are all set to occupied. arr_4 and arr_7 are in I_p. The integer value of arr_7 is 0011_2. The masking value is 1000_2 because only the leftmost bits are in the area. The masking result is $0011_2 | 1000_2 = 1011_2$. The entry arr_7 is then set to 1011_2, meaning that the leftmost bit, which is inside the area, is set to occupied. Notice that the right three bits of arr_7 are kept as they were because the algorithm masks the integer entry with 1000_2 to retain their previous values. The same process with different masking value is applied for the integer value of arr_4. After running these steps, all bits from coordinate $(2,1)$ to $(12,1)$ are set to occupied. However, the algorithm does not repeat the process for all rows 1 to 4. Instead, it only marks the first, the last, and every $labelHeight_{min}$ row as occupied; $labelHeight_{min}$ is the height of the label that has the shortest height. So, if $labelHeight_{min} = 2$, this process repeats for row 1, 3, and 4. Updating fewer rows of bits speeds up update operations, while not losing any information about the area marked as occupied. A label of at least height $labelHeight_{min}$ that overlaps with the occupied area is guaranteed to overlap with at least one of the rows set to occupied.

Checking for overlap or marking an integer entry as occupied can be done in a constant number of bitwise-operations. These operations have constant runtime, regardless of the size of the integer. Our implementation uses the largest available integer size, to process many bits in parallel.

To record the areas of the marks for the labels to avoid, we rasterize all the marks in M onto the bitmap. Every pixel that is not fully transparent is considered occupied and its corresponding bit in the bitmap set to one. The number of bits used to represent marks is bounded by the size of the chart. Thus, the runtime for rasterization linearly depends on the chart resolution and number of the graphical marks. After the rasterization, a labeling algorithm using the *occupancy bitmap* can efficiently perform occupancy checks and updates. The runtime for an occupancy check or an update only depends linearly on the size of the label, regardless of the number and size of the marks that need to not overlap with labels.

4 Fast Overlap Detection for Labeling Charts

In this section, we apply fast overlap detection using the occupancy bitmap to place labels in scatter and connected scatter plots, line charts, and maps. The algorithm for placing labels is greedy, following the labeling steps described in Sect. 2. It first rasterizes all marks onto an *occupancy bitmap*. It then places all labels in one pass. For each data point to label, the algorithm iterates through the candidate label positions. It places the label at the first candidate position that does not overlap with any mark in the occupancy bitmap (skipping the remaining candidates). Before continuing with the next label, it marks the area taken by the label placed as occupied in the occupancy bitmap by marking the rectangular bounding box of the label (Fig. 2). The algorithm to add labels in these example chart types only differs in terms of (1) the graphical marks to be avoided by labels and (2) the candidate positions for labels.

For scatter and connected scatter plots, we use the 8-position model [5] to generate candidate positions around each point. For scatter plots, the marks to be avoided by labels include the point marks that represent records in the plot. For connected scatter plots, the marks include the points that represent records in the plots and the lines that connect them (Fig. 3).

In a line chart, each line includes a series of points and a path that connects all the points. Line charts are similar to connected scatter plots, but often one label represents a whole line instead of a single record. Therefore, the labeling algorithm may place one label per line, at the end of the line it represents. In this case, candidate positions include top-right, right, and bottom-right of the rightmost point of each line.

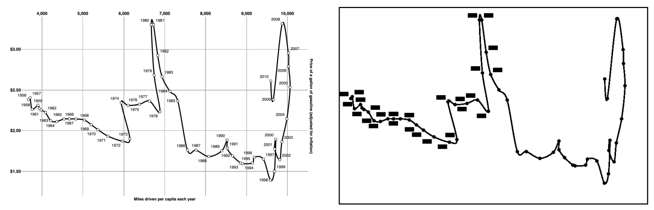

Figure 3: (Left) Labeled connected scatter plot. (Right) A snapshot of the bitmap when labeling the connected scatter plot. Here, a greedy labeling algorithm already placed labels in the left half of the chart.

As shown in Fig. 4, a map can contain points that represent locations, which need to be labeled, and paths that represent geographical features (e.g., country outlines). In this example, we also draw line segments to show paths between different locations. Similar to scatter plots, we use the 8-position model to generate candidate positions for maps.

5 Evaluation

To evaluate our labeling algorithm using *occupancy bitmap*, we compare it to Particle-Based Labeling [7], the current state-of-the-art fast labeling algorithm. To perform this comparison, we implemented both algorithms as transforms in Vega [9] and measure runtime and number of labels placed for each condition.

Our benchmark example is a map that shows airports in the US and travel routes between the Seattle-Tacoma airport (Sea-Tac) and other airports[2], as shown in Fig. 4. The dataset contains 3320 airports and 56 routes from Sea-Tac. In the chart, each black dot represents an airport with a route to Sea-Tac. A black line between the airport and Sea-Tac depicts the corresponding route. Red texts each in a red box are the labels representing names of airports that have a direct route to Sea-Tac. Meanwhile, a gray dot represents an airport without a direct route to Sea-Tac. The chart also outlines US states in light gray. In this benchmark, we run the algorithms to place labels (shown in teal) for airports without a direct route to Sea-Tac. Each airport contains eight candidate label positions (2 horizontal, 2 vertical, and 4 diagonal) around the airport location. The lines, points, red labels, and outline paths are placed before running the algorithm, acting as obstacles for the teal labels to avoid. To account for higher resolution displays, we test the algorithm with chart widths ranging from 1000 pixels up to 8000 pixels, with a fixed aspect ratio of 5:8.

For the baseline condition, we use the Particle-Based Labeling [7] with image-based sampling instead of vector-based sampling for two reasons. First, image-based sampling is a more practical approach to adopt in visualization tools because every standard graphic library can rasterize any mark types. Meanwhile, vector-based sampling requires separate implementations for different mark types. Second, the image-based approach is parameter-free. In contrast, the vector-based approach requires adjusting the sampling rate of particles to balance the fidelity against runtime performance.

We also notice that the mark rasterization process in Particle-Based Labeling has two issues. First, a particle that represents an occupied pixel is placed at the center of the pixel. This placement of particles may allow a label to slightly overlap with other marks by a half pixel, as shown in Fig. 4D. Second, the algorithm rasterizes every occupied pixel into a particle, which is unnecessarily too many. The number of particles used affects the runtime of the algorithm as overlap detection needs to compare a position to more particles.

We addressed these two issues in a version of Particle-Based Labeling, which we refer to as Improved Particle-Based Labeling.

[2]This map is originally from the Vega-Lite example gallery at `https://vega.github.io/vega-lite/examples/geo_rule.html`.

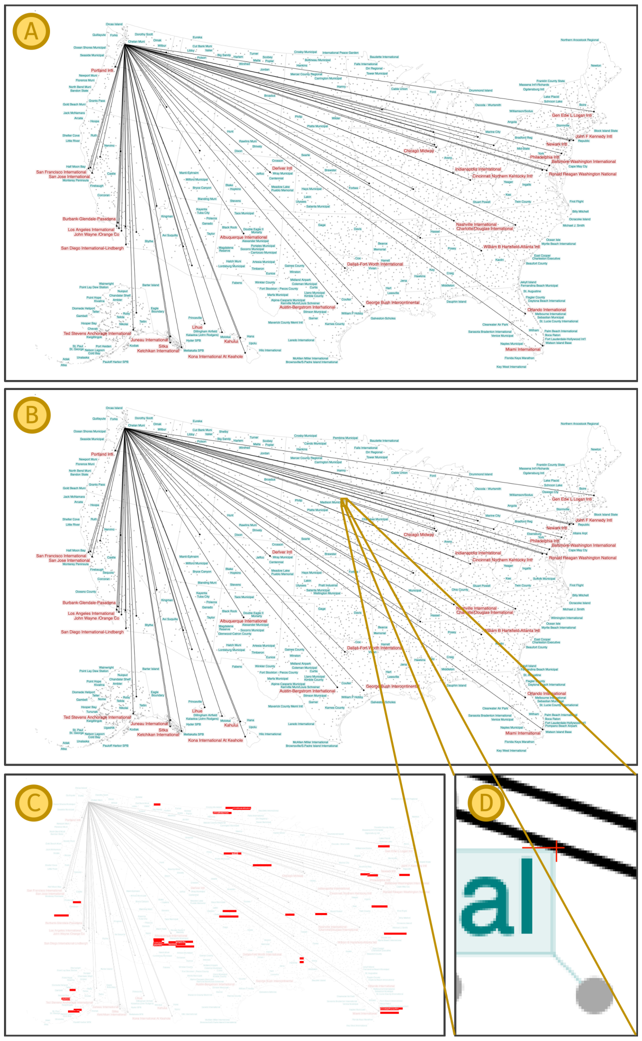

Figure 4: The labeling results from (A) our Bitmap-Based Labeling and (B) Particle-Based Labeling by Luboschik et al. [7]. (C) shows the visual difference between (A) and (B). The original Particle-Based Labeling may place a label that overlaps with existing marks by a half pixel. For example, the bounding box of the text's bounding box, as indicated with the red cross in (D), overlaps with a nearby line. Our Improved Particle-Based Labeling algorithm address this issue.

We addressed the first issue, a correctness issue, by placing particles at the four corners of an occupied pixel. Since a label's bounding box is an axis-aligned rectangle, it cannot overlap with an occupied pixel without overlapping with a particle at one of its corners first. We then address the runtime issue by omitting particles that are too close to others and thus are redundant. To do so, our improved algorithm rasterizes a mark in two phases. First, it rasterizes all particles along the outlines of the mark. Second, it rasterizes particles inside the marks for every other H_{min} pixels vertically and W_{min} pixels horizontally, where H_{min} is the height of the label with the shortest height and W_{min} is the width of the label with the shortest width. This optimization retains the algorithm's correctness, while greatly reducing the number of particles placed.

5.1 Performance

For each experimental condition (labeling algorithm and chart width), we run the task 20 times and calculate the median runtime and number of labels placed. Fig. 5 shows that the improved Particle-Based Labeling algorithm is faster than the original one as the chart size increases. Our Bitmap-Based algorithm performs significantly better than both the original and Improved Particle-Based Labeling algorithms, taking at least 22% less time to run across the chart sizes. The improvement also generally increases as the chart size increases.

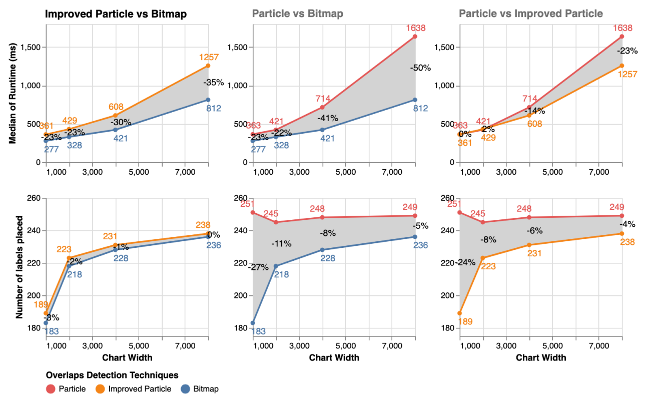

Figure 5: The runtime and the number of label placed by the Bitmap-Based algorithm, the original Particle-Based Labeling algorithm, and the Improved Particle-Based Labeling algorithm. The gray bands show the differences between conditions.

5.2 Number of Labels Placed

As we discussed earlier, the original Particle-Based Labeling may allow a label to overlap with a mark by a half pixel, thus it places significantly more labels than Improved Particle-Based Labeling and Bitmap-Based Labeling.

To avoid the effect of this correctness issue, we focus on the comparison of Bitmap-Based Labeling with Improved Particle-Based Labeling. Bitmap-Based Labeling placed 0.8% fewer labels for charts with 8000 pixels width and 3.2% fewer labels for charts with 1000 pixels width. Thus, we can conclude that Bitmap-Based Labeling can place a similar number of labels as Particle-Based Labeling if we only count labels that do not overlap with any marks.

6 Conclusion and Future Work

We present *occupancy bitmap*, a data structure that can efficiently detect overlaps between a label and other marks or labels in a chart. We apply this bitmap in a greedy label placement algorithm and apply it to label scatter plots, connected scatter plots, line charts, and maps. We compare this Bitmap-Based Labeling algorithm with the state-of-the-art Particle-Based Labeling algorithm, showing that the Bitmap-Based algorithm is significantly faster and can place similar numbers of labels in charts.

For future work, we plan to apply occupancy bitmaps to label other charts that need a different placement strategy other than the 8-position model used in this paper. For example, stacked area charts need a method to place a label inside each area shape.

For chart interactions like zooming or panning, a naïve greedy label placement algorithm may re-render label placements for every frame of animations, which can be too slow for large datasets. We plan to explore better optimization to avoid re-rendering in every new frame, while providing smooth interactions.

Acknowledgments

We thank all the three anonymous reviewers and the UW Interactive Data Lab for their helpful comments. This work was supported by a Moore Foundation Data-Driven Discovery Investigator Award and the National Science Foundation (IIS-1758030).

References

[1] L. Cmolik and J. Bittner. Real-time external labeling of ghosted views. *IEEE Transactions on Visualization and Computer Graphics*, 25:2458–2470, 07 2019. doi: 10.1109/TVCG.2018.2833479

[2] E. R. Gansner and Y. Hu. Efficient node overlap removal using a proximity stress model. In I. G. Tollis and M. Patrignani, eds., *Graph Drawing*, pp. 206–217. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[3] T. Götzelmann, K. Hartmann, and T. Strothotte. Agent-based annotation of interactive 3d visualizations. In *Smart Graphics*, 2006.

[4] S. A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

[5] E. Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[6] D. Kouril, L. Cmolik, B. Kozlikova, H.-Y. Wu, G. Johnson, D. Goodsell, A. Olson, E. Gröller, and I. Viola. Labels on levels: Labeling of multi-scale multi-instance and crowded 3d biological environments. *IEEE Transactions on Visualization and Computer Graphics*, 25:977–986, 01 2019. doi: 10.1109/TVCG.2018.2864491

[7] M. Luboschik, H. Schumann, and H. Cords. Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE Transactions on Visualization and Computer Graphics*, 2008.

[8] K. Mote. Fast point-feature label placement for dynamic visualizations. *Information Visualization*, 2007. doi: 10.1057/palgrave.ivs.9500163

[9] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *ACM User Interface Software & Technology (UIST)*, 2014.

[10] Wikipedia contributors. Bit array — Wikipedia, the free encyclopedia, 2020. [Online; accessed 12-July-2020].

[11] H.-Y. Wu, S. Takahashi, C.-C. Lin, and H.-C. Yen. A zone-based approach for placing annotation labels on metro maps. pp. 91–102, 07 2011. doi: 10.1007/978-3-642-22571-0_8

[12] S. Zoraster. The solution of large 0–1 integer programming problems encountered in automated cartography. *Operations Research*, 1990. doi: 10.1287/opre.38.5.752

[13] S. Zoraster. Practical results using simulated annealing for point feature label placement. *Cartography and Geographic Information Systems*, 1997. doi: 10.1559/152304097782439259

[14] L. Čmolík and J. Bittner. Layout-aware optimization for interactive labeling of 3d models. *Comput. Graph.*, 34:378–387, 2010.