An Efficient Algorithm for Scatter Chart Labeling

Sebastian Theophil
sebastian@theophil.net
Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin
Germany

Arno Schödl
aschoedl@think-cell.com
think-cell Software GmbH
Invalidenstr. 34
10115 Berlin
Germany

Abstract

This paper presents an efficient algorithm for a new variation of the *point feature labeling problem*. The goal is to position the largest number of point labels such that they do not intersect each other or their points. First we present an algorithm using a greedy algorithm with limited lookahead. We then present an algorithm that iteratively regroups labels, calling the first algorithm on each group, thereby identifying a close to optimal labeling order. The presented algorithm is being used in a commercial product to label charts, and our evaluation shows that it produces results far superior to those of other labeling algorithms.

Introduction

The problem of labeling points in the plane has been studied extensively. In its simplest form, the so-called point feature labeling problem (PFLP) consists of a set of points, each with a corresponding label. Each label can be placed at a discrete number of positions. Label-label intersections are not allowed. This problem was first described in the context of labeling geographic maps (Imhof 1975; Hirsch 1982). An alternative formulation called the sliding label model (van Kreveld, Strijk, & Wolff 1999) allows the labels to be placed at any position as long as it touches its point. Both instances have been shown to be NP-complete (Marks & Shieber 1991; Formann & Wagner 1991; Iturriaga & Lubiw 1997). Naturally, when more label positions are available, labeling results improve and more labels can be placed. In (van Kreveld, Strijk, & Wolff 1999) a 2-approximation algorithm is developed that has been extended in (Strijk & van Kreveld 1999) to place labels with varying height while respecting line-type geometric constraints. Very recently, a force-based labeling algorithm has been proposed in (Ebner, Klau, & Weiskircher 2003) showing excellent results when compared to the sliding label 2-approximation algorithm. It performed even better than simulated annealing, which performed best in the survey paper by Christensen et al. (Christensen, Marks, & Shieber 1995).

From a user's perspective, all these approaches suffer from a serious drawback. Even small dense problem instances as in Fig. 1 cannot be labeled satisfactorily because

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

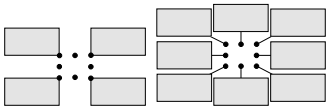

Figure 1: Simple problem instance were PFLP algorithms fail (left) and a possible resolution (right).

label positions are restricted to positions adjacent to each label's point. In most real-world cases however, the points will not be evenly distributed in the plane but they will concentrate in high-density clusters. We define a scatter chart as a set of points with rectangular labels, each of arbitrary dimensions. It is not only common among business charts but also among scientific illustrations. User expectation in such a chart is to show all labels the user has defined, if necessary with leader lines. We found that often, leader lines improve legibility in dense and hence confusing cases because they make the relation between a point and a label explicit. Thus, scatter chart labels can be placed at an arbitrary position but when a label is not adjacent to its point, both have to be connected by a leader line.

We therefore propose an algorithm for the scatter chart labeling problem (SCLP) which is an extension of the classic PFLP. Constraints disallow intersections between points, labels and leader lines. Additional constraints may include chart axes and their respective labels. Two criteria need to be optimized when searching for a scatter chart labeling: First, the number of placed labels must be maximized, and second, the total length of all leader lines must be minimized.

Since the algorithm is meant to be used in an interactive setting, to avoid user confusion, it must produce the same labeling in every run on the same problem. Thus, algorithms must be made deterministic if necessary. Of course, the interactive application setting also demands a fast algorithm which is able to produce results at interactive speed. In a real-world application it is also necessary to allow user control of label placement as argued by (do Nascimento & Eades 2003) and implemented by (Müller & Schödl 2005).

SCLP is shown to be NP-hard by reduction of discrete PFLP but the SCLP problem is much more complex than the discrete or sliding label PFLP problem. The search space of allowed label positions is much larger, and labeling de-

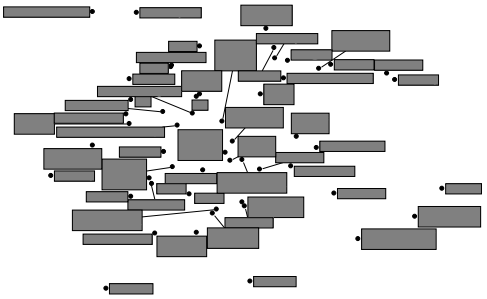

Figure 2: Example for a dense scatter chart and its labeling.

cisions may have large-scale effects, because especially in difficult cases, labels are no longer necessarily close to their points.

Solution Idea

In the sliding label model described by van Kreveld et al. the set of points of possible label centers form a rectangle around the point to be labeled. By clipping the rectangle's four line segments, the label can be constrained to positions where it does not intersect other labels or general line features. Our algorithm extends and generalizes this idea. In order to treat the complexity of the SCLP problem we discretize the angles at which labels can be placed around their point. A set of rays originating at each point define possible positions of the label center. Each ray carries information of free label position intervals along the ray, at which the label rectangle does not intersect other features. Whenever the algorithm fixes a label position, it updates the free intervals of rays belonging to other labels. We found 128 rays to result in aesthetically very pleasing solutions. At this resolution, the effect of only allowing discrete angles is not noticeable. We tried fewer than 128 rays but the results were less pleasing both by subjective human judgment as well as by objective measures when the number of rays became too small (< 32).

We solve the SCLP problem by using two largely independent algorithms. The first algorithm calculates the actual point labeling. When placing a label, the algorithm leaves as much space as possible for the remaining labels to be placed. Therefore, for each label placement, we need a measure for the remaining amount of valid label positions. We use the integral of a strictly decreasing function over the free intervals of a ray as a measure of this ray's free space. Consequently, the sum of all ray integrals belonging to one point is a measure for a label's free space. At each step of the algorithm, the decision which label to place is made based on maximizing the minimum remaining space for any other label. This limited lookahead is not sufficient to prevent obstruction of subsequently placed labels. The second algorithm improves the results of the first algorithm by finding a point partitioning into subsets and an order in which to pass these subsets to the first algorithm for labeling.

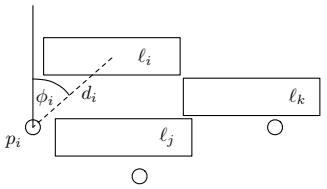

Figure 3: The scatter chart labeling problem.

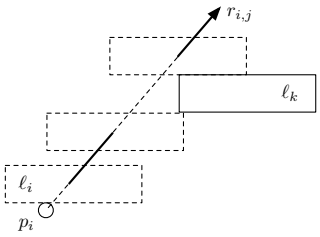

Figure 4: Representing a ray $r_{i,j}$ as an interval set $I_{i,j}$ containing the intervals of free label positions (solid lines).

The RayIntersection Algorithm

We first describe the algorithm used to solve the problem of finding a valid labeling for a given set of points. A set of available points $P \subseteq P_0$ is given, P_0 being the set of all points which need to be labeled, and each $p_i \in P$ has a label ℓ_i.

The position of each label ℓ_i can be described in polar coordinates (ϕ_i, d_i) (Fig. 3). The SCLP problem can be defined as the problem of assigning pairs (ϕ_i, d_i) to as many labels ℓ_i as possible, such that the total length of all leader lines $\sum_i d_i$ is minimized while avoiding intersections between labels, their leader lines and all points.

As already described, the main idea is to allow only a set of discrete values for the label's polar angle ϕ_i. Thus, each point $p_i \in P$ consists of a set of R rays where the j-th ray $r_{i,j}$ has an angle of $\phi_{i,j} = 2\pi/R \cdot j$, alternatively represented as the unit vector $\vec{v}_{i,j}$.

The rays define possible positions of the label center. On each ray $r_{i,j}$, the free label positions are a set of intervals $I_{i,j}$ as illustrated in Fig. 4. We define $d_{i,j} := \min\{a \mid [a,b) \in I_{i,j}\}$ as the free label position closest to point p_i on ray $r_{i,j}$. Initially, all intervals in which labels ℓ_i intersect other points are removed from $I_{i,j}$. Given a point p_k and a label ℓ_i on ray $r_{i,j}$ with the associated vector $\vec{v}_{i,j}$, the function LABELPOINTINTERSECTION computes the interval $[a,b)$ in which labels intersect the point: $\forall\, t \in [a,b)\ area(\ell_i \text{ translated by } \vec{v}_{i,j} \cdot t) \cap area'(p_k) \neq \emptyset$. The function sets $b = \infty$ if vector $\vec{v}_{i,j}$ itself intersects p_k, which means that the label cannot be moved beyond the obstructing point without causing its leader line to intersect this point. A similar function LABELRECTANGLEINTERSECTION is also defined which, given a label ℓ_k and a label ℓ_i on ray $r_{i,j}$ with the associated vector $\vec{v}_{i,j}$, computes the interval $[a,b)$ such that translating ℓ_i by $\vec{v}_{i,j} \cdot t$ with $t \in [a,b)$ makes ℓ_i and ℓ_k intersect. LABELRECTANGLEINTERSECTION includes the connecting line of ℓ_k in the computation

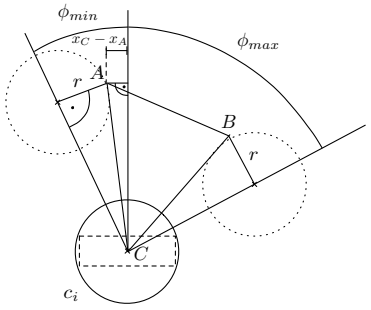

Figure 5: Computing the minimum and maximum angles $\phi_{\min}$ and $\phi_{\max}$ for intersecting c_i with a line. $\phi_{\min} = -\arcsin((x_C - x_A)/||\overline{AC}||) - \arcsin(r/||\overline{AC}||)$.

of $[a, b)$. This algorithmic framework would allow for different label-to-leader line attachments as long as the attachments are fixed a-priori, depending on leader line direction. Optimizing over leader line attachments would significantly add to the running time of the algorithm.

After initialization, the algorithm repeatedly chooses a point p_i and one of its rays $r_{i,j}$ for labeling. It places the label at the location $d_{i,j}$, the closest free location on ray $r_{i,j}$. As stated above, this choice should be made to minimize the impact on the remaining label positions. This criterion requires a measure for the available label space at each point. The remaining label positions at a point p_i are defined by the set of intervals $I_{i,j}$ for each ray $r_{i,j}$. To derive a measure for available space from these intervals, a strictly decreasing function $f(x)$ is chosen and the integral $\int_{I_{i,j}} f(x)\,dx$ over $I_{i,j}$ is calculated for each ray $r_{i,j}$. We chose $f(x) = e^{-ax}$ because it is easy to integrate in closed form. Since $f(x)$ is strictly decreasing, closer label positions are preferred over label positions farther from p_i. Summing up the integrals over all rays of point p_i gives a measure of the amount of remaining label positions called $\int p_i$.

This measure forms the basis for algorithm FINDBESTRAY shown on page 4. The algorithm is given a set $P \subseteq P_0$ and iterates over all $p_i \in P$. For each point p_i, it considers only the set of best rays R, i.e., the rays whose closest free position $d_{i,j}$ is minimal (within some bound to account for rounding errors). For each of these rays $r_{i,j} \in R$, the label ℓ_i is moved to $d_{i,j}$ resulting in label ℓ_i'. In the following loop, ℓ_i''s impact on all other labels ℓ_k of points $p_k \in P_0 \setminus \{p_i\}$ is calculated. For every ray $r_{k,l}$ of every point p_k, $k \neq i$, the algorithm computes the intersection interval J of label ℓ_k moving along the ray $r_{k,l}$ with label ℓ_i'. We then compute the integral of f over $I_{k,l} \setminus J$ as a measure of the remaining label space after obstruction by label ℓ_i'. This results in a new total $\int {p_k}^-$ for each point p_k which is stored in the list $V_{i,j}$. Thus, $V_{i,j}$ contains a measure for each point p_k representing the remaining label space after label ℓ_i is placed on ray $r_{i,j}$. Now, FINDBESTRAY chooses the ray $\max_{\forall i,j}\{\min_{v \in V_{i,j}}\{v\}\}$ which maximizes the minimum remaining label space for any label.

We use two observations to significantly reduce the runtime of our algorithm. Typically, the intersection tests are performed successively for all rays of a point. Thus, the computation time of our algorithm is dominated by the calculation of intersection intervals. It is therefore worthwhile to implement a fast rejection called PREINTERSECT, which determines for which rays an intersection can occur at all. In all our intersection tests, a label rectangle ℓ_i is translated by a certain vector $\vec{v}$ and intersected against circle, rectangle or line objects. We can approximate the label rectangle ℓ_i by an enclosing circle c_i and compute the minimum and maximum angle $\phi_{\min}$ and $\phi_{\max}$ for the vector $\vec{v}$ in between which the approximating circle c_i intersects the object (Fig. 5). Then, the exact intersection tests need to be performed only for rays $r_{i,j}$ with $\phi_{\min} \leq \phi_{i,j} \leq \phi_{\max}$.

The second observation is that we can abort the calculation of $V_{i,j}$ as soon as we encounter a $\int {p_k}^-$ less than the minimum of the best currently found V_{best}. Stopping the evaluation as early as possible gives a significant speed improvement. In order to find a good candidate ray earlier and thus reject later rays quicker, $p_i \in P$ are sorted in the beginning according to their initially available space in descending order. A label ℓ_i which has a lot of available space is likely to produce a better solution than any other label because it cannot obstruct other label positions.

The final RAYINTERSECTION algorithm on page 4 is very short. When a ray $r_{\text{best}} = r_{i,j}$ belonging to a point p_i and label ℓ_i has been found, the label is placed at position $\vec{v}_{i,j} \cdot d_{i,j}$. Placing a new label limits the available label positions of all other points. This needs to be reflected in the interval sets $I_{k,l}$ of every ray $r_{k,l}$ of every point $p_k \in P_0$, $i \neq k$. The method LABELRECTANGLEINTERSECTION computes the corresponding intersection interval $[a, b)$ which is then removed from $I_{k,l}$. RAYINTERSECTION is computable in $\mathcal{O}(|P_0|^3\, R^2)$ where R is the constant number of rays. Due to several optimizations we found its average complexity to be $\mathcal{O}(|P_0|^2\, R^2)$.

The IterativeGreedy Algorithm

The number of successfully placed labels can be improved over the naive application of RAYINTERSECTION by adapting the order in which labels are placed. The top-level ITERATIVEGREEDY algorithm is shown on page 5. RAYINTERSECTION is called first on P_0. This may result in a set of unlabeled points N, when the limited lookahead was unable to maintain free label positions for all labels. Often, these points in N are in dense clusters which are difficult to label. Therefore, it is a good idea to label the points in N first. Thus, RAYINTERSECTION is first called on N and then on the remaining points $P_0 \setminus N$. Generalizing this approach, let P_i be a queue of point sets. Initially, all points are in P_0. Assuming there were n non-empty point sets $P_0, \ldots, P_{n-1}$, RAYINTERSECTION is subsequently called on each set, starting with P_{n-1}. If for any point set P_i, after calling RAYINTERSECTION, a non-empty set N of unlabeled points is returned, the points in N are promoted to the group P_{i+1} which is labeled earlier in the next iteration. When a partition into point sets $P_0, \ldots, P_{n-1}$ is found such that all points can be labeled, the labeling result can often still be improved by promoting the label with

Algorithm 1: FindBestRay Algorithm

Input: A set of points $p_i \in P \subseteq P_0$ with rays $r_{i,j}$ for each point p_i.
Output: The best ray r_{best} to be labeled next.

1 sort $p_i \in P$ by available space $\int p_i$ in descending order;
2 $\int p^{\min} \leftarrow$ INT_MAX;
3 $r_{\text{best}} \leftarrow$ nil;
4 $V_{\text{best}} \leftarrow \emptyset$;
5 **foreach** $p_i \in P$ **do**
6 $\quad R \leftarrow \{\, r_{i,j} \mid d_{i,j} = \min_{\forall k}\{d_{i,k}\} + const\}$;
7 $\quad$ **foreach** $r_{i,j} \in R$ **do**
8 $\quad\quad V_{i,j} \leftarrow \emptyset$;
9 $\quad\quad \ell'_i \leftarrow \ell_i + \vec{v}_{i,j} \cdot d_{i,j}$;
10 $\quad\quad$ **foreach** $p_k \in P_0 \setminus \{p_i\}$ **do**
11 $\quad\quad\quad \int {p_k}^- \leftarrow 0$;
12 $\quad\quad\quad (\phi_{\min}, \phi_{\max}) \leftarrow$ PREINTERSECT(ℓ'_i, ℓ_k);
13 $\quad\quad\quad$ **foreach** *ray* $r_{k,l}$ *of* p_k *with* $\phi_{\min} \le \phi_{k,l} \le \phi_{\max}$ **do**
14 $\quad\quad\quad\quad J \leftarrow$ LABELRECTANGLEINTERSECTION$(\ell'_i, \ell_k, r_{k,l})$;
15 $\quad\quad\quad\quad \int {p_k}^- \leftarrow \int {p_k}^- + \int_{I_{k,l} \setminus J} f(x)\, dx$;
16 $\quad\quad\quad$ **end**
17 $\quad\quad\quad$ **if** $r_{\text{best}} \neq \text{nil} \wedge \int {p_k}^- < \int p^{\min}$ **then**
18 $\quad\quad\quad\quad$ continue with $r_{i,j+1}$;
19 $\quad\quad\quad$ **end**
20 $\quad\quad\quad V_{i,j}.\text{push}(\int {p_k}^-)$;
21 $\quad\quad$ **end**
22 $\quad\quad$ **if** $V_{i,j} < V_{\text{best}}$ *in lexicographic order* **then**
23 $\quad\quad\quad r_{\text{best}} \leftarrow r_{i,j}$;
24 $\quad\quad\quad V_{\text{best}} \leftarrow V_{i,j}$;
25 $\quad\quad\quad \int p^{\min} \leftarrow \min\{v \in V_{\text{best}}\}$;
26 $\quad\quad$ **end**
27 $\quad$ **end**
28 **end**
29 **return** r_{best};

the longest distance from its point. The algorithm continues as long as all points remain labeled and the total distance of all leader lines decreases. If RAYINTERSECTION returns an optimal solution we have $\neg promoted$ because all labels have been placed and subsequently searching the label with the longest connector will return nothing, i.e., all labels have been placed adjacent to their point. Therefore, the algorithm can terminate in line 25.

In its simplest form, ITERATIVEGREEDY as it is shown is not guaranteed to terminate. If a label exists that cannot even be placed as the first, the algorithm will loop infinitely. This label will eventually be in its own label set P_i. Labeling P_i will fail and RAYINTERSECTION returns the set $N = P_i$. In this case, the algorithm should not promote the set N, but instead remove this label and set $promoted$ to false. The algorithm will then continue as if all labels have been placed.

Algorithm 2: RayIntersection Algorithm

Input: A set of points $P \subseteq P_0$ with rays $r_{i,j}$ for each point $p_i \in P$.
Output: A set of points $N \subseteq P$ which could not be labeled.

1 $N \leftarrow \emptyset$;
2 **while** $P \neq \emptyset$ **do**
3 $\quad r_{i,j} \leftarrow$ FINDBESTRAY(P);
4 $\quad$ Place label ℓ_i at position $\vec{v}_{i,j} \cdot d_{i,j}$;
5 $\quad$ **foreach** $p_k \in P_0 \setminus \{p_i\}$ **do**
6 $\quad\quad (\phi_{\min}, \phi_{\max}) \leftarrow$ PREINTERSECT(ℓ_i, ℓ_k);
7 $\quad\quad$ **foreach** *ray* $r_{k,l}$ *of* p_k *with* $\phi_{\min} \le \phi_{k,l} \le \phi_{\max}$ **do**
8 $\quad\quad\quad [a, b) \leftarrow$ LABELRECTANGLEINTERSECTION$(\ell_i, \ell_k, r_{k,l}) \cap I_{k,l}$;
9 $\quad\quad\quad \int p_k \leftarrow \int p_k - \int_{[a,b)} f(x)\, dx$;
10 $\quad\quad\quad I_{k,l} \leftarrow I_{k,l} \setminus [a, b)$;
11 $\quad\quad$ **end**
12 $\quad\quad$ **if** $\int p_k = 0$ **then**
13 $\quad\quad\quad N \leftarrow N \cup \{p_k\}$;
14 $\quad\quad\quad P \leftarrow P \setminus \{p_k\}$;
15 $\quad\quad$ **end**
16 $\quad$ **end**
17 **end**
18 **return** N;

In another case of infinite running, the algorithm may cyclicly revisit the same label partitions. As a simple fix, the repeat ... until loop should be terminated after a constant number of iterations, returning the best solution found so far.

Evaluation

We chose to compare our algorithm's quality and performance with the results of three algorithms, selected for being the best performer in the survey they were tested in. In past papers, labeling algorithms have also been compared to random placement, Hirschs original heuristic or exponential optimal algorithms, all of which performed badly. In (Christensen, Marks, & Shieber 1995) the simulated annealing algorithm SIMANN with four possible label positions performed best. For the sliding label model we included both the approximation algorithm APPROX presented in (Strijk & van Kreveld 1999) and the more recent force-directed algorithm FDL described in (Ebner, Klau, & Weiskircher 2003). We used the implementations by Ebner et al. that are publicly available on the authors' website[1]. Besides the presented FULLALGORITHM, we also evaluated a simpler version of our algorithm called NOLOOKAHEAD that does not use any lookahead. Instead, FINDBESTRAY chooses the point with the least free space as the next point to label. This reduces our algorithm's complexity to $\mathcal{O}(|P_0|^2\, R)$. All performance data was measured on an AMD Athlon64 3800+

[1] http://www.ads.tuwien.ac.at/research/labeling/

Algorithm 3: IterativeGreedy Algorithm

Input: A set of points $p_i \in P_0$ to be labeled.

initialize $p_i \in P_0$ including rays and labels;
$n \leftarrow 1$;
$breakatlocalopt \leftarrow$ false;
while *true* **do**
  $\forall i: P'_i \leftarrow P_i$;
  $i \leftarrow n-1$;
  $promoted \leftarrow$ false;
  while $i \geq 0$ **do**
    $N \leftarrow$ RAYINTERSECTION(P_i);
    if $N \neq \emptyset$ **then**
      $P'_i \leftarrow P'_i \setminus N$;
      $P'_{i+1} \leftarrow P'_{i+1} \cup N$;
      $n \leftarrow n+1$;
      $promoted \leftarrow$ true;
    end
    $i \leftarrow i-1$;
  end
  if $\neg promoted$ **then**
    $breakatlocalopt \leftarrow$ true;
    Find label with longest connector $p \in P'_i$;
    if p *exists* **then**
      $P'_i \leftarrow P'_i \setminus \{p\}$ *// promote label*;
      $P'_{i+1} \leftarrow P'_i \cup \{p\}$;
    else
      $\forall i\ P^*_i \leftarrow P_i$ *// labeling is optimal*;
      break;
    end
  end
  if *first iteration or labeling* P_i *better than* P^*_i **then**
    $\forall i\ P^*_i \leftarrow P_i$;
  else if $breakatlocalopt$ **then**
    break *// We had a complete solution already*;
  end
  $\forall i\ P_i \leftarrow P'_i$;
end
place labels at stored solution P^*_i;

machine equipped with 1 GB RAM.

We compared these five algorithms first on the evenly distributed evaluation set used by Ebner et al. available on their website at the address mentioned above. We used the sets with up to 1300 labels which were generated to be always solvable. They were indeed relatively easy to label and both our NOLOOKAHEAD algorithm and the FDL algorithm maintained very good results even for large numbers of labels as Fig. 6 (a) demonstrates. Our NOLOOKAHEAD algorithm performed better than FDL, APPROX, and SIMANN when comparing the labeling quality. However, the tradeoff between time and quality does not seem to favor our algorithm (Fig. 6 (b)) as it is significantly slower than the others. Analysis of our algorithm's performance shows that it is slowed down by the larger number of points as well as a larger number of iterations of ITERATIVEGREEDY. The number of iterations increases very quickly early on, but later levels off at 18.

As a second test set we chose labeling problems generated by a Gaussian distribution, i.e., test sets with dense point clusters of up to 99 labels which more closely resemble real-world labeling problems, e.g. when labeling charts. Although the examples contained relatively few labels they proved to be very difficult as Fig. 7 (a) shows. The sets were small enough so we could also test our FULLALGORITHM. Both our scatter chart algorithms attained significantly better results than the best competitor, the force-directed labeling algorithm. The chart shows that our algorithms are able to frequently place all labels, which all others, using their limited placement options, never achieve. For very large numbers of labels, there is simply not enough space to fit all labels and the number of placed labels converges for all five algorithms. We chose to depict the computation time on logarithmic scale (Fig. 7 (b)) because it shows the similar runtime of NOLOOKAHEAD and FDL with NOLOOKAHEAD's labeling quality being far superior. The APPROX algorithm was faster than the Java timer resolution and is therefore not included.

Under the measure of placed labels, the advantage of using lookahead seems small, in particular considering its performance overhead. However, we measured the total distance of all labels to their points and found that the FULLALGORITHM version placed more labels and it placed some at least 20 % closer to their points than NOLOOKAHEAD (not shown).

Conclusion

The results show that our scatter chart labeling algorithm is able to solve more difficult labeling problems with dense label clusters than previously best competitors. Even for evenly distributed samples the solution quality is still superior to that of any other algorithm. In addition, our algorithmic framework is flexible enough to allow for easy adaptation to different label problems. We extended it to so-called bubble charts in which a point's third dimension is shown as its area or diameter. For these charts, labels can be placed inside and outside of bubbles, with the added difficulty that bubbles can partially intersect each other. It is similarly straightforward to incorporate user input by fixing labels to user-defined locations. These labels simply create additional obstructions initially excluded from the free intervals. Because users expect fast response times when interacting with a computer, we let the algorithm compute a labeling solution asynchronously, starting over whenever the user changes the input. On multi-processor machines, the algorithm runs in parallel.

The presented algorithm is used in a production environment encompassing over 15000 users worldwide at 120+ companies. Our users say that the quality of the automatic labeling quality is often superior to solutions created by humans due to its greater visual regularity.

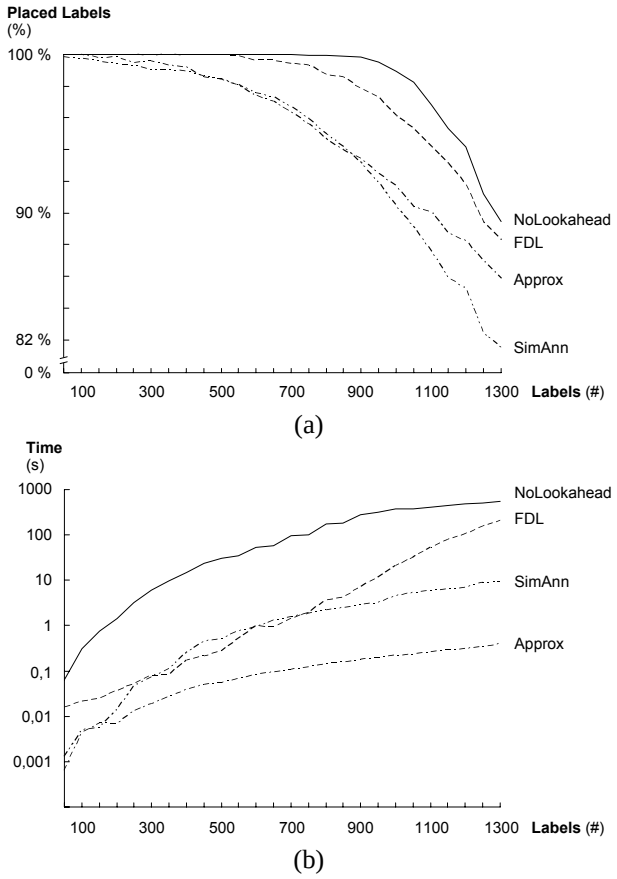

Figure 6: Results for labeling randomly distributed points.

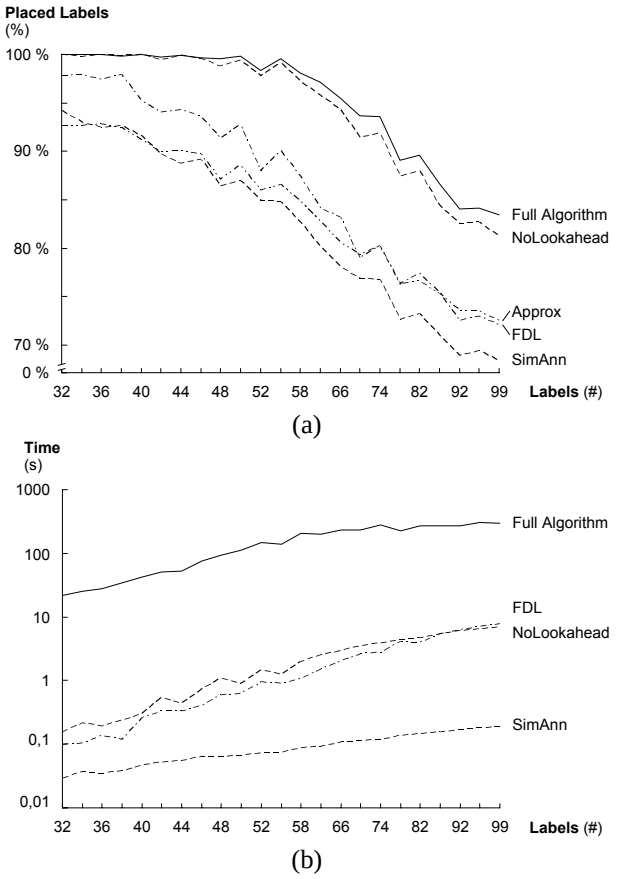

Figure 7: Results for labeling dense point clusters.

Acknowledgments

The authors would like to thank Markus Hannebauer and Hans-Dieter Burkhard for their valuable input.

References

Christensen, J.; Marks, J.; and Shieber, S. 1995. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics* 14(3):203 – 232.

do Nascimento, H. A. D., and Eades, P. 2003. User hints for map labelling. In *CRIPTS '03: Proceedings of the twenty-sixth Australasian computer science conference on Conference in research and practice in information technology*, 339–347. Australian Computer Society, Inc.

Ebner, D.; Klau, G. W.; and Weiskircher, R. 2003. Force-based label number maximization. Technical Report TR-186-1-03-02, TU Wien.

Formann, M., and Wagner, F. 1991. A packing problem with applications to lettering of maps. *Proceedings of the 7th Annual ACM Symposium on Computational Geometry* 281–288.

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

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

Iturriaga, C., and Lubiw, A. 1997. Np-hardness of some map labeling problems. Technical Report CS-97-18, University of Waterloo.

Marks, J., and Shieber, S. 1991. The computational complexity of cartographic label placement. Advanced Research in Computing Technology TR-05-91, Harvard University.

Müller, S., and Schödl, A. 2005. A smart algorithm for column chart labeling. In Butz, A.; Fisher, B.; Krüger, A.; and Olivier, P., eds., *Smart Graphics: 5th International Symposium, SG 2005, Frauenwörth Cloister, Germany*, volume 3638 of *Lecture Notes in Computer Science*, 127 – 137. Springer Verlag.

Strijk, T., and van Kreveld, M. 1999. Practical extensions of point labeling in the slider model. In *GIS '99: Proceedings of the 7th ACM international symposium on Advances in geographic information systems*, 47–52. New York, NY, USA: ACM Press.

van Kreveld, M.; Strijk, T.; and Wolff, A. 1999. Point labeling with sliding labels. *Computational Geometry: Theory and Applications* 13:21–47.