Labeling Moving Points with a Trade-Off between Label Speed and Label Overlap

Mark de Berg and Dirk H.P. Gerrits

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, The Netherlands

Abstract. Traditional map-labeling algorithms ensure that the labels being placed do not overlap each other, either by omitting labels or scaling them. This is undesirable in applications where the points to be labeled are moving. We develop and experimentally evaluate a heuristic for labeling moving points. Our algorithm labels all the points with labels of a fixed size, while trying to minimize the number of overlapping labels and ensuring smoothly moving labels. It allows a trade-off between label speed and label overlap.

1 Introduction

To do their jobs air-traffic controllers need a real-time visualization of the airplanes in their designated air space. Similarly, companies may want to track their fleet of taxis, trucks, or ships, and biologists may want to track wildlife tagged with GPS devices. A natural visualization for these kinds of applications is to represent each object as a moving point, and to place a *label* with each point that gives information about the object—an identifier, velocity and/or altitude, and so on. This leads to the *dynamic point-labeling problem*, which is the topic of our paper: how can we maintain a suitable labeling of a set of moving points in the plane?

Dynamic point labeling generalizes *static point labeling*, a problem in automated cartography that has attracted much attention; see the online Map Labeling Bibliography [9]. Here a static set of points (representing cities, say) is to be labeled (with their names). Such a *static labeling* should have readable labels, and an unambiguous association between labels and points. Readability is typically formalized by regarding the labels as axis-aligned rectangles slightly larger than the text they contain, and requiring that they be placed so that their interiors are disjoint. To associate labels with their points one usually requires each label to be placed so that it contains the point on its boundary. This can be done in several different ways, for example by requiring that the point is one of the four corners of the label (the 4-position model), or by allowing the point to be anywhere on the boundary (the 4-slider model). Other models, such as the 2-position model or the 1-slider model, have been studied as well.

Given a label model, one would like to label all of the points with non-overlapping labels. Unfortunately, this is not always possible and deciding

H.L. Bodlaender and G.F. Italiano (Eds.): ESA 2013, LNCS 8125, pp. 373–384, 2013.

whether this is the case is strongly NP-complete for most label models [5]. This led to the investigation of two optimization problems: the *size-maximization problem*, which asks for the maximum scaling factor for the labels that allows them to be placed without overlap, and the *number-maximization problem*, which asks for a maximum-cardinality subset of the points that allows a non-overlapping labeling. Results have been obtained for many variants of these problems.

One way to extend these results to dynamic point labeling would be to recompute a static labeling in real time—say 50 times per second. While several algorithms for static labeling are fast enough for this, and while the quality of the static solution is very good in practice, it still does not lead to satisfactory results. Indeed, algorithms for number-maximization would lead to labels appearing and disappearing between consecutive time steps, which is distracting for the user—note that heuristics for the dynamic number-maximization problem (see, for instance, [1,7,8] and their references) suffer from the same problem. Algorithms for size-maximization would lead to labels changing size (possibly in a non-continuous manner), which is disturbing as well. In an earlier paper [2], we therefore studied the *free-label maximization problem* which asks to label all points with labels of a given size, while maximizing the number of labels that are *free* (that is, interior-disjoint with all other labels). This avoids the downsides of the size- and number-maximization problems mentioned above. In our earlier paper we studied the static variant of the free-label maximization problem. In this paper we turn our attention to the dynamic version of the problem.

Our results. The formal problem setting is as follows. The input is a *dynamic point set* P, which specifies for each point $p \in P$ the time $\mathit{birth}(p)$ at which it is added to the point set, the time $\mathit{death}(p)$ at which it is removed, and its continuous trajectory between those times. For simplicity we assume that the trajectories are polygonal, although the results can be extended to curved trajectories. We refer to the time interval during which p is present in the point set as its *lifespan* and denote it by $\mathit{life}(p) = [\mathit{birth}(p), \mathit{death}(p)]$. Whenever $t \in \mathit{life}(p)$ we say that p is *alive* at time t, and then denote its position by $p(t)$. For such a dynamic point set P we compute a dynamic labeling $\mathcal{L}$, which for all t assigns a static labeling $\mathcal{L}(t)$ to the point set $P(t)$. Here $P(t) = \{p(t) \mid p \in P, t \in \mathit{life}(p)\}$ denotes the set of points that are alive at time t, at their respective locations. The label model we use is a slider model, where we require the axis-aligned label to be "behind" the point (with respect to its direction of movement) at all times. More precisely, the ray from the point to the center of its label should make an angle of at least 90° with the ray from the point in its direction of movement. This way, the label placement does not obscure the movement of the points.

Our global approach is simple: we compute a static labeling at regular intervals, and then interpolate between successive static labelings to obtain the dynamic labeling. For the static labeling we use an algorithm from our previous paper [2]; the crucial and novel part lies in the interpolation. Our solution has several attractive properties. Firstly, it is fast. The static labelings can be computed in $O(n \log n)$ time for n points, and interpolating takes time linear in

the combined complexity of the point trajectories. Secondly, the static labelings contain many free labels and the interpolation is such that it minimizes both the maximum speed of the labels as well as their average speed. Thirdly, the user can vary Δt, the time between successive time steps, to obtain a trade-off between label speed and label overlap. The trade-off turns out be to be very favorable in practice: a small sacrifice in label freeness can yield greatly reduced label speeds. Finally, the algorithm is "semi-online", in the sense that it only needs to know the trajectories and the times at which points are added or removed Δt time in advance.

2 A Heuristic Algorithm

As mentioned above, we propose a dynamic labeling algorithm that computes a series of static labelings and then moves the labels smoothly from their positions in one static labeling to their positions in the next. In computing the static labelings we try to maximize the number of free labels, in computing the interpolation between labelings we try to minimize the movement speed of the labels. Thus we hope to achieve a good dynamic labeling according to all criteria mentioned above. The algorithm can be expressed by the following pseudocode, where $\Delta t > 0$ is the time between successive static labelings, and $\mathcal{L}(t, t')$ refers to the part of a dynamic labeling $\mathcal{L}$ between times t and t'.

Algorithm InterpolativeLabeling(P, $t_{\max}$)
1. $\mathcal{L} \leftarrow \emptyset$; $t \leftarrow 0$
2. $\mathcal{L}(t) \leftarrow$ StaticLabeling(P, t)
3. **while** $t < t_{\max}$
4. **do** $t_{\text{next}} \leftarrow \min(t + \Delta t, t_{\max})$
5. $\mathcal{L}(t_{\text{next}}) \leftarrow$ StaticLabeling(P, t_{next})
6. $\mathcal{L}(t, t_{\text{next}}) \leftarrow$ Interpolate$(\mathcal{L}(t), t,\ \mathcal{L}(t_{\text{next}}), t_{\text{next}})$
7. $t \leftarrow t_{\text{next}}$
8. **return** $\mathcal{L}$

This method has three parameters. Firstly, the size of the time step Δt. Smaller time steps may lead to a greater number of free labels over time. Larger time steps may lead to less or slower label movement, and fewer calls to the StaticLabeling subroutine. Secondly, the algorithm used for StaticLabeling. We shall use a simple, greedy algorithm called FourGreedySweeps, which was described in an earlier paper [2]. For labels of equal dimensions that can be placed anywhere around their point it yields a constant-factor approximation to free-label maximization. The algorithm still works in our setting where labels may only be placed behind their points, but the proof of its approximation ratio unfortunately does not. Lastly, the algorithm used for Interpolate. We shall use the simple, linear-time algorithm described below. It minimizes both the average and the maximum movement speed of each label.

Interpolation algorithm. We are given two moments in time t_1 and t_2, and static labelings $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$ of the dynamic point set at those times. We then wish

to compute a dynamic labeling $\mathcal{L}(t_1, t_2)$ which *respects* them, that is, whose static labelings at times t_1 and t_2 equal $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$. We will do this independently for each moving point $p \in P$ with $life(p) \cap [t_1, t_2] \neq \emptyset$. For the rest of this section, let p be such a point and let $[t'_1, t'_2] = life(p) \cap [t_1, t_2]$.

Recall that the position of a label is restricted in that it must contain the point it labels on its boundary. Furthermore, the labels must trail "behind" the points. Specifically, a ray from the point in its direction of movement must make an angle of at least 90° with a ray from the point through its label's center. Now suppose we translate the coordinate system so that point p is always at the origin. The allowed positions for the center of p's label then trace out part of the surface of a box in 3-dimensional space-time. We refer to this *configuration space* for point p over the interval $[t'_1, t'_2]$ as $\mathcal{C}$. Figure 1(b) shows an example of such a configuration space for the point trajectory shown in Figure 1(a).

A label trajectory for p corresponds to a path through $\mathcal{C}$, monotone with respect to the time axis. To compute one, it will be convenient to "unfold" $\mathcal{C}$ into a rectilinear polygon R as in Figure 1(c). If point trajectory p has k vertices, then R is the union of $k - 1$ closed, axis-aligned rectangles. The intersection of any two consecutive rectangles is a vertical line segment which we refer to as a *portal*—see Figure 1(c). If a portal's coordinate along the time axis is t we refer to it as the *portal at* t, denoted by $\Psi(t)$. In addition to these $k - 2$ portals we define two extra portals at t'_1 and t'_2. These are simply the sets of label positions that respect $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$. If $t'_1 = t_1$ then $\Psi(t'_1)$ is a single point representing the label given to p by $\mathcal{L}(t_1)$. If $t'_1 \neq t_1$ then $\mathcal{L}(t_1)$ specifies nothing about p's label position at time t'_1, and $\Psi(t'_1)$ is simply the leftmost edge of R.

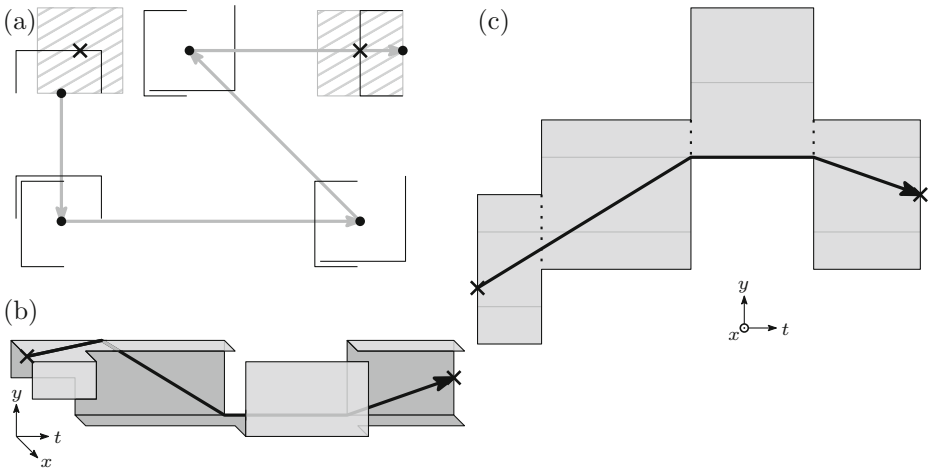

Fig. 1. (a) A piecewise linear point trajectory (in gray, with dots as vertices) with given labels (in gray, hatched, centers marked by crosses) at the endpoints. (b) The corresponding configuration space of allowed label positions. (c) An unfolding of (b) and a shortest path through it. Portals are shown as dotted line segments.

Analogously, $\Psi(t'_2)$ is either the rightmost edge of R, or a point representing p's label in $\mathcal{L}(t_2)$. With these definitions, a label trajectory for p corresponds to a time-monotone path through R from portal $\Psi(t'_1)$ to portal $\Psi(t'_2)$, passing through all intermediate portals in sequence. We will now argue that the shortest such path produces the most desirable label trajectory.

Lemma 1. *A shortest path from $\Psi(t'_1)$ to $\Psi(t'_2)$ through R yields a label trajectory minimizing both (i) the average speed and (ii) the maximum speed of p's label relative to p.*

Proof. (i) Let π be such a shortest path, which must be a t-monotone polygonal chain. Let $T(\pi)$ be the summed length of the projections onto the t-axis of the links of π, and let $Y(\pi)$ the same quantity for the y-axis. Every t-monotone path π' from $\Psi(t'_1)$ to $\Psi(t'_2)$ must traverse the same distance in the t-direction, namely $T(\pi') = T(\pi) = t'_2 - t'_1$. Since π is the shortest such path, it minimizes the distance traveled in y-direction, that is, $Y(\pi) \leqslant Y(\pi')$ for all π'. Thus π minimizes the average speed $Y(\pi)/T(\pi)$ of p's label.

(ii) Let ab be a steepest link of π, that is, ab has the maximum absolute slope among the links of π. Suppose that ab's projection onto the t-axis is $[t', t'']$ and that ab has negative slope (the case where it has positive slope is similar). Then π cannot make a left turn at a, as that would make ab less steep than the link preceding it. So π must either make a right turn at a, or start at a. If π makes a right turn at a, then a must be the bottom endpoint of portal $\Psi(t')$, as we could otherwise shorten π by a downward deformation at a. Since a lies above b, this makes ab the shortest path not only from a to b, but also from $\Psi(t')$ to b. If π instead starts at a, then $t'_1 = t'$, so the same condition holds. Reasoning symmetrically for b yields that ab must also be the shortest path from a to $\Psi(t'')$. We conclude that ab is the shortest path from $\Psi(t')$ to $\Psi(t'')$. This implies that any t-monotone path from $\Psi(t')$ to $\Psi(t'')$ must be at least as steep as ab somewhere between t' and t''. Thus π minimizes the maximum speed $Y(ab)/T(ab)$ of p's label. □

Theorem 1. *Suppose we are given a dynamic point set P with n points, along with static labelings $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$ of it at times t_1 and t_2. For each $p \in P$ let k_p be the number of vertices in its polygonal trajectory between times t_1 and t_2. In $O(\sum_{p\in P} k_p)$ time and $O(\max\{k_p \mid p \in P\})$ space we can then compute a dynamic labeling $\mathcal{L}(t_1, t_2)$ respecting $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$, that for each point minimizes both its average and its maximum label speed.*

Proof. Lemma 1 shows that $\mathcal{L}(t_1, t_2)$ can be obtained by computing a shortest path through the unfolded configuration space for each point. It remains to show that this can be done in $O(k_p)$ time and space for a point $p \in P$ with $\mathit{life}(p) \cap [t_1, t_2] = [t'_1, t'_2] \neq \emptyset$. As before, let R denote the simple, rectilinear polygon with $O(k_p)$ vertices that is p's unfolded configuration space. Through R we seek either a shortest point-to-point path (if $\mathit{life}(p) \supseteq [t_1, t_2]$), a shortest edge-to-edge path (if $\mathit{life}(p) \subset [t_1, t_2]$), or a shortest point-to-edge path (otherwise). This can be done in linear time using an algorithm by Lee and Preparata [6], with some additional techniques by Guibas et al. [4]. □

Hourglass trimming. In the previous section we described an algorithm to minimize both the average and the maximum speed of all labels in a dynamic labeling that interpolates between two given static labelings. As the experimental evaluation in the next section will show, however, high label speeds can still occur when a poor choice of static labelings is made. To see why this occurs, consider the example in Figure 2. Recall that we compute static labelings at regular intervals of Δt time. In the example, point p's trajectory makes a sharp left turn at time $t + \varepsilon$ for some small $\varepsilon > 0$. The static labeling algorithm completely disregards this when computing the labeling for time t, and comes up with the depicted leftmost label position for p. Now whatever labeling is picked for time $t + \Delta t$, the label for p will have to move substantially within ε time units. The problem here is that the chosen label position is just in the range of positions considered to be behind the point at time t, but the same position lies quite a bit in front of the point at time $t + \varepsilon$.

To fix this, we shall restrict the range of label positions that the static labeling algorithm is allowed to use, based on the trajectories of the points. We will need some definitions first. Consider a fixed point p, and let R be its unfolded configuration space. Let $\pi(a, b)$ denote the shortest path in R from a to b, for any $a, b \in R$. For a time interval $[t', t'']$ in which p is alive, we define the hourglass $H(t', t'')$ as the region enclosed by $\pi(a, c)$ and $\pi(b, d)$, where the segments ab and cd are the intersections of R with the vertical lines at t' and t''. Of the two paths $\pi(a, c)$ and $\pi(b, d)$ we call the upper one the hourglass's *upper chain*, and the other its *lower chain*. If the two chains intersect, then their intersection is a common subchain called the *string*. The hourglass is then called *closed*, and removal of the string leaves two connected components called *funnels*. If the hourglass is not closed, it is called *open*. Figure 3(a) illustrates these concepts.

Now consider a time instance $t \in \mathit{life}(p)$. Let $t_- = \max\big(\mathit{birth}(p), t - \Delta t\big)$, let $t_+ = \min\big(t + \Delta t, \mathit{death}(p)\big)$, and consider the hourglasses $H(t_-, t)$ and $H(t, t_+)$. Whatever label positions are chosen at times t_-, t, and t_+, the slowest interpolation between them (that is, the shortest path connecting them in R) must stay within the union of these two hourglasses. If $H(t_-, t)$ or $H(t, t_+)$ has steep edges, as in the example of Figure 3, then a fast moving label may result. We shall therefore narrow the ranges of valid label positions in such a way that these steep edges are "trimmed off" the hourglasses. If $H(t_-, t)$ is closed, then let $F_-(t)$ denote

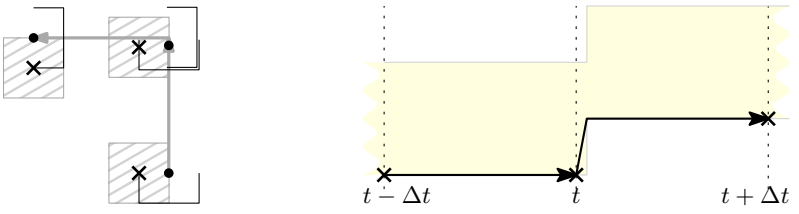

Fig. 2. An example of the static labeling algorithm making a poor choice with regards to the dynamic labeling

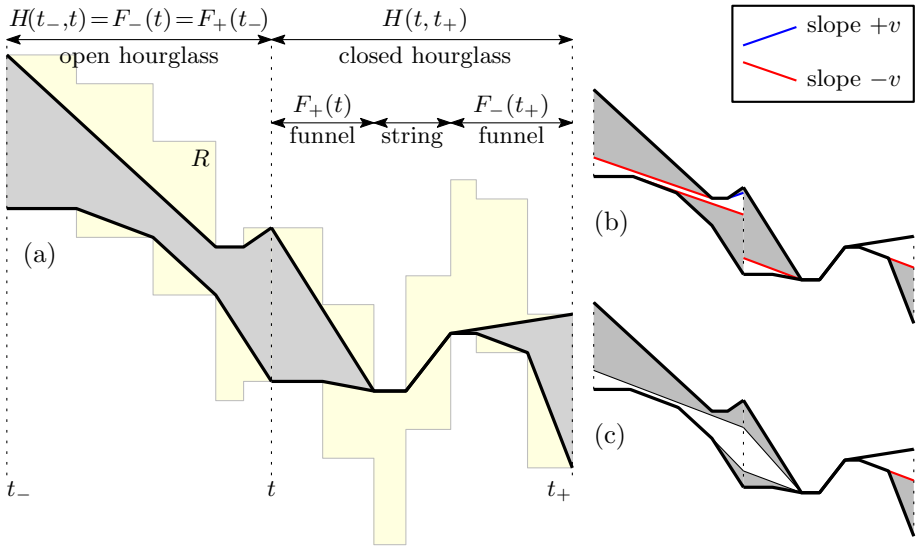

Fig. 3. (a) An example of an open hourglass (left) and a closed hourglass (right), the latter consisting of two funnels connected by a string. (b) The same hourglasses after trimming, shown in white. (c) When necessary (as in this case), the trimmed hourglasses are modified so that they connect to each other.

the rightmost funnel of $H(t_-,t)$, and let $F_-(t) = H(t_-,t)$ otherwise. Similarly, let $F_+(t)$ denote either $H(t,t_+)$ (if open) or its leftmost funnel (if closed). We assume we are given a parameter v that denotes a speed deemed reasonable for labels. We now translate a line with slope $+v$ down from positive infinity along the y-axis until it has become tangent to one of the two chains defining $F_-(t)$. Similarly, we translate a line with slope $-v$ up from negative infinity until it has become tangent to one of the two chains defining $F_-(t)$. These two lines define a narrower interval on the vertical line at t—see Figure 3(b). If we apply the same procedure at time t_- (using $F_+(t_-)$), then the new hourglass $H'(t_-,t)$ between the two narrowed intervals at t_- and t is the trimmed hourglass we are after. Specifically, the slowest label interpolation through $H'(t_-,t)$ cannot exceed the speed v, except in two cases. Firstly, this occurs when $H'(t_-,t)$ is closed, and its string contains an edge steeper than v, as on the right in Figure 3(b). In this case there is nothing that can be done to avoid exceeding the speed v with p's label. Secondly, this occurs when $H'(t_-,t)$ is open, and a bi-tangent of its upper and lower chain is steeper than v, as on the left in Figure 3(b). In this case it might be possible to trim the hourglass further, but sometimes this will simply result in a closed hourglass, making the previous case apply. Hence, we decided not to trim the hourglass further. Our experiments described in the next section show that our current method works quite well in practice.

In the same way, we compute the trimmed hourglass $H'(t,t_+)$, using $F_+(t)$ and $F_-(t_+)$. Typically, the narrowed interval at t that defines $H'(t_-,t)$ will differ from the narrowed interval at t that defines $H'(t,t_+)$. If they overlap this

poses no problem, as we may then simply take their intersection. In Figure 3(b), however, they are disjoint. In that case, there is nothing we can do to avoid high label speeds on both sides of t. We then take the interval in between the two, as in Figure 3(c). Note that this can undo some of the trimming, making the hourglass wider again.

3 Experimental Evaluation

We will now evaluate the effect of varying the time-step parameter Δt on the quality of the produced labeling, that is, on the number of free labels and on the speeds at which labels move. Intuitively, one would expect both the number of free labels and the label speeds to increase as the time step approaches 0. Thus there should be a trade-off between how many labels are free and how slow the labels move. Our main goal is to quantify this trade-off experimentally.

Computation time. We have measured the computation time of our C++ implementation only informally, just to ensure it was fast enough for use in interactive applications. On the modest hardware used for our experiments (an Intel Q6600 2.40GHz with 3GB RAM running Ubuntu 12.10), INTERPOLATION takes about 0.4 milliseconds to interpolate between two 100-point labelings, and 2.2 milliseconds between two 1,000-point labelings. Producing such labelings with FOUR-GREEDYSWEEPS takes about 5 milliseconds for 100 points, and 189 milliseconds for 1,000 points. Note that this is with a simple $O(n^2)$-time implementation, even though a more sophisticated $O(n \log n)$-time implementation is possible [2]. The latter could no doubt label 1,000 points much more quickly, but such large numbers of labels cannot be legibly displayed on a reasonably sized screen anyway. The extra engineering effort was therefore deemed unnecessary.

Experimental setup. To evaluate the quality of the produced dynamic labelings, we have used a network of streets in the Dutch city of Eindhoven (see Figure 5). Moving points were created to move along five polygonal paths through the network, at constant and equal speeds of 35 px/s. The arrival times of the points on each route were created by a Poisson process with parameter 5 s. This makes the time between arrivals of successive points on a route an exponentially distributed random variable with a mean of 5 s. Different seeds for the random number generator thus create different problem instances. We used 100 different seeds to create 100 problem instances. For each problem instance we used our algorithm to produce dynamic labelings from $t = 0$ to $t = t_{\max} = 60$ s for several different values of the time step Δt. To determine the quality of such a dynamic labeling $\mathcal{L}$ we did not compute the exact intervals of time during which each label was free. Instead, $\mathcal{L}$ was sampled at regular times at a rate of 25.6 samples per second (1537 samples total over 60 seconds). This is roughly the same framerate as used in movies (24 frames per second), but with the time between samples changed to have a finite binary floating point representation (from 1/24 s = 5/120 s to 5/128 s). For each sample we recorded the number of free and non-free labels, as well as the amount each label moved (relative to its point) since the last sample. In addition, we recorded the *free label area* for

each sample: the area covered by the union of the labels, minus the area covered by more than one label. In practice, this might be a more useful measure to maximize than the number of completely free labels. All of the above was done for several different time steps, the lowest being $\Delta t = 1/25.6$ s ≈ 0, so that each sample is labeled independently without regard for label speed, and the highest being $\Delta t = 61$ s > 60 s, so that label speeds are minimized without regard for label freeness. The graphs below offer a summary of the resulting data. The software used to generate the data and the graphs can be downloaded from `http://dirkgerrits.com/programming/flying-labels/`.

Results at a glance. The graphs in Figure 4 provides a high-level overview of the quality of labelings computed by our algorithm. The top graph shows how the label speed (measured in pixels per second), averaged over all moving points and over the whole time interval $[0, t_{\max}]$, decreases as we pick higher and higher values for Δt. The middle graph shows the same for the fraction of the labels that are completely free. The bottom graph, lastly, shows the free label area divided by the total area that would be spanned by the labels if they did not overlap. Each graph shows the minimum and maximum (dotted lines), 25% and 75% quantiles (dashed lines), and mean (solid line) over the 100 problem instances. The red lines show the results of the algorithm without hourglass trimming, the black lines show the results when hourglass trimming is used with a parameter of $v = 10$ px/s. In both cases we see a very sharp decline in label speeds until around $\Delta t = 2$ s, with a more modest corresponding decrease in label freeness. For higher Δt the decrease in label speeds slows down substantially while the decrease in label freeness continues. Thus, these preliminary results suggest that a time step of around 2 seconds should yield good labelings.

Detailed analysis of one instance. The effect of turning hourglass trimming on is similar to that of increasing the time step: the label speeds and freenesses both decrease. In this sense it forms an alternative to raising the time step. Hourglass trimming does something more, however, as can be seen when we examine a single problem instance in more details. We have selected an instance with particularly few free labels, but the effect can also be seen in other instances.

In Figure 6, seven graphs show how the label speeds of the individual moving points develop over time in this problem instance, for seven different values of Δt. Each of the moving points is drawn as a polyline, showing its speeds at all time samples. The red lines show the situation when hourglass trimming is not used. In the case of $\Delta t \approx 0$ almost all labels move rather violently (as was to be expected). For the other values of Δt, and especially the higher ones, a curious pattern appears. Label movement tends to be slow overall and decrease when Δt increases, but there are "spikes" of very high label movement near times that are a multiple of Δt. This effect is caused by a point changing direction near a multiple of Δt, as was explained in Section 2. This was our motivation for introducing hourglass trimming. The black lines show what happens when hourglass trimming is employed with the parameter $v = 10$ px/s. The regularly occurring spikes vanish, leaving label speeds with less variance. Note that the resulting label speeds do still exceed v, and by quite a bit for smaller time steps.

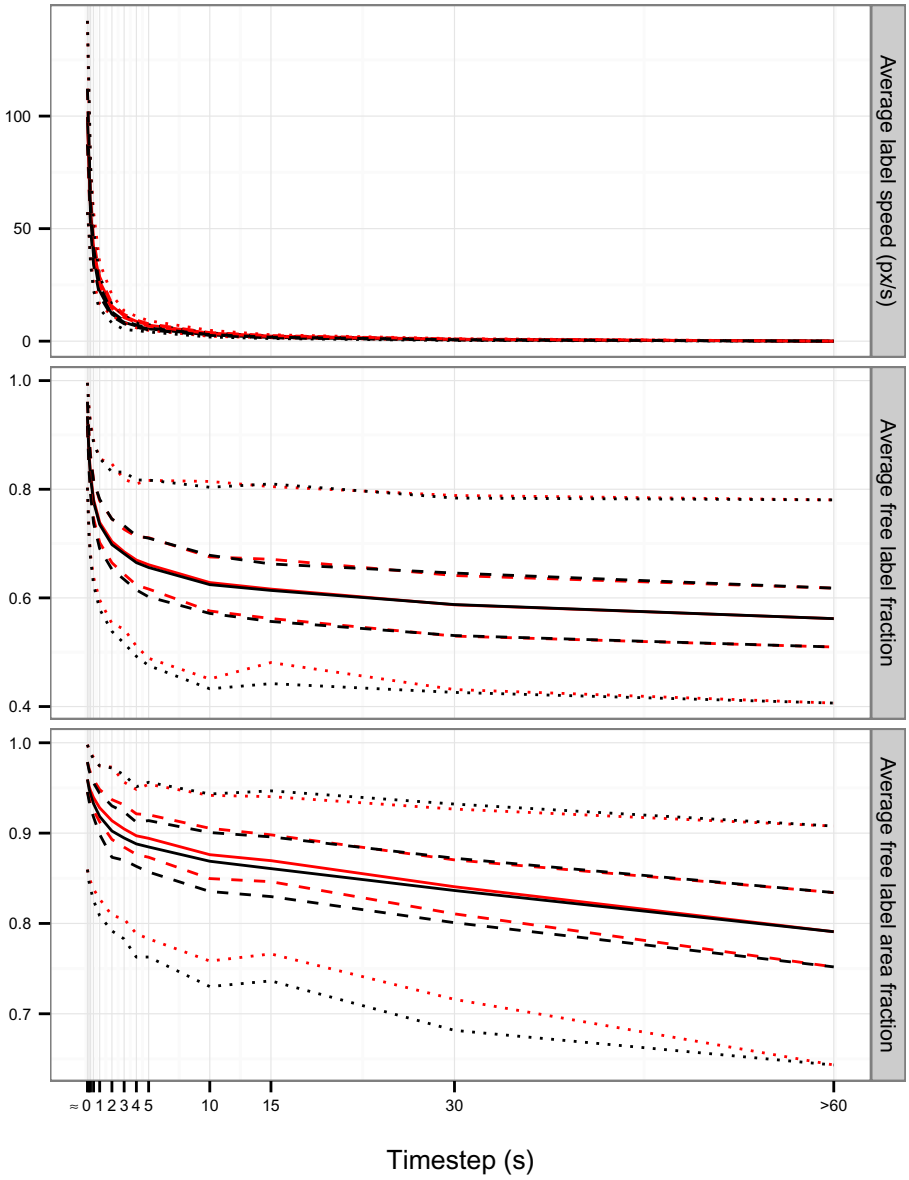

Fig. 4. The effects of varying the time step from $\Delta t \approx 0$ to $\Delta t > 60$ s on label speeds, number of free labels, and free label area, both with (black) and without (red) hourglass trimming. Shown are the minimum and maximum (dotted lines), 25% and 75% quantiles (dashed lines), and mean (solid line) over the 100 problem instances.

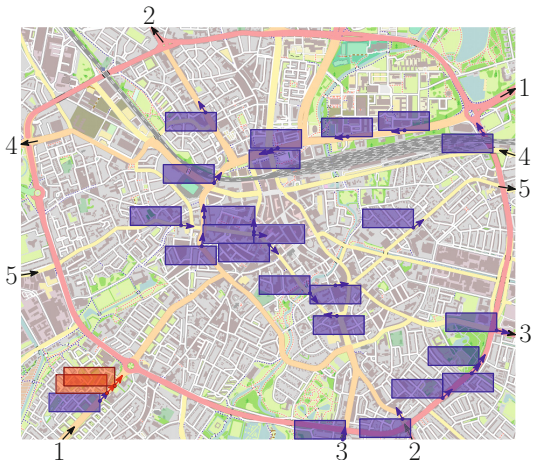

Fig. 5. The road network used for our experiments, and some labeled points moving across it along five routes. Blue labels are free, red labels are not.

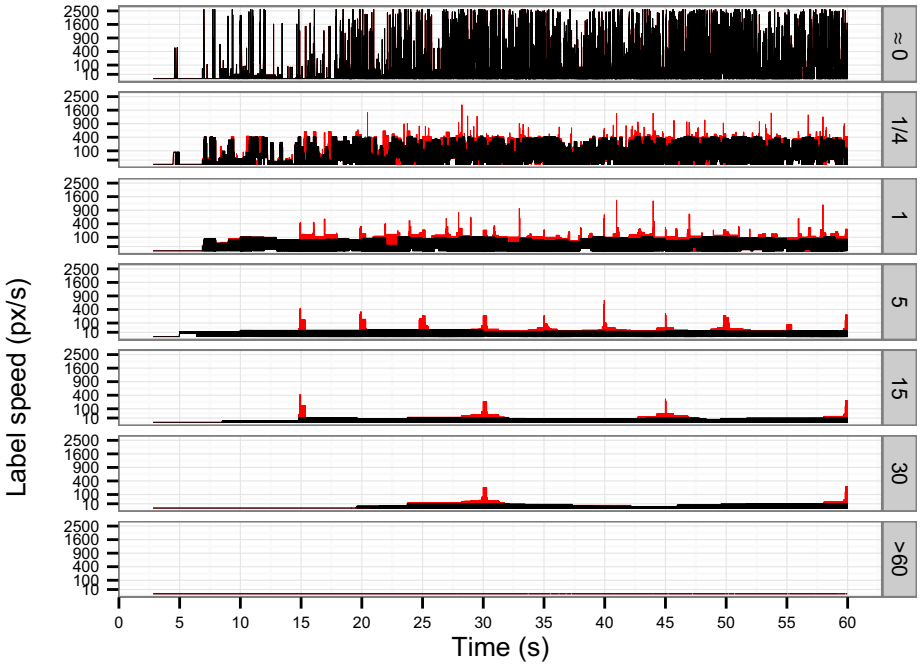

Fig. 6. The progression of label speeds over one particular problem instance, for several different time steps, both with (black) and without (red) hourglass trimming. Note that the y-axis uses a square root scale. With a linear scale the higher time steps would have indistinguishably low speeds, while with a logarithmic scale the lower time steps would have indistinguishably high speeds.

4 Conclusion

We developed a heuristic algorithm for free-label maximization on dynamic point sets, and evaluated it experimentally. The algorithm has been presented with the assumption that all points move on polygonal trajectories, but could be implemented just as well for curved trajectories. Instead of operating on polygons our algorithms will then work with curved *splinegons*. This change can be effected using techniques due to García-López and Ramos [3]. Our algorithm works by computing static labelings with many free labels at regular intervals, and then interpolating between these static labelings in a way that minimizes both the average and the maximum label speed. By varying the time between static labelings one obtains a trade-off between the number of free labels over time, and the speeds of the labels. The trade-off seemed favorable in experiments: a small increase in label speeds can yield a great increase in the number of free labels.

With these preliminary results we have only scratched the surface. From a theoretical point of view, algorithms with proven approximation ratios are still sorely lacking for dynamic labeling. From a practical point of view, there is room for improvement in other directions. While hourglass trimming was a step in the right direction, our method for choosing static labelings can undoubtedly be improved further. Testing on real-world data is also needed to get a more realistic picture of our method's performance.

References

1. Bell, B., Feiner, S., Höllerer, T.: View management for virtual and augmented reality. In: Proc. 14th ACM Sympos. User Interface Software and Technology (UIST 2001), pp. 101–110. ACM (2001)
2. de Berg, M., Gerrits, D.H.P.: Approximation algorithms for free-label maximization. Comput. Geom. Theory Appl. 45(4), 153–168 (2012)
3. García-López, J., Ramos, P.A.: A unified approach to conic visibility. Algorithmica 28(3), 307–322 (2000)
4. Guibas, L., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.E.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica 2(1), 209–233 (1987)
5. van Kreveld, M., Strijk, T., Wolff, A.: Point labeling with sliding labels. Comput. Geom. Theory Appl. 13, 21–47 (1999)
6. Lee, D.T., Preparata, F.P.: Euclidean shortest paths in the presence of rectilinear barriers. Networks 14(3), 393–410 (1984)
7. Rosten, E., Reitmayr, G., Drummond, T.: Real-time video annotations for augmented reality. In: Bebis, G., Boyle, R., Koracin, D., Parvin, B. (eds.) ISVC 2005. LNCS, vol. 3804, pp. 294–302. Springer, Heidelberg (2005)
8. Vaaraniemi, M., Treib, M., Westermann, R.: Temporally coherent real-time labeling of dynamic scenes. In: Proc. 3rd Internat. Conf. Computing for Geospatial Research and Applications, COM.Geo 2012, article no. 17 (2012)
9. Wolff, A., Strijk, T.: The Map Labeling Bibliography (2009), `http://liinwww.ira.uka.de/bibliography/Theory/map.labeling.html`