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 novel label placement algorithm, which leverages \emph{occupancy bitmaps} to accelerate overlap detection.
To create an occupancy bitmap, we project all marks onto a bitmap based on the area they occupy in the chart.
The projection is image-based, so it supports projections of marks with any shape.
With the bitmap, we can efficiently place labels without overlapping existing marks,
regardless of the number and geometric complexity of the marks.
Our algorithm offers significant performance improvements over the state-of-the-art approach

while maintaining a similar level of visual quality.
```

```
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 \emph{Bitmap-Based} label placement algorithm, which leverages \emph{occupancy bitmaps} 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.
```

Introduction:

```
\firstsection{Introduction}
\maketitle

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 (\eg \cite{luboschik:particle,mote:informed-greedy,zoraster:int-program,zoraster:annealing}).

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 (\eg \cite{luboschik:particle,mote:informed-greedy}) use a greedy approach, instead of examining \emph{all} combinations of label placements.
To place each label, these algorithms first determine a list of candidate placements.
They then place each label at one of the positions without overlapping with any labels placed earlier.
If all possible placements lead to overlap, they omit the particular label.
This greedy approach greatly reduces the search space to be linear with respect to the number of labels.
Now, 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.

Its runtime still does not scale well for charts with many marks.

Particle-Based Labeling~\cite{luboschik:particle}, 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.
```

```
\firstsection{Introduction}
\maketitle

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 (\eg \cite{luboschik:particle,mote:informed-greedy,wu:zone,zoraster:int-program,zoraster:annealing}).
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 (\eg \cite{luboschik:particle,mote:informed-greedy}) use a greedy approach, instead of examining \emph{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~\cite{emden:prism}, which can be problematic for charts with many marks.

Particle-Based Labeling~\cite{luboschik:particle}, 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.
```

27	As the number of points to check with depends on the number of marks and their sizes, the required computation increases significantly for plot with many large marks.	26	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.
28		27	
29	In this paper, we aim to improve the performance of label placement algorithms	28	In this paper, we aim to improve the performance of label placement algorithms
30	with a more efficient way to detect overlapping elements.	29	with a more efficient way to detect overlapping elements.
31	In addition, we also aim to generalize the overlap detection technique so that it can be used with different types of charts.	30	In addition, we aim to generalize the overlap detection technique so that it can be used with different types of charts.
32	To achieve these goals, we make three contributions.	31	To achieve these goals, we make three contributions.
33		32	
34	First, we present \emph{occupancy bitmaps}, which record for each pixel on a particular chart whether it is occupied, as a new type of data structure for fast label overlap detection.	33	First, we present \emph{occupancy bitmaps}, which record if pixels on a particular chart are occupied, as a new data structure for fast label overlap detection.
35	All graphical marks are projected to a bitmap to record the area of pixels that they occupy.	34	All graphical marks are rasterized to a bitmap to record the pixels that they occupy.
36	This bitmap structure leverages 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.	35	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.
37	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.	36	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.
38		37	
39	Second, we apply occupancy bitmaps to label various charts with different mark types and placement strategies.	38	Second, we apply occupancy bitmaps to label various charts with different placement strategies including scatter plots, connected scatter plots, line charts, and cartographic maps.
40	We apply them to place labels (\ie position them around data points) in scatter plots, line charts, cartographic maps, and their combinations.		
41		39	
42	Third, to evaluate our approach, we compare it to Particle-Based Labeling.	40	Third, to evaluate our approach, we compare it to Particle-Based Labeling~\cite{luboschik:particle}.
43	Our approach requires over 22\% \emph{less} time to label a map of 3320 airports in the US and reachable airports from SEA-TAC airport,	41	Our approach requires over \emph{22\% less} time to label a map of 3320 airports in the US and reachable airports from SEA-TAC airport,
44	while producing labels with comparable visual quality measured as the number of labels placed.	42	while placing a comparable number of labels.
45	To facilitate this evaluation and the adoption of our method, we implement it as an extension to the Vega visualization tool~\cite{satyanarayan:vega}.	43	To facilitate this evaluation and the adoption of our method, we implement it as an extension to the Vega visualization tool~\cite{satyanarayan:vega}.
46		44	

Related Work:

1	\section{Related Work}	1	\section{Related Work}
2		2	
3	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.	3	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.
4		4	
5	Existing approaches for placing labels often either prioritize visual quality or performance.	5	Existing approaches for placing labels often either prioritize visual quality %, the number of labels placed,
		6	or runtime performance.
		7	Several projects aimed to improve visual quality of certain chart types by defining and optimizing certain quality metrics~\cite{ladislav:layout-aware, cmolik:ghost,timo:agent,kouril:level,wu:zone}.
		8	However, these approaches are not generalizable as these quality metrics are typically specific
		9	to the chart types.
		10	As the number of labels placed is important for giving more information to readers,
		11	the number of labels placed is often used as a proxy for visual quality.
6	Approaches that focus on visual quality include simulated annealing \cite{zoraster:annealing} and 0-1 integer programming \cite{zoraster:int-program}.	12	Some has applied several techniques such as simulated annealing \cite{zoraster:annealing} and
		13	0-1 integer programming \cite{zoraster:int-program} to increase the number of labels placed.
7	However, these approaches are slow as they repeatedly adjust labels layouts for better ones.	14	However, these approaches are slow as they iteratively adjust label layouts for better ones.
8	To achieve interactive runtime performance, prior works use a greedy approach \cite{luboschik:particle, mote:informed-greedy}.	15	To achieve interactive runtime performance, prior works use a greedy approach \cite{luboschik:particle, mote:informed-greedy}.
9	These algorithms can place 10,000 labels within the order of millisecond running time.	16	These algorithms can place 10,000 labels within the order of milliseconds.

```latex
Therefore, they are suitable for applications visualizing large datasets or in interactive visualizations.

In general, a greedy labeling algorithm has three inputs: (1) a set of existing marks $M$ that labels need to avoid; (2) a set of data points $D$ to label; and (3) a set of candidate position $P$ of each data point.
A candidate position is a position of a label relative to the position of the data point it represents (top, left, top-left \etc).
With these three inputs, a labeling algorithm has to annotate each data point in $D$ with a label at one of the positions in $P$ without overlapping with each other and marks in $M$.
To achieve this goal, a greedy labeling algorithm has two main steps:
\begin{enumerate}
  \vspace{-5pt}
  \item Record the areas that are occupied by the marks in $M$ in a data structure $O$ for overlap detection.
  \vspace{-5pt}
  \item Place a label for each data point in $D$, with two steps.
  \vspace{-5pt}
  \begin{enumerate}
    \item Find a position in $P$ that does not overlap with any mark recorded in $O$.
    \vspace{-2.5pt}
    \item Place the label and record the area that is occupied by the label as a rectangle mark in $O$.
  \end{enumerate}
\end{enumerate}

In the steps shown above, each of the mentioned greedy algorithms proposes different
(1) approaches to records the areas that are occupied by the marks in $M$,
(2) data structures $O$ to detect overlapping marks.

To record areas that all marks in $M$ occupy in a chart, Luboschik \ea \cite{luboschik:particle} propose an Image-Based approach and a Vector-Based approach.
The Image-Based approach projects all the marks in $M$ into an image of the same size as the chart.
Drawing marks into an image can be done efficiently with graphic libraries of major programming languages such as C/C++ (graphics.h), Java (awt), Python (PIL), and JavaScript (canvas).
Any pixel that lies within the area covered by a mark is considered occupied.
The Vector-Based approach samples points to represent contours of vector graphics of marks in $M$ and records it to $O$.
Our bitmap supports both approaches by projecting either occupied pixels or sampled points into the bitmap.

An efficient data structure to detect overlapping marks is an essential part of a labeling algorithm.
Different data structures have been proposed to speed up labeling algorithms.
```

```latex
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:
\begin{enumerate}
  \vspace{-5pt}
  \item Include all the marks $M$ in a data structure $O$ that stores occupancy information.
  \vspace{-5pt}
  \item For each data point in $D$:
  \vspace{-5pt}
  \begin{enumerate}
    \item Determines a list of candidate positions $P$ nearby its corresponding marks, ordered by their preferences.
    \vspace{-2.5pt}
    \item Find the most preferable position $p \in P$ that does not overlap with any mark as recorded in $O$.
    \vspace{-2.5pt}
    \item If a non-overlapping position $p$ exists, place the label at the position $p$ and update $O$ to include the label placed.
  \end{enumerate}
\end{enumerate}

To determine candidate label positions for a mark, labeling algorithms often use 8-position model~\cite{imhof1975positioning}, 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~\cite{hirsch1982algorithm} 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 \emph{trellis strategy} by Mote \ea \cite{mote:informed-greedy} 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 \ea presents Particle-Based Labeling \cite{luboschik:particle},
which represents a mark as a set of \emph{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
```

42

43 The \emph{trellis strategy} in Mote \ea \cite{mote:informed-greedy} subdivides a chart into a two dimensional grid.

44 To check if a label can be placed at a position, it compares the position of the label to the position of each data point or each data point and its placed label in neighboring grid boxes.

45 Unfortunately, this overlap detection approach with the \emph{trellis strategy} only supports labeling scatter plots.

46

47 Luboschik \ea \cite{luboschik:particle} present a data structure that represents all graphical marks as a collection of \emph{virtual particles} covering the marks.

48 To detect overlap, it checks for overlap with the \emph{virtual particles} instead of the actual marks.

49 Also, the chart is subdivided into a 2D grid,

50 where each grid box contains the \emph{virtual particles} that lie within the area of the box.

51 With this subdivision, the algorithm only needs to check for overlap against the \emph{virtual particles} inside the neighboring grid boxes.

52 This data structure is optimized for label-to-label overlap detection as the number of particles needed to represent each placed label is small.

53 Particle-Based Labeling focuses on labeling charts with small graphical marks (\eg scatter plots).

54

55 An aspect of this data structure that we wish to improve is its ability to efficiently represent and detect overlap with large filled graphical marks.

56 This data structure represents a filled mark with many \emph{virtual particles} placed densely inside the area that the mark occupies.

57 As a result, grid boxes can contain many \emph{virtual particles}.

58 So, checking whether the position of a label is occupied by any mark in a particular grid box is expensive.

59 As a result, the runtime of the algorithm overall is significantly increased.

60

61 In this paper, we present the \emph{occupancy bitmap} as the data structure for overlap detection in greedy labeling algorithms.

62 It focuses on quickly detecting overlap in charts with large filled graphical marks.

63

53 from occupied pixels.

54 Alternatively, the vector-based approach samples points to represent the contours of vector graphics of marks.

55

56 Particle-Based Labeling works for any kind of marks, but it is more efficient for detecting overlaps between labels and small marks. % such as overlaps between points in a scatter plot or between labels.

57 For large filled marks (such as an area in area chart), Particle-Based Labeling can be inefficient

58 because it needs to represent a filled mark with many particles densely placed inside the mark's occupied area.

59 Thus, checking whether the position of a label is occupied by any mark in a particular grid box is expensive.

60 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.

61

Technique:

1 \section{Fast Overlap Detection with Occupancy Bitmaps}

2

3 As we discussed earlier, the bottleneck of a label placement algorithm is to detect whether existing elements overlap with the new label to be placed.

4 This section presents the \emph{occupancy bitmap}, a data structure that we use to accelerate overlap detection.

5

6 Our occupancy bitmap is responsible for three parts of a label placement algorithm.

7 First, it records occupied areas of existing marks that labels need to avoid.

8 Second, it checks if a position of a label is available for placing (overlap detection).

9 And third, it records occupied areas of placed labels (occupancy updates).

10

11 An occupancy bitmap is a two-dimensional bitmap of the same size as the chart.

12 Building on well-known bitmaps~\cite{wiki:bitarray}, each bit in the occupancy bitmap records the occupancy of its corresponding pixel in the chart as shown in

13 \autoref{fig:place_label}.

1 \section{Fast Overlap Detection with Occupancy Bitmaps}

2

3 We now present an \emph{occupancy bitmap} as a data structure to accelerate overlap detection, which is the bottleneck of label placement.

4

5 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.

6 Once a new label is placed, the labeling algorithm can also quickly

7 update the occupancy bitmap to include the new occupied area.

8

9 An occupancy bitmap is a two-dimensional bitmap of the same resolution as the screen-space (in pixel area) of the chart.

10 Building on well-known bitmap (or bit arrays) structures~\cite{wiki:bitarray}, each bit in the occupancy bitmap records the occupancy of its corresponding pixel in the chart as shown in

11 \autoref{fig:place_label}.

A bit is set to one if its corresponding pixel is occupied and zero otherwise.

Occupancy bitmaps provide 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 overlap detection and occupancy updates.

We implement the bitmap using a one-dimensional array of n-bits integers, in which each integer is a bitstring representing

a subpart of a row in the bit matrix.
Thus, an integer in the array encodes the occupancy of n-horizontally consecutive pixels in the chart.\footnote{In our JavaScript implementation, we use 32-bit integer as it is the largest available integer size in JavaScript.}
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.
Both operations are performed on a rectangular area.
(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 placed label as occupied.

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 \autoref{fig:bitmask}.
Second, I_p is the set of integer entries that are partly covered by the area, in the red column 0 and 4 in \autoref{fig:bitmask}.

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 \
\autoref{fig:bitmask} 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 $arr[4]$ is 0000_2.
The masking value is 0011_2 because only the right two bits are in the area.
The masking result is $0000_2 \& 0011_2 = 0000_2$, meaning that the two bits inside the area, are all unoccupied.
The integer value of $arr[7]$ is 0011_2.
The masking value is 1000_2 because only the leftmost bits 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.

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 bits outside of the area.

A bit is set to one if its corresponding pixel is occupied and zero otherwise.

Occupancy bitmaps provide 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 \emph{lookup} operation checks if the area is partly occupied to decide whether the area is available for placing labels;
(2) The \emph{update} operation marks all bits in the area taken up by a new label placed as occupied.

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.\footnote{In our JavaScript implementation, we use 32-bit integer as it is the largest available integer size in JavaScript.}
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 \autoref{fig:bitmask}.
Second, I_p is the set of integer entries that are partly covered by the area, in the red column 0 and 4 in \autoref{fig:bitmask}.

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 \
\autoref{fig:bitmask} 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 bits outside of the area.

```
48 For the example shown in \autoref{fig:bitmask}, $arr[5]$ 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.
49 $arr[4]$ and $arr[7]$ are in $I_p$.
50 The integer value of $arr[4]$ is $0000_2$.
51 The masking value is $0011_2$ because only the right two bits are in the area.
52 The masking result is $0000_2 | 0011_2 = 0011_2$.
53 The entry $arr[4]$ is then set to $0011_2$, meaning that the two bits inside the area, are all set to occupied.
54 The integer value of $arr[7]$ is $0011_2$.
55 The masking value is $1000_2$ because only the leftmost bits are in the area.
56 The masking result is $0011_2 | 1000_2 = 1011_2$.
57 The entry $arr[7]$ is then set to $1011_2$, meaning that the leftmost bit, which is inside the area, is set to occupied.
58 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.
59 After running the steps above, all bits from coordinate $(2, 1)$ to $(12, 1)$ are set to occupied.
60 However, the algorithm does not repeat the process for all rows 1 to 4.
61 Instead, it repeats for the first, the last, and every $labelHeight_{min}$ row where $labelHeight_{min}$ is the height of the label that has the shortest height.
62 So, if $labelHeight_{min}=2$, this process repeats for row 1, 3, and 4.
63 Updating fewer rows of bits speeds up update operations, while not losing any information about the area marked as occupied.
64 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.
65
66 Checking for overlap or marking an integer entry to occupied can be done in a constant number of bitwise-operations.
67 These operations have constant runtime, regardless of the size of the integer.
68 Our implementation uses the largest available integer size, to process many bits in parallel.
69
70 To record the projection of the marks that labels need to avoid (marks in $M$).
71 First, we draw all the marks in $M$ into a canvas.
72 Every pixel that is not fully opaque is considered occupied and its corresponding bit in the bitmap set to one.
73 The number of bits used to represent graphical marks is bounded by the size of the chart.
74 Therefore, our labeling algorithm using the \emph{occupancy bitmap} can perform occupancy checks and updates consistently,
75 regardless of the number and size of the graphical marks that need to not overlap with labels.
76
```

```
43 For the example shown in \autoref{fig:bitmask}, $arr_5$ 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.
44 $arr_4$ and $arr_7$ are in $I_p$.
45 The integer value of $arr_7$ is $0011_2$.
46 The masking value is $1000_2$ because only the leftmost bits are in the area.
47 The masking result is $0011_2 | 1000_2 = 1011_2$.
48 The entry $arr_7$ is then set to $1011_2$, meaning that the leftmost bit, which is inside the area, is set to occupied.
49 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.
50 The same process with different masking value is applied for the integer value of $arr_4$.
51 After running these steps, all bits from coordinate $(2, 1)$ to $(12, 1)$ are set to occupied.
52 However, the algorithm does not repeat the process for all rows 1 to 4.
53 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.
54 So, if $labelHeight_{min}=2$, this process repeats for row 1, 3, and 4.
55 Updating fewer rows of bits speeds up update operations, while not losing any information about the area marked as occupied.
56 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.
57
58 Checking for overlap or marking an integer entry as occupied can be done in a constant number of bitwise-operations.
59 These operations have constant runtime, regardless of the size of the integer.
60 Our implementation uses the largest available integer size, to process many bits in parallel.
61
62 To record the areas of the marks for the labels to avoid, we rasterize all the marks in $M$ onto
63 the bitmap.
64 Every pixel that is not fully transparent is considered occupied and its corresponding bit in the bitmap set to one.
65 The number of bits used to represent marks is bounded by the size of the chart.
66 Thus, the runtime for rasterization linearly depends on the chart resolution and number of the graphical marks.
67 After the rasterization, a labeling algorithm using the \emph{occupancy bitmap} can efficiently perform occupancy checks and updates.
68 The runtime for an occupancy check or an update only depends linearly on the size of the label,
69 regardless of the number and size of the marks that need to not overlap with labels.
```

Applications:

```
1 \section{Applications to Labeling Algorithms}
2
3 In this section, we apply fast overlap detection using the occupancy bitmap
4 to place labels in scatter plots, line charts, and cartographic maps.
5 The algorithm for placing labels is greedy, following the labeling steps mentioned in the related work section.
6 It first projects all the graphical marks to be avoided by the labels into an \emph{occupancy bitmap}.
7 It then looks through each of the data points in one pass.
8 For each data point, the algorithm iterates through the candidate positions of the label to be placed.
```

```
1 \section{Fast Overlap Detection for Labeling Charts}
2
3 In this section, we apply fast overlap detection using the occupancy bitmap
4 to place labels in scatter and connected scatter plots, line charts, and maps.
5 The algorithm for placing labels is greedy, following the labeling steps described in \autoref{sec:related}.
6 It first rasterizes all marks onto an \emph{occupancy bitmap}.
7 It then places all labels in one pass.
8 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 graphical mark in the occupancy bitmap (skipping the rest of the candidate locations).

To place the label, it marks the rectangular bounding box of the label as an occupied area in the occupancy bitmap (\autoref{fig:bitmask}).
After placing the label, it continues with the next data point.
The algorithm to add labels in the three chart types mentioned only differs in

(1) the graphical marks to be avoided by labels and (2) the candidate positions for labels.

In scatter and connected scatter plots, candidate positions of a label are around the graphical point mark it represents.
For a scatter plot, the marks to be avoided by labels include the point marks that represent data points in the plot.
For a connected scatter plot, the marks to be avoided by labels include the point marks that represent data points in the plot and the line marks that connect the points (\autoref{fig:connected_scatter}).

In line charts, each line is composed of a series of data points and a line that connects them all as a single path.
Line charts are similar to connected scatter plots, but often one label represents a whole line instead of a data point.
Therefore, the labeling algorithm places one label per line.

Candidate positions of a label for a line are around each data point that represents the line.

The marks to be avoided by labels include the line marks.

A cartographic map (\autoref{fig:eval_airport_vis}) contains geographical locations that need to be annotated by labels and geographic properties that need to be avoided by the labels.
Candidate positions of a label are (similar to scatterplots) around the locations to annotate.

The marks to be avoided by labels include the point marks that represent locations and marks that represent geographical features (\eg country outlines and paths).

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 (\autoref{fig:bitmask}).

The algorithm to add labels in the 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~\cite{imhof1975positioning} 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 (\autoref{fig:connected_scatter}).

In a line chart, each group of 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.

As shown in \autoref{fig:eval_airport_vis}, a map can contain points that represent locations, which need to be labelled, and paths that represent geographical features (\eg 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.

Evaluation:

\section{Evaluation}

To evaluate our labeling algorithm using \emph{occupancy bitmap}, we compare our algorithm to the Particle-Based Labeling algorithm from Luboschik~\ea \cite{luboschik:particle}, the current state-of-the-art fast labeling algorithm.
The criteria that we use to compare are (1) runtime and (2) visual quality of the label placements.
We use the number of labels placed as an indication for visual quality because more labels give more information to readers.
To facilitate the comparison, we implemented both algorithms as transforms in Vega~\cite{satyanarayan:vega}.

The task that we use to benchmark our algorithm is to label a map that shows airports in the US
and travel routes between the Seattle-Tacoma airport (Sea-Tac) and other airports\footnote{This map is originally from the Vega-Lite example gallery at \url{https://vega.github.io/vega-lite/examples/geo_rule.html}.}, as shown in \autoref{fig:eval_airport_vis}.
The dataset contains 3320 airports and 56 routes from Sea-Tac.

\section{Evaluation}

To evaluate our labeling algorithm using \emph{occupancy bitmap}, we compare it to Particle-Based Labeling \cite{luboschik:particle}, the current state-of-the-art fast labeling algorithm.

To perform this comparison, we implemented both algorithms as transforms in Vega~\cite{satyanarayan:vega}
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\footnote{This map is originally from the Vega-Lite example gallery at \url{https://vega.github.io/vega-lite/examples/geo_rule.html}.}, as shown in \autoref{fig:eval_airport_vis}.
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 states in the US in light gray.
In this benchmark task, 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 positioned before running the algorithm.
They act as obstacles for the teal labels to avoid.

To account for higher resolution displays, we test the algorithm with chart widths from \$1000\$ up to \$8000\$.
The benchmark uses the image-based approach to project the graphical marks to be avoided by labels.
We choose to use the image-based over the vector-based approach for two reasons.
First, the image projection is easy to implement with existing image libraries.
The vector-based approach needs to be re-implemented for each mark type.
Second, the image-based approach is parameter-free.
In contrast, the vector based approach requires adjusting the sampling rate of particles to balance the projection's accuracy and performance when detecting overlaps.

From Luboschik \ea \cite{luboschik:particle}, we notice that there are two issues when projecting graphical marks.
First, a particle that represents an occupied pixel is placed at the center of the pixel.
This placement of particles allows labels to up to half-way overlap with a pixel that is supposed to be occupied (\autoref{fig:eval_airport_vis}).
Second, the algorithm projects every occupied pixel into a particle, which is unnecessarily too many.
The amount 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.
We addressed the first issue, a correctness issue, by placing particles at the four corners of an occupied pixel.
Since a label is a zero-degree-angled rectangle, it cannot overlap with an occupied pixel without overlapping with a particle at one of its corners first.
The second is a runtime issue that we addressed by omitting some particles that are too close to an existing particle.
Projection in our improved algorithm has two parts.
First, it projects all particles along the outlines of graphical marks.
Second, it projects particles inside graphical marks every other \$H_{min}\$ pixels vertically and \$W_{min}\$ horizontally pixels.
\$H_{min}\$ is the height of the label with the shortest height.
\$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.

\subsection{Performance}

For each parameter (labeling algorithm and chart's width), we run the task 20 times and take their median value.

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\cite{luboschik:particle} 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 \autoref{fig:eval_airport_vis}D.
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.
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 rasterize 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.

\subsection{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.

49	The result in \autoref{fig:eval_airport_performance} shows that	48	\autoref{fig:eval_airport_performance} shows that
50	the improved Particle-Based Labeling algorithm is faster than the original one as the chart size increases.	49	the improved Particle-Based Labeling algorithm is faster than the original one as the chart size increases.
51	Our bitmap-based algorithm performs significantly better than both the original and the improved Particle-Based Labeling algorithm.	50	Our Bitmap-Based algorithm performs significantly better than both the original and Improved Particle-Based Labeling algorithms,
52	Our algorithm takes at least 22\% less time to run, and the improvement tends to be higher as the chart size increases.	51	taking at least 22\% less time to run across the chart sizes.
		52	The improvement also generally increases as the chart size increases.
53		53	
54	\subsection{Visual Quality}	54	\subsection{Number of Labels Placed}
55		55	
56	The original Particle-Based Labeling algorithm places significantly more labels than our improved version.	56	As we discussed earlier, the original Particle-Based Labeling may allow a label
57	This means that its projection process of marks to be avoided by labels largely contributes to the increase of placed labels.	57	to overlap with a mark by a half pixel, thus it places significantly more labels than
58	Therefore, these excess labels overlap with one of the graphical marks in the chart as shown in \autoref{fig:eval_airport_vis} (D).	58	Improved Particle-Based Labeling and Bitmap-Based Labeling.
59	The improved version can catch these overlaps because its particles placed at each pixel's corners cover the whole pixel.		
60	The original version cannot catch these overlaps because its placement of a particle at each pixel's center does not cover the whole pixel.		
61		59	
62	Comparing our algorithm to the improved Particle-Based Labeling algorithm, our algorithm placed 0.8\% fewer labels in the chart with a width of 8000, and 3.2\% fewer labels at a width of 1000.	60	To avoid the effect of this correctness issue, we focus on the comparison of Bitmap-Based Labeling
		61	with Improved Particle-Based Labeling.
		62	Bitmap-Based Labeling placed 0.8\% fewer labels for charts with 8000 pixel width
		63	and 3.2\% fewer labels for charts with 1000 pixel width.
63	Our algorithm is able to place a similar number of labels as the Particle-Based Labeling algorithm if we only count labels that do not overlap with any marks.	64	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.
64		65	

Conclusion and Future Work:

1	\section{Conclusion}	1	\section{Conclusion and Future Work}
2		2	
3	We have developed an \emph{occupancy bitmap} that efficiently detects overlappings of labels to graphical marks and other labels in a chart.	3	We present \emph{occupancy bitmap}, a data structure that can efficiently detect overlaps between a label and other marks or labels in a chart.
4	The bitmap is used in a greedy-based labeling algorithm to quickly label charts.		
5	We show that this labeling algorithm can label scatter/connected scatter plots, line charts, cartographic maps, and their combinations.	4	We apply this bitmap in a greedy label placement algorithm and apply it to label scatter plots, connected scatter plots,
		5	line charts, and maps.
6	We compare our algorithm with the state-of-the-art Particle-Based Labeling algorithm.	6	We compare this Bitmap-Based Labeling algorithm with the state-of-the-art Particle-Based Labeling algorithm,
7	Our algorithm is significantly faster and able to place similar numbers of labels in charts.	7	showing that the Bitmap-Based algorithm is significantly faster and can place similar numbers of labels in charts.
8		8	
9	\section{Future Work}	9	For future work, we plan to apply occupancy bitmaps to label other charts that
		10	need a different placement strategy other than the 8-position model used in this paper.
		11	For example, stacked area charts need a method to place a label inside each area shape.
10		12	
11	In future work, we would like to apply our \emph{occupancy bitmap} and our algorithm to label other types of charts.	13	For chart interactions like zooming or panning, a na{\"\i}ve greedy label placement algorithm may re-render label placements
12	For example, area charts are widely used but are not supported by our current labeling algorithm since they have different positioning of labels.	14	for every frame of animations, which can be too slow for large datasets.
13	Fortunately, our \emph{occupancy bitmap} can be used to detect overlapping of labels and areas.	15	We plan to explore better optimization to avoid re-rendering in every new frame, while providing smooth interactions.
14	Our bitmap is designed to be a module for labeling algorithms, so it can be swapped into other labeling algorithms implemented in the future that make use of it to label area charts.	16	
15		17	

Figures' captions:

Fig 1:	Fig 1:
(Left) the orange pixels are the projection of the connected scatter plot into the bitmap to mark those pixels as occupied.	(Left) We rasterize connected scatter plot onto the bitmap to mark occupied pixels, shown in orange.
(Middle) When placing a label for a data point, the eight rectangles are the eight candidate positions for the label to be placed.	(Middle) We use the 8-position model~\cite{ imhof1975positioning} to generate candidate positions for label placements.
The green positions are available, while the red ones are not.	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.	(Right) After placing the label ``1975'', the pixels under the label need to be marked as occupied.
Fig 2:	Fig 2:
The black indices indicate the x/y coordinate of pixels in the chart.	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.	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.	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 blue circles are marking occupied pixels.
The yellow box is the area to lookup or update.	The yellow box is the area to lookup or update.
Fig 3:	Fig 3:
(Left) Labeled connected scatter plot.	(Left) Labeled connected scatter plot.
(Right) A snapshot of the bitmap when labeling the connected scatter plot. In this figure, a greedy labeling algorithm already placed labels in the left half of the chart.	(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.
Fig 4:	Fig 4:
The visualizations from an evaluation to compare (A) our algorithm to (B) the Particle-Based Labeling algorithm from Luboschik \ea \cite{luboschik:particle}.	The labeling results from (A) our Bitmap-Based Labeling and (B) Particle-Based Labeling by Luboschik \ea \cite{luboschik:particle}.
(C) is the visual difference of (A) and (B).	(C) shows the visual difference between (A) and (B).
(D) One of the labels placed by the Particle-Based Labeling algorithm overlaps with the line that the label is supposed to avoid as indicated with the red cross.	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 proposed algorithms address these issues.	Our Improved Particle-Based Labeling algorithm address this issue.
Fig 5:	Fig 5:
Results of evaluations comparing our algorithm, the original Particle-Based Labeling algorithm, and our improved Particle-Based Labeling algorithm.	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 evaluations compare runtime and number of label placed.	
Gray bands show the difference for each comparison.	The gray bands show the differences between conditions.