A D3 plug-in for automatic label placement using simulated annealing

Evan Wang

Abstract—Although labeling graphical features can help viewers quickly grasp complex nuances of the data, it is a very time-consuming process. For the majority of point-feature label placement problems, the rules are relatively straightforward. Therefore, this is a problem very much suited for automated labeling algorithms. Automatic label placement is widely used by map generators, but surprisingly, there is little evidence that individuals use it. One reason might be that for many advanced plotting tools favored by scientists and engineers, there is often no sophisticated built-in labeling function. A possible solution is to create a plug-in (also called extension or add-on) for an existing program that performs label placement. Such plug-ins do exist for some plotting programs, with varying sophistication and ease of use. In this work, I plan to write build an automatic labeling plug-in for the popular JavaScript visualization library D3 that implements simulated annealing and easily incorporates into existing D3 code, with syntax mirroring other D3 layouts.

Index Terms—automatic label placement, simulated annealing, D3, plug-in

1 Introduction

Labeling features or data points, more commonly known as *point-feature label placement*, is an integral part of a visualization because it helps viewers quickly see what the data represents and points out nuances that could otherwise be overlooked. However, manually labeling features is a very time-consuming task. According to Cook and Jones, cartographers place approximately 20-30 labels per hour [1].

Given the time required to manually place labels, automatic label placement is used widely by map generators such as LineDrive [2]. However, there is little published or empirical evidence that indicates individuals frequently use automatic labeling. This presents a conundrum. Why would so many individuals manually label graphs and charts even though doing so requires a substantial time commitment? Not only that, an optimization problem such as this is perfectly suited for one of many standard search/optimization algorithms available. One reason is that for advanced plotting tools (R, D3, Matlab, Mathematica, and Matplotlib) favored by scientists and engineers, there is often no sophisticated built-in automatic label placement function. In order to label a scatter plot, one often has to resort to tweaking labels manually. A possible solution to the problem is to create a plug-in (sometimes called extension or add-on) for an existing program that performs the label placement. Plug-ins do exist for some of the aforementioned plotting tools.

In particular for D3, an increasingly popular Javascript visualization library, there are presently label placement implementations available that make use of the D3 force layout [16, 17]. However, the force layout (based on force-directed algorithms) is an algorithm for drawing graphs, specifically undirected graphs [3]. And although it can be effectively adapted for labeling, it is not built for such a task and therefore many of the nuances of good labeling practices cannot be implemented. For my project I have implemented a D3 plug-in for automatic label placement using simulated annealing that incorporates good label placement practices and is easy-to-use.

2 Related work

2.1 Label placement rules

The layout of a labeling problem is shown in Fig. 1. Each *label* corresponds to an *anchor point*. A *leader line* may be used to help with the correspondence between the *label* and *anchor point* [14]. None of the elements may cross the *graph boundary*. The general rules for label placement have been carefully studied by many, most notably described by Imhof [4, 5] and more quantitatively formulated by Yeoli [6]. The basic rules are:

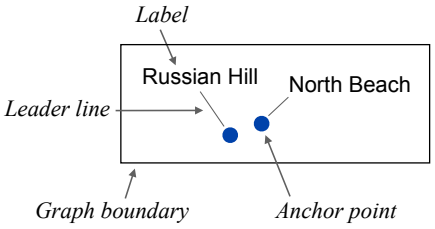

Fig. 1. **Anatomy of a label problem.** Each *label* corresponds to an *anchor point*. A *leader line* may be used to help with the correspondence between the *label* and *anchor point*. None of the elements may cross the *graph boundary*.

- Spatial overlap: Labels should not overlap with each other or other point features.
- Unambiguity: Labels should be clearly and unambiguously identified with its corresponding graphical feature(s).
- Legibility: Labels should be easily readable.

Comparisons of good and bad labeling practices are showin in Fig. 2. Fig. 2 (a) is an example of a bad label configuration because it does not follow Imhof and Yeolis rules. There are several label-label overlaps, label-anchor overlaps, and in general a sense of randomness in how the labels are placed. This results in ambiguity in label-point correspondence and for the viewers, a longer time or even inability to process the data. In contrast, Fig. 2 (b) shows a much better label configuration because there are no overlaps of any kind and there exists a much greater readability due to the ordered nature of the label placements.

In addition to these rules, Imhof described a set of more specific stylistic rules pertaining to specific label positions [4]:

- Label position on the right is preferred over left.
- Label position on top is preferred over below.
- Label position closer to the corresponding point feature is preferred.

A visual representation of Imhof's stylistic rules are shown in Fig. 3.

• *Author email: evan.wang@berkeley.edu*

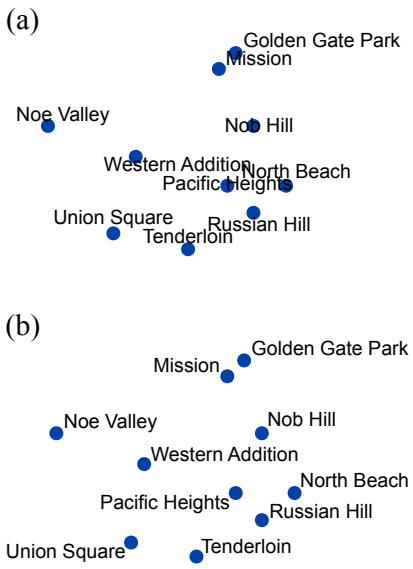

Fig. 2. **Good and bad label placements.** (a) *Bad label placement*: Some labels overlap with each other and/or with anchor points. In addition, there are no conventions for preferred placement positions. This creates ambiguity in terms of which label corresponds to which data point. (b) *Good label placement*: There are no overlaps, labels tend to be situated in a corner of the data point, and consequently, no ambiguity for label-data correspondence.

2.2 Search space

So far, our description of the problem has been qualitative (i.e. description based rules such as avoid overlaps, place labels in more preferred positions). For an optimization problem such as this, we have to first define the search space [7]. In a labeling problem, the search space consists of the collection of label positions. Each configuration of label positions, or *state* (v), can be written as

$$v = \{r_1, r_2, r_3, \ldots r_N\}, \tag{1}$$

where $r = (r_x, r_y)$ is used as a shorthand to denote the position of each label. Additionally, each individual component of every label position $\mathbf{r}$ must satisfy the constraints of the graph boundaries,

$$\begin{aligned} r_x &> L_x^{min} \\ r_x &< L_x^{max} \\ r_y &> L_y^{min} \\ r_y &< L_y^{max}. \end{aligned} \tag{2}$$

All label configurations that satisfy the boundary conditions are permissible. The search space contains $2N$ degrees of freedom, where N is the total number of labels. It is important to note that we have allowed neither the fontsize of the labels nor the orientation (tilt) to be a free parameter. Adding these elements as free parameters would increase the dimensionality of the problem (an additional N degrees of freedom for each added parameter), but is otherwise entirely possible and straightforward. However, size and orientation are mostly used in cartographic applications were the various geographical features necessitates their usage and utility. For label point features in graphs, features such as size and orientation are usually fixed and is the reason for their exclusion.

2.3 Energy function

Of course, not all points in the search space are equally preferable. Label configurations that result in less overlap and in general adhere to label placement rules are more preferred. In Fig. 2, both (a) and (b)

Fig. 3. **Preferred label positions.** The relative preference of the positions (according to Imhof [4]) are indicated according to the color saturation of the label. Darker labels represent more desirable positions.

are permissible states or configurations in the search space. However, Fig. 2 (a) is the more preferred label configuration because there are less overlaps and greater readability. In order to distinguish between the quality of different configurations in our search space, we need construct a function which takes as input a label configuration and outputs a score indicating the quality of the placements. In a labeling problem, the inputs are themselves functions of various parameters such as the amount of overlaps, distances between labels and their corresponding anchor points, and various stylistic preferences. This function, often an energy (also called cost or objective) function, is what we need to optimize. Based on Imhof and Yeoli's rules [4, 5, 6], our energy function for each individual label i depends on

$$E_i = f(A_{ll}, A_{lp}, L_{lp}, \theta), \tag{3}$$

where A_{ll} is the area of label-label overlaps, A_{lp} is the area of label-point overlaps, L_{lp} is the distance between the label and the point, and θ is the orientation of the label relative to the point. The form of our energy function for label i for a system with N total labels is then

$$E_i = \sum_{j \neq i}^{N} A_{ll} W_{ll} + \sum_{j \neq i}^{N} A_{lp} W_{lp} + L_{lp} W_{lp} + \theta W_\theta, \tag{4}$$

where the W terms are weight constants signifying the relative importance of each rule. For each label i, we sum over all other labels and points in order to compute the total area overlap. The greater the overlap, the higher the resulting energy. To obtain the total energy, we simply sum over each label, or index i

$$E_{tot} = \sum_{i}^{N} E_i. \tag{5}$$

These are only an example of an energy function for a labeling problem. Different terms can be inserted or removed based on individual preferences.

2.4 Overview of algorithms

The labeling problem is essentially an optimization problem on a complex and high-dimensional energy landscape. Therefore, many possible classes of algorithms can be applied. A shortlist of the more well-known algorithms include random placement, exhaustive search algorithms, greedy algorithms, local search algorithms, stochastic search algorithms, genetic algorithms, and mathematical programming [7]. In general, these algorithms may be categorized into local search methods and global optimization methods. For the sake of brevity, I will give a more detailed overview for the more widely used algorithms.

2.4.1 Greedy algorithm

In general, a greedy algorithm makes a locally optimal choice at every step without backtracking. As applied to label placement, the algorithm is the following. With respect to labeling, this means that labels are placed serially, and each label is placed to minimize the current energy, given the positions of the other labels. Algorithmically,

it is implemented as follows.

while *all points are not labeled* **do**
  select an unlabeled point;
  place label according to preset rules;
  if *no free position* **then**
    leave out the label;
    or
    label point even with overlap;
    or
    ask for feedback from human;
  end
end

Algorithm 1: Greedy algorithm

Greedy algorithms are best suited for problem in which "thinking ahead has little benefit. Physically, since we are freezing the configuration of all other labels, the energy landscape is always two-dimensional. With each label, we are finding the global minima of that two-dimensional and changing landscape. We never consider the entire $2N$-dimensional landscape. This is the so-called myopic behavior associated with greedy algorithms [15]. However, the labeling problem is best solved by finding a minimum on the full energy landscape, and not simply minima on two-dimensional slices of the landscape. Thus while generally very fast, the greedy algorithm frequently generates label configurations that could be vastly improved [7].

2.4.2 Gradient descent

An gradient descent implementation of automatic label placement is generally considered better than using a greedy algorithm because we now can fine tune a particular configuration by local label movements. The algorithm is as follows [8].

while *convergence not reached* **do**
  randomly make several perturbations to the configuration;
  calculate decrease in cost function ΔE;
  accept the perturbation with the most negative ΔE;
end

Algorithm 2: Gradient descent

Although it generally performs better than greedy algorithms, gradient descent is still a local optimization technique. This means that it tends to get stuck in minima close to the starting configuration and cannot escape into a deeper local minimum that is further away. Therefore, the initial placement of label positions is extremely important because only the local landscape is accessible. This idea is illustrated in Fig. 4. In the one-dimensional slice of the energy surface, the green circle illustrates the starting configuration. Local optimization results in configurations that at the bottom of the nearest local minimum following the steepest gradient. However, a nearby deeper local minimum is avoided. So often, the problem reduces to a very careful placement of initial label positions, which is more or less the problem we are trying to solve in the first place. Our goal is to spend minimum effort on the initial configuration and let the algorithm find a good label configuration. For these reasons, using gradient descent algorithms often result in less optimum label configurations for anything except the simplest of label placement problems.

2.4.3 Simulated annealing

Given the high dimensional nature of a problem such as label placement, a global search algorithm is preferred over a local one because global algorithms can avoid being trapped in local minima. Simulated annealing is one such algorithm [9, 10]. It is not an exhaustive global search algorithm. For any sizable system (i.e. relatively high number of labels), the search space is enormous ($2N$ degrees of freedom) and sampling all the configurations is neither feasible nor a wise use of computational resources. Therefore, we want to focus on label configurations that are low in energy. This is exactly what is done in simulated annealing. We make random changes to the label positions, and we accept these changes with a probability proportional to their Boltzmann weight. The probability by which we accept these moves is modulated by a temperature term (T). Temperature can be viewed

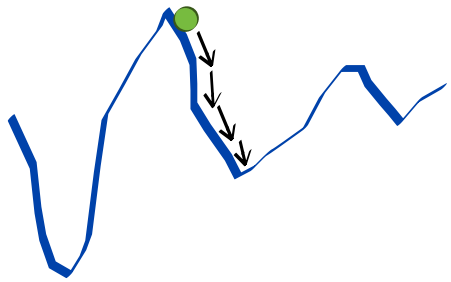

Fig. 4. **Local search algorithms.** Local search algorithms such as gradient descent find nearby local minimum. In this one-dimensional slice of the energy landscape (the unlabeled vertical axis is energy or cost), the green circle illustrates the starting configuration. Local optimization results configurations located at the bottom of the nearest local minimum following the steepest gradient. However, a nearby deeper local minimum is avoided.

as the amount of energy the system possesses in order to jump over energy barriers. This is visually represented in Fig. 5. Higher the temperature, the more likely we are to accepting configurations that have a high energy. Lower the temperature, the less likely we are to accepting these moves. In the case of $T = 0$, we go to the bottom of the nearest energy well. We start at a higher temperature in order to explore the high dimensional energy surface. However, we slowly decrease the temperature according to a schedule, also known as annealing the system. By doing so, we are focusing our search on deeper minima, and avoiding the higher energy configurations. The hope is that as the temperature gets very low, we have found either the global minimum or a very low energy local minimum.

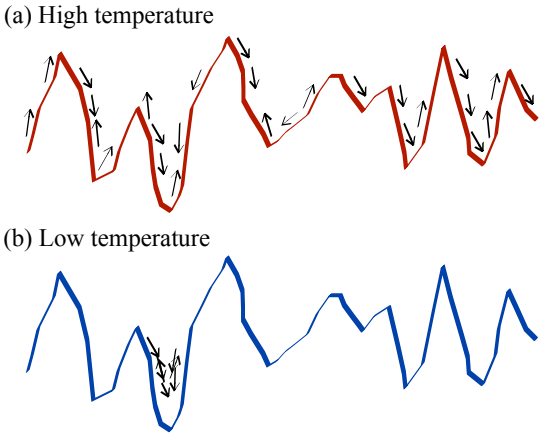

Fig. 5. **Simulated annealing at high and low temperatures.** The unlabeled vertical axis is energy. (a) At higher temperatures, we are not likely to accept configurations that have a high energy. We start at a higher temperature term in order to explore the high dimensional energy surface. (b) At lower temperatures, we are less likely to accept high energy configurations. In the case of $T = 0$, we go to the bottom of the nearest energy well. As the simulation progresses, we slowly decrease the temperature, also known as annealing the system. By doing so, we are focusing our search on deeper minima, and avoiding the higher energy configurations. The hope is that as the temperature gets very low, we have reached either the global minimum or a very low energy local minimum.

The protocol by which we lower the temperature is very important to the resulting configuration. If the temperature is lowered very fast, often called "quenching", the system can potentially get trapped in a nearby local minima, resulting in a less than optimum final config-

uration. It is recommended that the system cools slowly rather than rapidly [12], the reason being that we always want to keep the at or close to the equilibrium at all temperatures. The most often used cooling schedules have been either linear

$$T(n) = T_0 \alpha^n, \tag{6}$$

or exponential

$$T(n) = T_0 - \beta n, \tag{7}$$

where T_0 is the initial temperature, α, β are constants, and n is the number of steps taken [13].

Simulated annealing is often used with the Metropolis acceptance criteria [11],

$$P_{acc} = \begin{cases} e^{-\Delta E/k_B T} & \text{if} \quad \Delta E > 0 \\ 1 & \text{if} \quad \Delta E \leq 0 \end{cases} \tag{8}$$

where P_{acc} is the probability of accepting a new label configuration, ΔE is the change in energy of the system going from one configuration to another, k_B is Boltzmann's constant, and T is the system temperature. In our case, we will denote temperature in reduced units of k_B for simplicity. The algorithm for simulated annealing is as follows.

while *convergence not reached* **do**
  attempt a move $v \rightarrow v'$ by translating or rotating labels;
  evaluate change in energy $\Delta E = E_{v'} - E_v$;
  if $\Delta E < 0$ **then**
    accept new configuration;
  else
    generate random number ξ between 0 and 1;
    if $\xi < e^{-\Delta E/T}$ **then**
      accept new configuration;
    else
      reject configuration;
    end
  end
  decrease T according to schedule;
end

Algorithm 3: Simulated annealing

3 Methods

3.1 Choice of algorithm

For any nontrivial labeling problem, the energy landscape will be very high-dimensional ($2N$ degrees of freedom) and rough. In order to find a good label placement configuration, we have to avoid getting trapped in local minima. Therefore, global search algorithms are more suited than local search algorithms. Out of the many global search algorithms available, simulated annealing is one of the more favored algorithms because of its simplicity, flexibility, and intuitive physical basis. It is also important for me to choose an algorithm that many people are familiar with, so that they can build on my work and add additional functionality to suit their specific labeling preferences. For these reasons I have implemented the automatic label placement plugin using simulated annealing.

3.2 Incorporation within D3

My implementation is more generally a D3 simulated annealing graphical layout, adapted for the problem of label placement. The energy function in the layout corresponds to a particular set of graphical rules that are described in the sections below. However, the users have the option to define custom energy functions in order to suit individual labeling preferences (details in Sec. 5.2). Furthermore, with additional changes, the simulated annealing tools within the plug-in can be adapted for any optimization problem, such as drawing a node-link diagram.

3.3 Energy function

The specific form of the energy function dictate the landscape of the search space. Therefore, a carefully constructed energy function that incorporates good labeling practices is crucial to the resulting labeling configuration. The energy function in the current implementation includes terms for rules that I believe will appeal to a wide audience of students, academics, and scientists in the industry setting. A tabular summary of all the energy terms is shown in Table 1. In the following subsections I will describe in more detail the various terms in the energy function and their corresponding labeling rules.

3.3.1 Label-label and label-feature overlap

Perhaps the most important rule in label placement is to avoid label-label and label-anchor overlaps. Overlaps not only hinder our ability to decipher text but also compromise the overall aesthetics of the plot, giving it an unprofessional look. We use the following energy terms to account for overlaps

$$E^{overlap} = \sum_i^N \sum_{j \neq i}^N W^{lab-lab} A_{ij}^{lab-lab} + \sum_i^N \sum_j^N W^{lab-anc} A_{ij}^{lab-anc}, \tag{9}$$

where the first double sum represents the label-label overlaps and the second double sum represents the label-anchor overlaps. A_{ij} is the area of the overlap between label i and label or anchor j, and W is the weight of the overlap energy penalties. In order to calculate overlaps, the labels as well as the anchor points have been approximated as rectangles. The overlap term in the energy function is simply the area of overlap between the rectangles. This is illustrated in Fig. 6.

Fig. 6. **Label-label and label-feature overlap.** In order to calculate overlaps, the labels and anchor points have been approximated as rectangles. The overlap term in the energy function is simply the area of overlap between the rectangles. Overlap areas for (a) label-label and (b) label-anchor are colored in red.

3.3.2 Distance between feature and corresponding label

Label positions closer to the corresponding point feature are preferred [4]. The further a label is positioned from its anchor point, the more difficult it is to detect correspondence. To account for this, we use an energy penalty that is linear in the distance. The contribution to energy corresponding to the label-feature distance is

$$E^{dist} = \sum_i^N W^{dist} d_i, \tag{10}$$

where d_i is the Euclidean distance between the label i and its corresponding anchor point, and W^{dist} is the weight of the energy penalty.

3.3.3 Intersection of leader lines

The intersection of leader lines can hinder our ability to following a line to the end point, delaying our comprehension of the data. The energy penalty is simply a linear function of the number of leader line intersections

$$E^{intersect} = \sum_i^N W^{intersect} I_i, \tag{11}$$

where I is a count of the total number of leader line intersections and $W^{intersect}$ is the weight of the energy penalty.

3.3.4 Label orientation

In addition to the more obvious rules of avoiding overlaps and line intersections, Imhof also described a set of stylistic rules, including the relative desirability of label positions. This is shown in Fig. 3. An energy term was constructed in order to mimic these stylistic rules

$$E^{orient} = \sum_{i}^{N} W^{orient} Q_i, \tag{12}$$

where $Q = 1, 2, 3, or\ 4$ is the quadrant preference index in Fig. 7 and W^{orient} is the weight of the energy penalty. A pictorial representation of this energy function is shown in Fig. 7.

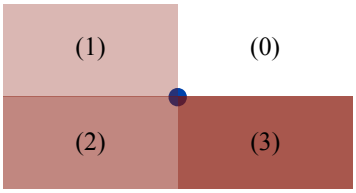

Fig. 7. **Orientation bias.** In Imhof's seminal work on label rules, he set forth the relative desirability of label positions. This is shown in Fig. 3. An energy term was constructed in order to mimic these stylistic rules. Red is used to indicate presence of an energy penalty. Darker color and higher number (quadrant preference index Q) represents higher energy penalty.

3.4 Monte Carlo moves

If we only use label translation moves, in theory, we are able to sample the entire configuration space of label positions. However, this will not ensure the most efficient sampling protocol, as measured by the time needed to find a sufficiently low energy minimum. This is because in general, it is more preferential for a label to be close to the corresponding data point. If a label is at an optimum distance away from its data point but in the wrong orientation, it would likely take many translation moves for the label to shift to the right orientation, because in general translation a label changes its distance from the data point. In this case, rotation moves will on average shift the label to the right orientation faster. Therefore, to ensure both an ergodic and efficient sampling of the conformational space, we use a combination of label translation and label rotation moves. These moves are shown in Fig. 8.

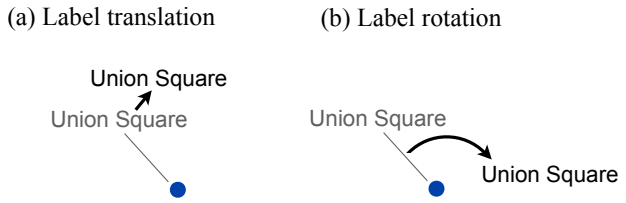

Fig. 8. **Monte Carlo moves.** To ensure both an ergodic and efficient sampling of the conformational space, we use a combination of (a) label translation and (b) label rotation moves.

3.5 Annealing schedule

As mentioned before, the choice of good annealing schedule is very important to the quality of the final configuration. For this problem I choose the popular linear cooling protocol [13]

$$T(n) = T_0 - \frac{T_0}{n_{tot}} n, \tag{13}$$

where T_0 is the initial temperature, n is the current number of Monte Carlo sweeps that have been implemented, and n_{tot} is the total number of Monte Carlo sweeps.

4 Results

4.1 Sample label configurations

Sample label configurations using the plugin are shown in Fig. 9. In test runs, labels are initialized (Fig. 9a) such that the bottom left corner of the label is placed at the center of the anchor point. The resulting configurations are relatively insensitive to any reasonable label initialization schemes (i.e. labels are close to the anchor point). For well-separated labels, the most preferred position is the upper right corner, without any overlaps. This minimum can be easily found using the implemented simulated annealing scheme (Fig. 9b). When two anchor points are close together, most likely the two corresponding labels cannot be both in their most preferred position (Fig. 9c). In this case, one of the labels (Node 43) is in the preferred position while the other label (Node 16) is rotated and translated slightly in the vertical direction. In a slightly different example (Fig. 9d), the algorithm finds a minima where the label for Node 7 is in its preferred position and the label for Node 5 is rotated to the bottom with respect its anchor.

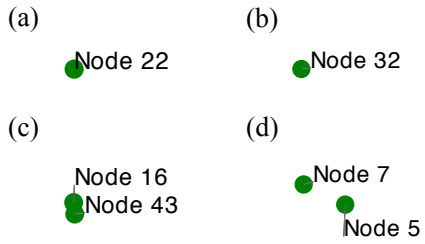

Fig. 9. **Sample configurations.** (a) Initialization: Labels are initialized such that the bottom left corner of the label is placed at the center of the anchor point. The final resulting configurations are relatively insensitive to reasonable label initialization schemes (i.e. labels are close to the anchor point). (b) Global minima for individual, well-separated labels: For well-separated labels, the most preferred position is the upper right corner, without any overlaps. This minimum can be easily found using the implemented simulated annealing scheme. (c) Labels close together: When two anchor points are close together, most likely the two corresponding labels cannot be both in their most preferred position. In this case, one of the labels (Node 43) is in the preferred position while the other label (Node 16) is rotated and translated slightly in the vertical direction. (d) Labels close together: In this case, the algorithm finds a minima where label for Node 7 is in its preferred position and the label for Node 5 is rotated to the bottom with respect its anchor.

4.2 Benchmark results

To further evaluate the effectiveness of the plugin, benchmark simulations have been performed for different number of labels (N = 25, 50,, 75, 100, 125, 150). They are summarized in Fig. 2. Results are evaluated based on run times, number of label-label overlaps, number of label-anchor overlaps, and number of leader line intersections. These tests were run on an Intel Core i5 2.7 GHz machine with 4 GB of RAM. For each N (number of labels), 100 independent trials were performed, each for 1000 Monte Carlo sweeps[1] The results shown in Fig. 2 are averaged from the 100 trials. For each N, the final label configuration of one run was saved and the snapshots are shown in Fig. 10.

4.2.1 Timing

Overall, the energy function is dominated by $O(n^2)$ terms, and so we expect to see a quadratic dependence on the number of labels. This

[1] One Monte Carlo sweep means that on average, each label is translated or rotated once. To obtain the actual number of Monte Carlo steps taken, multiply the number of sweeps by the number of labels N.

Energy term	Description	Mathematical form	Complexity
$E^{label-label\ overlap}$	Penalizes label-label overlaps	$\sum_i^N \sum_{j \neq i}^N W^{label-label} A_{ij}^{label-label}$	$O(n^2)$
$E^{label-anchor\ overlap}$	Penalizes label-anchor overlaps	$\sum_i^N \sum_j^N W^{label-anchor} A_{ij}^{label-anchor}$	$O(n^2)$
E^{dist}	Penalizes labels far from the corresponding anchor	$\sum_i^N W^{dist} d_i$	$O(n)$
$E^{intersect}$	Penalizes leader line intersection	$\sum_i^N W^{intersect} I_i$	$O(n)$
E^{orient}	Penalizes poorly oriented labels	$\sum_i^N W^{orient} Q_i$	$O(n)$

Table 1. **Individual terms in the energy function.** The terms in the energy function are based on label placement rules set forth by Imhof and Yeoli [4, 5, 6]. Each term is multiplied with its own weighting term, which accounts for the relative importance of each term.

is exactly what we observe in the benchmark timings in Fig. 2. For number of labels less than 75, the algorithm takes on the order of a second to run. This is the range typically expected for most users and suitable for most figures. Figures with many more data points are usually not ideal for labeling. For a figure with more than 75 labels, the time to complete the simulated annealing procedure begins to be more noticeable. In the largest case tested ($N = 150$), it takes on average 4.5 seconds to anneal the labels. This is still a small (even negligible) fraction of the time it takes to place labels manually.

4.2.2 Label-label overlaps

Label-label overlaps are arguably the worst labeling offense because it interferes the most with the purpose of label to clarify data points. In our benchmark tests, we see that below $N = 75$, there is on average less than 1 label-label overlap. It is important to note that most of these overlaps only involve a small portion of two labels, as opposed to two labels directly on top of each other. The number of overlaps increase to almost 2 when we increase the number of labels to $N = 100$. However, from Fig. 10 (d), we can see that given the space of the figure, this makes a relatively crowded result. For $N > 100$, labeling is not suggested. This can be seen in Fig. 10 (e) and (f). In many cases, overlaps can be avoided by allowing the labels to be far apart from the anchor (decreasing the weight of the distance penalty). However, visually it is often easier to see the label anchor correspondence with a (partial overlap, short distance) combination than with a (no overlap, long distance) combination. Therefore the weights were adjusted accordingly to favor the former configuration.

4.2.3 Leader line intersections

In the energy function, we penalize each leader-line intersection with an energy of $W_{intersect}$. From Fig. 2, we see that overall, there are very few intersections. Below, $N = 75$, the number of leader line intersections are essentially negligible. Even in the case of $N = 150$, there is on average $\sim$1 intersection, hardly noticeable for such a high number of labels. The small number of intersections is partly a result of an energy term that explicitly penalizes intersections. However, it is also because we have a separate term penalizing long distances between labels and anchor points. The shorter the leader line, the smaller is the probability of two lines intersecting.

4.2.4 Label-anchor overlaps

Compared to label-label overlaps, label-anchor overlaps are not as severe. As long as the labels and anchor points have different color, such overlaps often do not distract the viewers too much. This observation was used in adjusting label-anchor penalty weights. However, it is of course still good practice to avoid such overlaps. From Fig. 10, we see that compared to label-label overlaps, there are more label-anchor overlaps for each N. Below $N = 50$, there is on average approximately 1 (partial) overlap per configuration. That number increases to slightly more than 4 when $N = 75$. Past $N = 75$, the number of overlaps increases rapidly.

5 USAGE

5.1 Installation

To use the plug-in, first download *labeler.js*. Then include the plug-in within the relevant *.html* file with:

Labels	Time (sec.)	L-L overlap	Leader intersect.	L-A overlap
25	0.13	0.0	0.01	0.15
50	0.45	0.14	0.05	1.25
75	1.10	0.62	0.13	4.35
100	1.84	1.77	0.37	9.88
125	2.91	5.52	0.57	23.59
150	4.40	17.17	1.01	47.60

Table 2. **Benchmark results for various N.** Benchmark tests were run on an Intel Core i5 2.7 GHz machine with 4 GB of RAM. For each N (number of labels), 100 simulations of simulated annealing with 1000 Monte Carlo sweeps were run and the results were averaged. It is important to note that most of these overlaps only involve a small portion of the labels or anchors.

```
<script src="labeler.js"></script>
```

5.2 API

To automatically place labels, users declare a labeler (simulated annealing) layout, input label and anchor positions, the figure boundaries, and the number of Monte Carlo sweeps for simulated annealing. The general pattern is as follows:

```
var labels = d3.labeler()
               .label(labArray)
               .anchor(ancArray)
               .width(w)
               .height(h)
               .start(nsweeps);
```

The default settings are: $w = 1$, $h = 1$, and *nsweeps* = 1000. The default *labArray* and *ancArray* are empty arrays. Here we describe each term in more detail.

```
d3.labeler()
```

Start by declaring a labeling layout, the same as declaring any other D3 layout.

```
labeler.label(labArray)
```

Each label has the following attributes:

- x - the *x*-coordinate of the label
- y - the *y*-coordinate of the label
- width - the *width* of the label
- height - the *height* of the label
- name - the label text

Note that width and height (which are used to calculate the overlap areas) can be easily measured using the SVG `getBBox()` method.

```
labeler.anchor(ancArray)
```

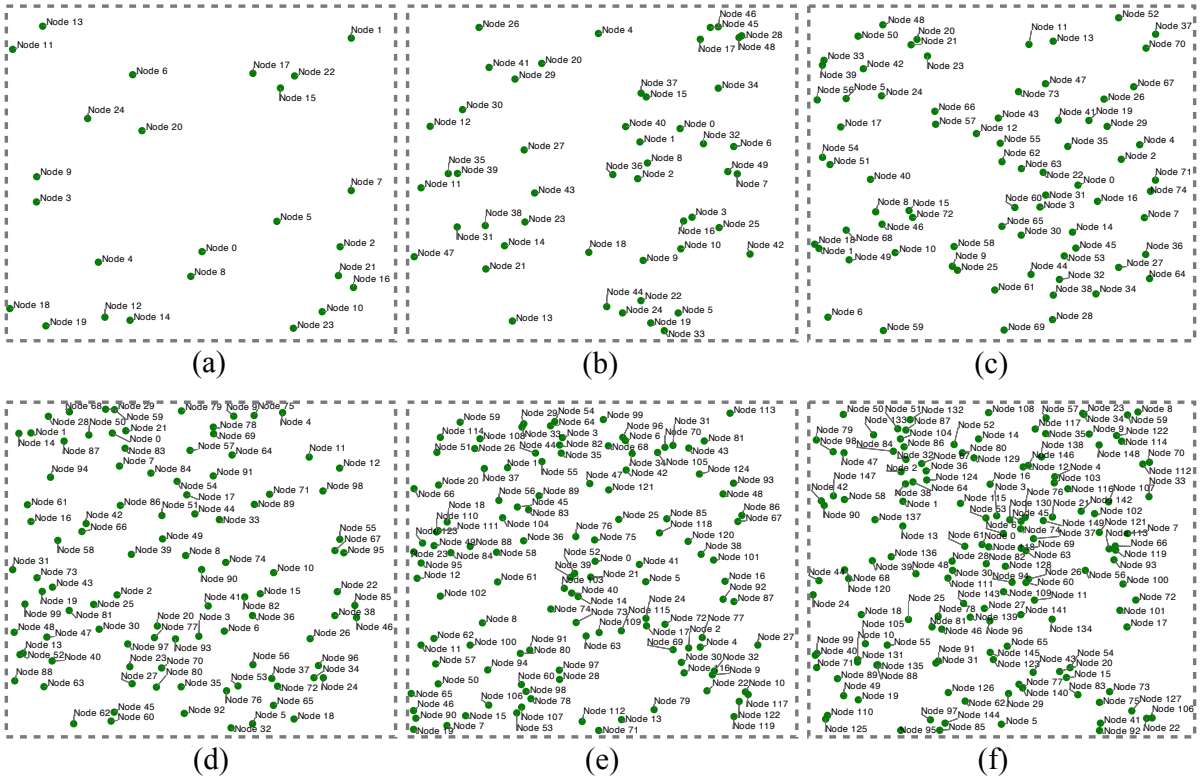

Fig. 10. **Snapshots of the final label configuration for various number of labels (N).** (a) $N = 25$ (b) $N = 50$ (c) $N = 75$ (d) $N = 100$ (e) $N = 125$ (f) $N = 150$.

Each anchor has the following attributes:

- x - the x-coordinate of the anchor
- y - the y-coordinate of the anchor
- r - the anchor radius

```
labeler.width(w)
labeler.height(h)
```

The width and height are used to set the boundary conditions so that labels do not go outside the width and height of the figure. More specifically, Monte Carlo moves in which the labels cross the boundaries are rejected. If they are not specified, both the width and height default to 1.

```
labeler.start(nsweeps)
```

Finally, we specify the number of Monte Carlo sweeps for the optimization and run the simulated annealing procedure. The default for textitnsweeps is 1000. Note that one Monte Carlo sweep means that on average, each label is translated or rotated once. To obtain the actual number of Monte Carlo steps taken, multiply the number of sweeps by the number of labels.

```
labeler.alt_energy(user_defined_energy)
```

This function is constructed for expert users. The quality of the configuration is closely related to the energy function. The default energy function includes general labeling preferences and is suggested for most users. However, a user may wish to define his or her own energy function to suit individual preferences.

```
energy = function(idx, labArray, ancArray) {
    var ener = 0;
    // insert interaction energies here
    return ener;
}
```

The newly constructed function must take as input an integer *idx*, an array of labels *labArray*, and an array of anchors *ancArray*. This function must also return an energy term that should correspond to the energy of a particular label, namely *labArray[idx]*. One may wish calculate an energy of interaction for *labArray[idx]* with all other labels and anchors.

```
labeler.alt_schedule(user_defined_schedule)
```

Similarly, an expert user may wish to include a custom cooling schedule used in the simulated annealing procedure. The default cooling schedule is linear.

```
schedule = function(currT, initT, nsweeps) {
    // insert user-defined schedule here
    return updatedT;
}
```

This function takes as input the current simulation temperature *currT*, the initial temperature *initT*, and the total number of sweeps *nsweeps* and returns the updated temperature *updatedT*. The user defined functions can be included as follows:

```
var labels = d3.labeler()
               .label(label_array)
               .anchor(anchor_array)
               .width(w)
               .height(h)
               .alt_energy(ener)
               .alt_schedule(schedule)
               .start(nsweeps);
```

6 Discussion and Conclusion

The intent of this work is to implement a good algorithm for automatic label placement in D3 and save designers time from manually

labeling graphs. Often, creating a figure takes many iterations of modifying the size of data points, axis ranges and scales, and the aspect ratio of the figure. Whenever each of these parameters change, the position of the labels could change as well and more time is spent by aligning the labels in accordance with the other changes. All these considerations are on the mind of the designer. However, if a designer is constantly worrying about having to shift and re-shift labels due to changes in the creative process, then in a way the creative process has been altered.

The main purpose of the figure is to showcase and tell a story using the data. This is where the bulk of the work should be. The less complicated and more streamlined we can make this process, the more time can be given to the presentation of the actual data. While labels are important, they are still ancillary to the data and should not overshadow the data in the creative process. This plug-in allows designers to focus on the data in the creative process and not worry about labeling because labels can be generated quickly and automatically.

7 Future Work

This plug-in is general purpose in the sense that it can be used to label any feature-based graphs (i.e. data points). To this end, I have implemented some of the more important labeling rules set forth by Imhof and Yeoli in order to appeal to a wide audience of people who use D3. While these rules prevent the more egregious errors, there are still many more aesthetic rules or features yet to be implemented. For example, while penalties for overlaps are included in the energy function, there is no term corresponding to the spacing between labels, which may be important for visual aesthetics. In addition, the labels are horizontally aligned and cannot be tilted. Although this is the case for the majority of labeling problems, there are graphs where a different orientation of the label might be useful (i.e. labeling the different functional dependence of a time-series graph). Implementing such additional features and rules can be an important direction for this work.

Currently, the default energy function supports labeling point-feature graphs. My hope is that ultimately, the project will have a variety of (built-in) energy functions in order to support many types of graphs (pie-chart, bar-graph, time-series). For now, I have implemented methods in which the individual users may insert their own energy functions in order to suit the particular needs of their graph.

Another important future direction is not in the labeling aspect of the work, but rather applying the simulated annealing layout for other problems. The automatic labeling placement plug-in I have created is essentially a simulated annealing layout with energy terms corresponding to good labeling practices. If swapped out for other energy terms, this plug-in can be easily adapted for other optimization problems, such as configurations for graphs. I hope that apart from its labeling capabilities, users will also use the simulated annealing framework that I have created for various other problems.

Acknowledgments

I would like to thank Prof. Maneesh Agrawala and members of the class for insightful comments and suggestions.

References

[1] A. Cook and C. Jones. "A Prolog rule-based system for cartographic name placement", *Computer Graphics Forum 9.* **1990**, 109-126.

[2] M. Agrawala and C. Stolte. "Rendering Effective Route Maps: Improving Usability Through Generalization", *Siggraph 2001.* **2001**, 241-250.

[3] "Handbook of Graph Drawing and Visualization", *Ed. R. Tamassia.* **2013**, CRC Press.

[4] E. Imhof. "Die Anordnung der Namen in der Karte", *International Yearbook of Cartography.* **1962**, 2:93-129.

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

[6] P. Yoeli. "The logic of automated map lettering", *The Cartographic Journal.* **1972**, 9:99-108.

[7] S. Shieber, J. Christensen, and J. Marks. "An empirical study of algorithms for point feature label placement", *Transactions on Graphics.* **1995**, 14(3).

[8] S. Hirsch. "An Algorithm for Automatic Name Placement Around Point Data", *Cartography and Geographic Information Science.* **1982**, 9:5-17.

[9] V. Cerny. "A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm", *Journal of Optimization Theory and Applications.* **1985**, 45:41-51.

[10] S. Kirkpatrick, C. Gelatt Jr., and M. Vecci. "Optimization by simulated annealing", *Science.* **1983**, 220:671-680.

[11] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. "Equation of state calculations by fast computing machines", *J. Chem. Phys.* **1953**, 21:1087-1092.

[12] H. Guo, M. Zuckermann, R. Harris, M. Grant. "A fast algorithm for simulated annealing", *Physica Scripta.* **1991**, 38:40-44.

[13] Y. Nourani and B. Andresen. "A comparison of simulated annealing cooling strategies", *J. Phys. A: Math. Gen.* **1998**, 31:8373-8385.

[14] I. Vollick, D. Vogel, M. Agrawala, and A. Hertzmann. "Specifying Label Layout Style by Example", *Proc. UIST* **2007**, 221-230.

[15] S. Dasgupta, H. Papadimitriou, and V. Vazirani. "Algorithms", **2006**, McGraw-Hill .

[16] http://bl.ocks.org/MoritzStefaner/1377729.

[17] http://bl.ocks.org/ZJONSSON/1691430.