Trajectory-Based Dynamic Map Labeling

Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg

Karlsruhe Institute of Technology (KIT), Germany

Abstract. In this paper we introduce *trajectory-based labeling*, a new variant of dynamic map labeling where a movement trajectory for the map viewport is given. We define a general labeling model and study the active range maximization problem in this model. The problem is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard. In the restricted, yet practically relevant case that no more than k labels can be active at any time, we give polynomial-time algorithms. For the general case we present a practical ILP formulation with an experimental evaluation as well as approximation algorithms.

1 Introduction

In contrast to traditional static maps, dynamic digital maps support continuous movement of the map viewport based on panning, rotation, or zooming. Creating smooth visualizations under such map dynamics induces challenging geometric problems, e.g., continuous generalization [12] or dynamic map labeling [2]. In this paper, we focus on map labeling and take a trajectory-based view on it. In many applications, e.g., car navigation, a movement trajectory is known in advance and it becomes interesting to optimize the visualization of the map locally along this trajectory.

Selecting and placing a maximum number of non-overlapping labels for various map features is an important cartographic problem. Labels are usually modeled as rectangles and a typical objective in a static map is to find a maximum (possibly weighted) independent set of labels. This is known to be $\mathcal{NP}$-complete [6]. There are several approximation algorithms and PTAS's in different labeling models [1, 5], as well as practically useful heuristics [13, 14].

With the increasing popularity of interactive dynamic maps, e.g., as digital globes or on mobile devices, the static labeling problem has been translated into a dynamic setting. Due to the temporal dimension of the animations occurring during map movement, it is necessary to define a notion of *temporal consistency* or *coherence* for map labeling as to avoid distracting effects such as jumping or flickering labels [2]. Previously, consistent labeling has been studied from a global perspective under continuous zooming [3] and continuous rotation [8]. In practice, however, an individual map user with a mobile device, e.g., a tourist or a car driver, is typically interested only in a specific part of a map and it is thus often more important to optimize the labeling locally for a certain trajectory of the map viewport than globally for the whole map.

We introduce a versatile trajectory-based model for dynamic map labeling, and define three label activity models that guarantee consistency. We apply this model to point feature labeling for a viewport that moves and rotates along a differentiable trajectory in a fixed-scale base map in a forward-facing way. Although we present our approach in

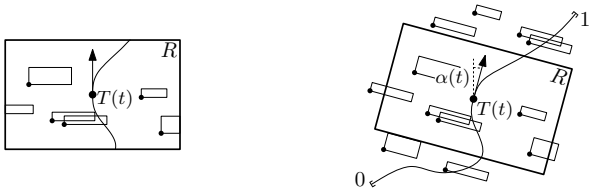

Fig. 1: Illustration of the viewport moving along a trajectory. Left the user's view and right a general view of the map and the viewport.

a very specific problem setting, our model is very general. Our approach can be applied for every dynamic labeling problem that can be expressed as a set of label availability intervals over time and a set of conflict intervals over time for pairs of labels. The exact algorithms hold for the general model, the approximation algorithm itself is also applicable, but the analysis of the approximation ratio requires problem-specific geometric arguments, which must be adjusted to the specific setting.

Contribution. For our specific problem, we show that maximizing the number of visible labels integrated over time in our model is $\mathcal{NP}$-complete; in fact it is even $\mathcal{W}[1]$-hard and thus it is unlikely that a fixed-parameter tractable algorithm exists. We present an integer linear programming (ILP) formulation for the general unrestricted case, which is supported by a short experimental evaluation. For the special case of unit-square labels we give an efficient approximation algorithm with different approximation ratios depending on the actual label activity model. Moreover, we present polynomial-time algorithms for the restricted case that no more than k labels are active at any time for some constant k. We note that limiting the number of simultaneously active labels is of practical interest as to avoid overly dense labelings, in particular for dynamic maps on small-screen devices such as in car navigation systems. Due to space constraints we omitted some proofs, which can be found in the full version [7] of the paper.

2 Trajectory-Based Labeling Model

Let M be a labeled north-facing, fixed-scale map, i.e., a set of points $P = \{p_1, \ldots, p_N\}$ in the plane together with a corresponding set $L = \{\ell_1, \ldots, \ell_N\}$ of labels. Each label ℓ_i is represented by an axis-aligned rectangle of individual width and height. We call the point p_i the *anchor* of the label ℓ_i. Here we assume that each label has an arbitrary but fixed position relative to its anchor, e.g., with its lower left corner coinciding with the anchor. The *viewport* R is an arbitrarily oriented rectangle of fixed size that defines the currently visible part of M on the map screen. The viewport follows a trajectory that is given by a continuous differentiable function $T\colon [0,1] \to \mathbb{R}^2$. For an example see Fig. 1. More precisely, we describe the viewport by a function $V\colon [0,1] \to \mathbb{R}^2 \times [0, 2\pi]$. The interpretation of $V(t) = (c, \alpha)$ is that at time t the center of the rectangle R is located at c and R is rotated clockwise by the angle α relatively to a north base line of the map. Since R moves along T we define $V(t) = (T(t), \alpha(t))$, where $\alpha(t)$ denotes the direction of T at time t. For simplicity, we sometimes refer to R at time t as $V(t)$. To ensure good readability, we require that the labels are always aligned with the viewport

axes as the viewport changes its orientation, i.e., they rotate around their anchors by the same angle $\alpha(t)$, see Fig. 1. We denote the rotated label rectangle of ℓ at time t by $\ell(t)$.

We say that a label ℓ is *present* at time t, if $V(t) \cap \ell(t) \neq \emptyset$. As we consider the rectangles $\ell(t)$ and $V(t)$ to be closed, we can describe the points in time for which ℓ is present by closed intervals. We define for each label ℓ the set Ψ_ℓ that describes all disjoint subintervals of $[0, 1]$ for which ℓ is present, thus $\Psi_\ell = \{[a, b] \mid [a, b] \subseteq [0, 1]$ is maximal so that ℓ is present at all $t \in [a, b]\}$. Further, we define the disjoint union $\Psi = \{([a, b], \ell) \mid [a, b] \in \Psi_\ell$ and $\ell \in L\}$ of all Ψ_ℓ. We abbreviate $([a, b], \ell) \in \Psi$ by $[a, b]_\ell$ and call $[a, b]_\ell \in \Psi$ a *presence interval* of ℓ. In the remainder of this paper we denote the number of presence intervals by n.

Two labels ℓ and ℓ' are in *conflict* with each other at time t if $\ell(t) \cap \ell'(t) \neq \emptyset$. If $\ell(t) \cap \ell'(t) \cap V(t) \neq \emptyset$ we say that the conflict is *present* at time t. As in [8] we can describe the occurrences of conflicts between two labels $\ell, \ell' \in L$ by a set of closed intervals: $C_{\ell,\ell'} = \{[a, b] \subseteq [0, 1] \mid [a, b]$ is maximal and ℓ and ℓ' are in conflict at all $t \in [a, b]\}$. We define the disjoint union $C = \{([a, b], \ell, \ell') \mid [a, b] \in C_{\ell,\ell'}$ and $\ell, \ell' \in L\}$ of all $C_{\ell,\ell'}$. We abbreviate $([a, b], \ell, \ell') \in C$ as $[a, b]_{\ell,\ell'}$ and call it a *conflict interval* of ℓ and ℓ'. Two presence intervals $[a, b]_\ell$ and $[c, d]_{\ell'}$ *are in conflict* if there is a conflict $[f, g]_{\ell,\ell'} \in C$ s.t. the intersection of the intervals $[f, g]_{\ell,\ell'} \cap [a, b]_\ell \cap [c, d]_{\ell'} \neq \emptyset$.

The tuple (P, L, Ψ, C) is called an *instance* of trajectory-based labeling. Note that the essential information of T is implicitly given by Ψ and C and that for each label $\ell \in L$ there can be several presence intervals. In this paper we assume that Ψ and C is given as input. In practice, however, we usually first need to compute Ψ and C given a continuous and differentiable trajectory T. An interesting special case is that T is a continuous, differentiable chain of m circular arcs (possibly of infinite radius), e.g., obtained by approximating a polygonal route in a road network. Niedermann [11] showed that in this case the set Ψ can be computed in $O(m \cdot N)$ time and the set C in $O(m \cdot N^2)$ time. His main observation was that for each arc of T the viewport can in fact be treated as a huge label and that "conflicts" with the viewport correspond to presence intervals. We refer to [11, Chapter 15] for details.

Next we define the *activity* of labels, i.e., when to actually display which of the present labels on screen. We restrict ourselves to closed and disjoint intervals describing the activity of a label ℓ and define the set $\Phi_\ell = \{[a, b] \subseteq [0, 1] \mid [a, b]$ is maximal such that ℓ is active at all $t \in [a, b]\}$, as well as the disjoint union $\Phi = \{([a, b], \ell) \mid [a, b] \in \Phi_\ell$ and $\ell \in L\}$ of all Φ_ℓ. We abbreviate $([a, b], \ell) \in \Phi$ with $[a, b]_\ell$ and call $[a, b]_\ell \in \Phi$ an *active interval* of ℓ.

It remains to define an *activity model* restricting Φ in order to obtain a reasonable labeling. Here we propose three activity models AM1, AM2, AM3 with increasing flexibility. All three activity models exclude overlaps of displayed labels and guarantee consistency criteria introduced by Been et al. [2], i.e., labels must not flicker or jump. To that end they share the following properties **(A)** a label ℓ can only be active at time t if it is present at time t, **(B)** to avoid flickering and jumping each presence interval of ℓ contains at most one active interval of ℓ, and **(C)** if two labels are in conflict at a time t, then at most one of them may be active at t to avoid overlapping labels.

What distinguishes the three models are the possible points in time when labels can become active or inactive. The first and most restrictive activity model AM1 demands

that each activity interval $[a, b]_\ell$ of a label ℓ must coincide with a presence interval of ℓ. The second activity model AM2 allows an active interval of a label ℓ to end earlier than the corresponding presence interval if there is a *witness label* ℓ' for that, i.e., an active interval for ℓ may end at time c if there is a starting conflict interval $[c, d]_{\ell,\ell'}$ and the conflicting label ℓ' is active at c. However, AM2 still requires every active interval to begin with the corresponding presence interval. The third activity model AM3 extends AM2 by also relaxing the restriction regarding the start of active intervals. An active interval for a label ℓ may start at time c if a present conflict $[a, c]_{\ell,\ell'}$ involving ℓ and an active witness label ℓ' ends at time c. In this model active intervals may begin later and end earlier than their corresponding presence intervals if there is a visible reason for the map user to do so, namely the start or end of a conflict with an active witness label.

A common objective in both static and dynamic map labeling is to maximize the number of labeled points. Often, however, certain labels are more important than others. To account for this, each label ℓ can be assigned a weight W_ℓ that corresponds to its significance. Then we define the weight of an interval $[a, b]_\ell \in \Phi$ as $w([a, b]_\ell) = (b-a) \cdot W_\ell$. Given an instance (P, L, Ψ, C), then with respect to one of the three activity models we want to find an activity Φ that maximizes $\sum_{[a,b]_\ell \in \Phi} w([a, b]_\ell)$; we call this optimization problem GENERALMAXTOTAL. If we require that at any time t at most k labels are active for some k, we call the problem k-RESTRICTEDMAXTOTAL. In particular the latter problem is interesting for small-screen devices, e.g., car navigation systems, that should not overwhelm the user with additional information.

3 Solving GENERALMAXTOTAL

We first prove that GENERALMAXTOTAL is $\mathcal{NP}$-complete. The membership of GENERALMAXTOTAL in $\mathcal{NP}$ follows from the fact that the start and the end of an active interval must coincide with the start or end of a presence interval or a conflict interval. Thus, there is a finite number of candidates for the endpoints of the active intervals so that a solution $\mathcal{L}$ can be guessed. Verifying that $\mathcal{L}$ is valid in one of the three models and that its value exceeds a given threshold can obviously be checked in polynomial time.

For the $\mathcal{NP}$-hardness we apply a straight-forward reduction from the $\mathcal{NP}$-complete maximum independent set of rectangles problem [6]. We simply interpret the set of rectangles as a set of labels with unit weight, choose a short vertical trajectory T and a viewport R that contains all labels at any point of T. Since the conflicts do no change over time, the reduction can be used for all three activity models. By means of the same reduction and Marx' result [10] that finding an independent set for a given set of axis-parallel unit squares is $\mathcal{W}[1]$-hard we derive the next theorem.

Theorem 1. GENERALMAXTOTAL *is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard for all activity models AM1–AM3.*

As a consequence, GENERALMAXTOTAL is not fixed-parameter tractable unless $\mathcal{W}[1] = \mathcal{FPT}$. Note that this also means that for k-RestrictedMaxTotal we cannot expect to find an algorithm that runs in $O(p(n) \cdot C(k))$ time, where $p(n)$ is a polynomial that depends only on the number n of presence intervals and the computable function $C(k)$ depends only on the parameter k.

3.1 Integer Linear Programming for GENERALMAXTOTAL

Since we are still interested in finding an optimal solution for GENERALMAXTOTAL we have developed integer linear programming (ILP) formulations for all three activity models. We present the formulation for the most involved model AM3 and then argue how to adapt it to the simpler models AM1 and AM2.

We define E to be the totally ordered set of the endpoints of all presence and all conflict intervals and include 0 and 1; see Fig. 2. We call each interval $[c, d]$ between two consecutive elements c and d in E an *atomic segment* and denote the i-th atomic segment of E by $E(i)$. Further, let $X(\ell, i)$ be the set of labels that are in conflict with ℓ during $E(i-1)$, but not during $E(i)$, i.e., the conflicts end with $E(i-1)$. Analogously, let $Y(\ell, i)$ be the set of labels that are in conflict with ℓ during $E(i+1)$, but not during $E(i)$, i.e., the conflicts begin with $E(i+1)$. For each label ℓ we introduce three binary variables $b_i, x_i, e_i \in \{0, 1\}$ and the following constraints.

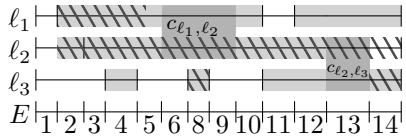

Fig. 2: Depiction of presence intervals (light gray), active intervals (hatched), and conflicts (dark gray).

$$b_i^\ell = x_i^\ell = e_i^\ell = 0 \qquad \forall 1 \le i \le |E| \text{ s.t. } \forall [c,d] \in \Psi_\ell : E(i) \cap [c,d] = \emptyset \quad (1)$$

$$\sum_{j \in J} b_j^\ell \le 1 \text{ and } \sum_{j \in J} e_j^\ell \le 1 \qquad \forall [c,d] \in \Psi_\ell \text{ where } J = \{j \mid E(j) \subseteq [c,d]\} \quad (2)$$

$$x_i^\ell + x_i^{\ell'} \le 1 \qquad \forall 1 \le i \le |E| \; \forall [c,d]_{\ell,\ell'} \in C : \; E(i) \subseteq [c,d] \quad (3)$$

$$x_{i-1}^\ell + b_i^\ell = x_i^\ell + e_{i-1}^\ell \qquad \forall 1 \le i \le |E| \text{ (set } x_0 = e_0 = 0) \quad (4)$$

$$b_j^\ell \le \sum_{\ell' \in X(\ell,j)} x_{j-1}^{\ell'} \qquad \forall [c,d]_\ell \in \Psi \; \forall E(j) \subset [c,d]_\ell \text{ with } c \notin E(j) \quad (5)$$

$$e_j^\ell \le \sum_{\ell' \in Y(\ell,j)} x_{j+1}^{\ell'} \qquad \forall [c,d]_\ell \in \Psi \; \forall E(j) \subset [c,d]_\ell \text{ with } d \notin E(j) \quad (6)$$

Subject to these constraints we maximize $\sum_{\ell \in L} \sum_{i=1}^{|E|-1} x_i^\ell \cdot w(E(i))$. The intended meaning of the variables is that $x_i^\ell = 1$ if ℓ is active during $E(i)$ and otherwise $x_i^\ell = 0$. Variable $b_i^\ell = 1$ if and only if $E(i)$ is the first atomic segment of an active interval of ℓ, and analogously $e_i^\ell = 1$ if and only if $E(i)$ is the last atomic segment of an active interval of ℓ. Recall the properties of the activity models as defined in Section 2. Constraints (1)–(3) immediately ensure properties (A)–(C), respectively. Constraint (4) means that if ℓ is active during $E(i-1)$ ($x_{i-1}^\ell = 1$), then it must either stay active during $E(i)$ ($x_i^\ell = 1$) or the active interval ends with $E(i-1)$ ($e_{i-1}^\ell = 1$), and if ℓ is active during $E(i)$ ($x_i^\ell = 1$) then it must be active during $E(i-1)$ ($x_{i-1}^\ell = 1$) or the active interval begins with $E(i)$ ($b_i^\ell = 1$). Constraint (5) enforces that for ℓ to become active with $E(j)$ at least one witness label of $X(\ell, j)$ is active during $E(j-1)$. Analogously, constraint (6) enforces that for ℓ to become inactive with $E(j)$ at least one witness label of $Y(\ell, j)$ is active during $E(j+1)$. Note that without the explicit constraints (5) and (6) two conflicting labels could switch activity at any point during the conflict interval rather than only at the endpoints.

Theorem 2. *Given an instance $I = (P, L, \Psi, C)$, the ILP (1)–(6) computes an optimal solution Φ of* GENERALMAXTOTAL *in AM3. It uses $O(N \cdot (|\Psi| + |C|))$ variables and constraints.*

We can adapt the above ILP to AM1 and AM2 as follows. For AM2 we replace the right hand side of constraint (5) by 0, and for AM1 we also replace the right hand side of constraint (6) by 0. This excludes exactly the start- and endpoints of the activity intervals that are forbidden in AM1 or AM2. It is easy to see that these ILP formulations can be modified further to solve k-RESTRICTEDMAXTOTAL by adding the constraint $\sum_{\ell \in L} x_i^\ell \leq k$ for each atomic segment $E(i)$.

Corollary 1. *Given an instance $I = (P, L, \Psi, C)$,* GENERALMAXTOTAL *and k-*RESTRICTEDMAXTOTAL *can be solved in AM1, AM2, and AM3 by an ILP that uses $O(N \cdot (|\Psi| + |C|))$ variables and constraints.*

Experiments. We have evaluated the ILP in all three models using Open Street Map data of the city center of Karlsruhe (Germany) which contains more than 2,000 labels. To this end we generated 1,000 shortest paths on the road network of Karlsruhe by selecting source and target vertices uniformly at random and transformed those shortest paths into trajectories consisting of circular arcs. The experimental evaluation in the full version [7] indicates that the ILP formulations are indeed applicable in practice.

3.2 Approximation of GENERALMAXTOTAL

In this section we describe a simple greedy algorithm for GENERALMAXTOTAL in all three activity models assuming that all labels are unit squares anchored at their lower-left corner. Further, we assume that the weight of each presence interval $[a, b]_\ell$ is its length $w([a, b]_\ell) = b - a$.

Starting with an empty solution Φ, our algorithm GREEDYMAXTOTAL removes the longest interval I from Ψ and adds it to Φ, i.e., I is set active. Then, depending on the activity model, it updates all presence intervals that have a conflict with I in Ψ and continues until the set Ψ is empty.

For AM1 the update process simply removes all presence intervals from Ψ that are in conflict with the newly selected interval I. For AM2 and AM3 let $I_j \in \Psi$ and let $I_j^1, \ldots, I_j^k$ be the longest disjoint sub-intervals of I_j that are not in conflict with the selected interval I. We assume that $I_j^1, \ldots, I_j^k$ are sorted by their left endpoint. The update operation for AM2 replaces every interval $I_j \in \Psi$ that is in conflict with I with I_j^1. In AM3 we replace I_j by I_j^1, if I_j^1 is not fully contained in I. Otherwise, I_j is replaced by I_j^k. Note that this discards some candidate intervals, but the chosen replacement of I_j is enough to prove the approximation factor. Note that after each update all intervals in Ψ are valid choices according to the specific model. Hence, we can conclude that the result Φ of GREEDYMAXTOTAL is also valid in that model.

In the following we analyze the approximation quality of GREEDYMAXTOTAL. To that end we first introduce a purely geometric packing lemma. Similar packing lemmas have been introduced before, but to the best of our knowledge for none of them it is sufficient that only one prescribed corner of the packed objects lies within the container.

Lemma 1. *Let C be a circle of radius $\sqrt{2}$ in the plane and let Q be a set of non-intersecting closed and axis-parallel unit squares with their bottom-left corner in C. Then Q cannot contain more than eight squares.*

Based on Lemma 1 we now show that for any label with anchor p there is no point of time $t \in [0, 1]$ for which there can be more than eight active labels whose anchors are within distance $\sqrt{2}$ of p. We call a set $X \subseteq \Psi$ *conflict-free* if it contains no pair of presence intervals that are in conflict. Further, we say that X is in conflict with $I \in \Psi$ if every element of X is in conflict with I, and we say that X contains $t \in [0, 1]$ if every element of X contains t.

Lemma 2. *For every $t \in [0, 1]$ and every $I \in \Psi$ any maximum cardinality conflict-free set $X_I(t) \subseteq \Psi$ that is in conflict with I and contains t satisfies $|X_I(t)| \leq 8$.*

With this lemma we can finally obtain the approximation guarantees for GREEDY-MAXTOTAL for all activity models.

Theorem 3. *Assuming that all labels are unit squares and $w([a, b]) = b - a$,* GREEDY-MAXTOTAL *is a 1/24-, 1/16-, 1/8-approximation for AM1–AM3, respectively, and needs $O(n \log n)$ time for AM1 and $O(n^2)$ time for AM2 and AM3.*

Proof. To show the approximation ratios, we consider an arbitrary step of GREEDY-MAXTOTAL in which the presence interval $I = [a, b]_\ell$ is selected from Ψ. Let C_ℓ^I be the set of presence intervals in Ψ that are in conflict with I.

Consider the model AM1. Since I is the longest interval in Ψ when it is chosen, the intervals in C_ℓ^I must be completely contained in $J = [a - w(I), b + w(I)]$. As C_ℓ^I contains all presence intervals that are in conflict with I it is sufficient to consider J to bound the effect of selecting I. Obviously, the interval J is three times as long as I. By Lemma 2 we know that for any $X_I(t)$ it holds that $|X_I(t)| \leq 8$ for all $t \in J$. Hence, in an optimal solution there can be at most eight active labels at each point $t \in J$ that are discarded when $[a, b]_\ell$ is selected. Thus, the cost of selecting $[a, b]_\ell$ is at most $3 \cdot 8 \cdot w(I)$.

For AM2 we apply the same arguments, but restrict the interval J to $J = [a, b + w(I)]$, which is only twice as long as I. To see that consider for an interval $[c, d]_{\ell'} \in C_\ell^I$ the prefix $[c, a]$ if it exists. If $[c, a]$ does not exist (because $a < c$), removing $[c, d]_{\ell'}$ from Ψ changes Ψ only in the range of J. If $[c, a]$ exists, then again Ψ is only changed in the range of I, because by definition $[c, d]_{\ell'}$ is shortened to an interval that at least contains $[c, a]$ and is still contained in Ψ. Thus, the cost of selecting I is at most $2 \cdot 8 w(I)$.

Analogously, for AM3 we can argue that it is sufficient to consider the interval $J = [a, b]$. By definition of the update operation of GREEDYMAXTOTAL at least the prefix or suffix subinterval of each $[c, d]_{\ell'} \in C_\ell^I$ remains in Ψ that extends beyond I (if such an interval exists). Thus, selecting I influences only the interval J and its cost is at most $8w(I)$. The approximation bounds of $1/24$, $1/16$, and $1/8$ follow immediately.

We use a heap to achieve the time complexity $O(n \log n)$ of GREEDYMAXTOTAL for AM1 since each interval is inserted and removed exactly once. For AM2 and AM3 we use a linear sweep to identify the longest interval contained in Ψ. In each step we need $O(n)$ time to update all intervals in Ψ, and we need a total of $O(n)$ steps. Thus, GREEDYMAXTOTAL needs $O(n^2)$ time in total for AM2 and AM3. □

4 Solving k-RESTRICTEDMAXTOTAL

Corollary 1 showed that k-RESTRICTEDMAXTOTAL can be solved by integer linear programming in all activity models. In this section we prove that unlike GENERALMAXTOTAL the problem k-RESTRICTEDMAXTOTAL can actually be solved in polynomial time. We give a detailed description of our algorithm for AM1, and then show how it can be extended to AM2. Note that solving k-RESTRICTEDMAXTOTAL is related to finding a maximum cardinality k-colorable subset of n intervals in interval graphs. This can be done in polynomial time in both n and k [4]. However, we have to consider additional constraints due to conflicts between labels, which makes our problem more difficult. First, we discuss how to solve the case for $k = 1$, then give an algorithm that solves k-RESTRICTEDMAXTOTAL for $k = 2$, and finally extend this result recursively to any constant $k > 2$.

4.1 An Algorithm for 2-RESTRICTEDMAXTOTAL

We start with some definitions before giving the actual algorithm. We assume that the intervals of $\Psi = \{I_1, \ldots, I_n\}$ are sorted in non-decreasing order by their left endpoints; ties are broken arbitrarily. First note that for the case that at most one label can be active at any given point in time ($k = 1$), conflicts between labels do not matter. Thus, it is sufficient to find an independent subset of Ψ of maximum weight. This is equivalent to finding a maximum weight independent set on interval graphs, which can be done in $O(n)$ time using dynamic programming given n sorted intervals [9]. We denote this algorithm by $\mathcal{A}_1$. Let $L_1[I_j]$ be the set of intervals that lie completely to the left of the left endpoint of I_j. Algorithm $\mathcal{A}_1$ basically computes a table $\mathcal{T}_1$ indexed by the intervals in Ψ, where an entry $\mathcal{T}_1[I_j]$ stores the value of a maximum weight independent set Q of $L_1[I_j]$ and a pointer to the rightmost interval in Q.

We call a pair of presence intervals (I_i, I_j), $i < j$, a *separating pair* if I_i and I_j overlap and are not in conflict with each other. Further, a separating pair $\boldsymbol{v} = (I_p, I_q)$ is *smaller* than another separating pair $\boldsymbol{w} = (I_i, I_j)$ if and only if $p < i$ or $p = i$ and $q < j$. This induces a total order and we denote the ordered set of all separating pairs by S_2=$\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_z\}$. The weight of a separating pair $\boldsymbol{v}$ is defined as $w(\boldsymbol{v}) = \sum_{I \in \boldsymbol{v}} w(I)$.

We observe that a separating pair $\boldsymbol{v} = (I_i, I_j)$ contained in a solution of 2-RESTRICTEDMAXTOTAL splits the set of presence intervals into two independent subsets. Specifically, a left (right) subset $L_2[\boldsymbol{v}]$ ($R_2[\boldsymbol{v}]$) that contains only intervals which lie completely to the left (right) of the intersection of I_i and I_j and are neither in conflict with I_i nor I_j; see Fig. 3.

We are now ready to describe our dynamic programming algorithm $\mathcal{A}_2$. For ease of notation we add two dummy separating pairs to S_2. One pair $\boldsymbol{v}_0$ with presence intervals strictly to the left of 0 and one pair $\boldsymbol{v}_{z+1}$ with presence intervals strictly to the right of 1. Since all original presence intervals are completely contained in $[0, 1]$ every optimal solution contains both dummy separating pairs. Our algorithm computes a one-dimensional table $\mathcal{T}_2$, where for each separating pair $\boldsymbol{v}$ there is an entry $\mathcal{T}_2[\boldsymbol{v}]$ that stores the value of the optimal solution for $L_2[\boldsymbol{v}]$. We compute $\mathcal{T}_2$ from left to right starting with the dummy separating pair $\boldsymbol{v}_0$ and initialize $\mathcal{T}_2[\boldsymbol{v}_0] = 0$. Then, we recursively define $\mathcal{T}_2[\boldsymbol{v}_j]$ for every $\boldsymbol{v}_j \in S_2$ as $\mathcal{T}_2[\boldsymbol{v}_j] = \max_{i<j}\{\mathcal{T}_2[\boldsymbol{v}_i] + w(\boldsymbol{v}_i) + \mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j) \mid \boldsymbol{v}_i \in$

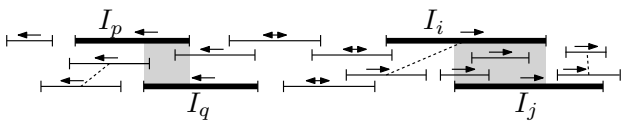

Fig. 3: Illustration of presence intervals. Intervals that are in conflict are connected by a dotted line. Both (I_i, I_j) and (I_p, I_q) are separating pairs. The intervals of $L_2[i, j]$ ($R_2[p, q]$) are marked by a left (right) arrow.

S_2, $\boldsymbol{v}_i \subseteq L_2[\boldsymbol{v}_j]$, $\boldsymbol{v}_j \subseteq R_2[\boldsymbol{v}_i]\}$. Additionally, we store a backtracking pointer to the predecessor pair that yields the maximum value. In other words, for computing $\mathcal{T}_2[\boldsymbol{v}_j]$ we consider all possible direct predecessors $\boldsymbol{v}_i \in S_2$ with $i < j$, $\boldsymbol{v}_i \cap \boldsymbol{v}_j = \emptyset$, and no conflict with $\boldsymbol{v}_j$. Each such $\boldsymbol{v}_i$ induces a candidate solution whose value is composed of $\mathcal{T}_2[\boldsymbol{v}_i]$, $w(\boldsymbol{v}_i)$, and the value of an optimal solution of algorithm $\mathcal{A}_1$ for the intervals between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ with $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ active.

Since by construction $L_2[\boldsymbol{v}_{z+1}] = \Psi \cup \boldsymbol{v}_0$, the optimal solution to 2-RESTRICTEDMAXTOTAL is stored in $\mathcal{T}_2[\boldsymbol{v}_{z+1}]$ once $\boldsymbol{v}_0$ is removed. To compute a single entry $\mathcal{T}_2[\boldsymbol{v}_j]$ our algorithm needs to consider all possible separating pairs preceding $\boldsymbol{v}_j$, and for each of them obtain the optimal solution from algorithm $\mathcal{A}_1$ under some additional constraints. For the call $\mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j)$ in the recursive equation above, we distinguish two cases. If the rightmost endpoint of $\boldsymbol{v}_i$ is to the left of the leftmost endpoint of $\boldsymbol{v}_j$ then we run algorithm $\mathcal{A}_1$ on the set of intervals $L_2[\boldsymbol{v}_j] \cap R_2[\boldsymbol{v}_i]$ and obtain the value $\mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j)$. Otherwise, there is an overlap between an interval I_a of $\boldsymbol{v}_i$ and an interval I_b of $\boldsymbol{v}_j$. Since for $k = 2$ no other interval can cross this overlap, we actually make two calls to $\mathcal{A}_1$, once on the set $R_2[\boldsymbol{v}_i] \cap L_2[(I_a, I_b)]$ and once on the set $R_2[(I_a, I_b)] \cap L_2[\boldsymbol{v}_j]$. We add both values to obtain $\mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j)$. Since we run algorithm $\mathcal{A}_1$ for each of $O(z)$ separating pairs, the time complexity to compute a single entry of $\mathcal{T}_2$ is $O(nz)$. To compute the whole table the algorithm repeats this step $O(z)$ times, which yields a total time complexity of $O(nz^2)$. Note that the number of separating pairs z is in $O(n^2)$.

We prove the correctness of the algorithm by contradiction. Assume that there exists an instance for which our algorithm does not compute an optimal solution and let OPT be an optimal solution. This means, that there is a smallest separating pair $\boldsymbol{v}_j$ for which the entry in $\mathcal{T}_2[\boldsymbol{v}_j]$ is less than the value of OPT for $L_2[\boldsymbol{v}_j]$. Note that $\boldsymbol{v}_j$ cannot be the dummy separating pair $\boldsymbol{v}_0$ since $\mathcal{T}_2[\boldsymbol{v}_0]$ is trivially correct. Let $\boldsymbol{v}_i$ be the rightmost separating pair in OPT that precedes $\boldsymbol{v}_j$ and is disjoint from it (possibly $\boldsymbol{v}_i = \boldsymbol{v}_0$). Since there is no other disjoint separating pair between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ in OPT, all intervals in OPT between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ form a subset of $R_2[\boldsymbol{v}_i] \cap L_2[\boldsymbol{v}_j]$ that is a valid configuration for $k = 1$. We can obtain an optimal solution for $k = 1$ of the intervals in $R_2[\boldsymbol{v}_i] \cap L_2[\boldsymbol{v}_j]$ by computing $\mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j)$ as described above. Since, by assumption, $\mathcal{T}_2[\boldsymbol{v}_i]$ is optimal, $\mathcal{A}_1$ is correct [9], and our algorithm explicitly considers all possible preceding separating pairs including $\boldsymbol{v}_i$, the entry $\mathcal{T}_2[\boldsymbol{v}_j]$ must be at least as good as OPT for $L[\boldsymbol{v}_j]$. This is a contradiction and the correctness of $\mathcal{A}_2$ follows.

Theorem 4. *Algorithm $\mathcal{A}_2$ solves* 2-RESTRICTEDMAXTOTAL *in AM1 in $O(nz^2)$ time and $O(z)$ space, where z is the number of separating pairs in the input instance.*

4.2 An Algorithm for k-RESTRICTEDMAXTOTAL

In the following we extend the dynamic programming algorithm $\mathcal{A}_2$ to a general algorithm $\mathcal{A}_k$ for the case $k > 2$. To this end, we extend the definition of separating pairs to separating k-tuples. A *separating k-tuple* $\boldsymbol{v}$ is a set of k presence intervals that are not in conflict with each other and that have a non-empty intersection $Y_{\boldsymbol{v}} = \bigcap_{I \in \boldsymbol{v}} I$. We say a separating k-tuple $\boldsymbol{v}$ is *smaller* than a separating k-tuple $\boldsymbol{w}$ if $Y_{\boldsymbol{v}}$ begins to the left of $Y_{\boldsymbol{w}}$. Ties are broken arbitrarily. This lets us define the ordered set $S_k = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_z\}$ of all separating k-tuples of a given set of presence intervals. We say a set C of presence intervals is *k-compatible* if no more than k intervals in C intersect at any point and there are no conflicts in C. Two separating k-tuples $\boldsymbol{v}$ and $\boldsymbol{w}$ are *k-compatible* if they are disjoint and $\boldsymbol{v} \cup \boldsymbol{w}$ is k-compatible. The definitions of the sets $R_2[\boldsymbol{v}]$ and $L_2[\boldsymbol{v}]$ extend naturally to the sets $R_k[\boldsymbol{v}]$ and $L_k[\boldsymbol{v}]$ of all intervals completely to the right (left) of $Y_{\boldsymbol{v}}$ and not in conflict with any interval in $\boldsymbol{v}$. Now, we recursively define the algorithm $\mathcal{A}_k$ that solves k-RESTRICTEDMAXTOTAL given a pair of active k-compatible boundary k-tuples. Note that in the recursive definition these boundary tuples may remain k-dimensional even in $\mathcal{A}_{k'}$ for $k' < k$. For $\mathcal{A}_k$ we define as boundary tuples two k-compatible dummy separating k-tuples $\boldsymbol{v}_0$ and $\boldsymbol{v}_{z+1}$ with all presence intervals strictly to the left of 0 and to the right of 1, respectively. The algorithm fills a one-dimensional table $\mathcal{T}_k$. Similarly to the case $k = 2$, each entry $\mathcal{T}_k[\boldsymbol{v}]$ stores the value of the optimal solution for $L_k[\boldsymbol{v}]$, i.e., the final solution can again be obtained from $\mathcal{T}_k[\boldsymbol{v}_{z+1}]$. We initialize $\mathcal{T}_k[\boldsymbol{v}_0] = 0$. Then, the remaining entries of $\mathcal{T}_k$ can be obtained by computing $\mathcal{T}_k[\boldsymbol{v}_j] = \max_{i<j}\{\mathcal{T}_k[\boldsymbol{v}_i] + w(\boldsymbol{v}_i) + \mathcal{A}_{k-1}(\tilde{\boldsymbol{v}}_i, \tilde{\boldsymbol{v}}_j) \mid \boldsymbol{v}_i \in S_k,\ \boldsymbol{v}_i \subseteq L_k[\boldsymbol{v}_j] \cup \boldsymbol{v}_0,\ \boldsymbol{v}_j \subseteq R_k[\boldsymbol{v}_i] \cup \boldsymbol{v}_{z+1},\ \boldsymbol{v}_0 \cup \boldsymbol{v}_{z+1} \cup \boldsymbol{v}_i \cup \boldsymbol{v}_j \text{ is } k\text{-compatible}\}$, which uses the algorithm $\mathcal{A}_{k-1}$ recursively on a suitable subset of presence intervals between the boundary tuples $\tilde{\boldsymbol{v}}_i$ and $\tilde{\boldsymbol{v}}_j$. Here $\tilde{\boldsymbol{v}}_i$ is defined as the union of the tuple $\boldsymbol{v}_i$ and all intervals in $\boldsymbol{v}_0 \cup \boldsymbol{v}_{z+1}$ that intersect the right endpoint of $Y_{\boldsymbol{v}_i}$; analogously $\tilde{\boldsymbol{v}}_j$ is defined as the union of the tuple $\boldsymbol{v}_j$ and all intervals in $\boldsymbol{v}_0 \cup \boldsymbol{v}_{z+1}$ that intersect the left endpoint of $Y_{\boldsymbol{v}_i}$. This makes sure that in each subinstance all active intervals that are relevant for that particular subinstance are known. Note that by the k-compatibility condition $\tilde{\boldsymbol{v}}_i$ and $\tilde{\boldsymbol{v}}_j$ contain at most k elements each. In fact, $\mathcal{A}_{k-1}(\tilde{\boldsymbol{v}}_i, \tilde{\boldsymbol{v}}_j)$ uses $\tilde{\boldsymbol{v}}_i$ and $\tilde{\boldsymbol{v}}_j$ as boundary k-tuples (and thus does not create dummy boundary tuples) and the set $R_k[\boldsymbol{v}_i] \cap L_k[\boldsymbol{v}_j]$ as the set of presence intervals from which separating $(k-1)$-tuples can be formed.

Theorem 5. *Algorithm $\mathcal{A}_k$ solves k-*RESTRICTEDMAXTOTAL *in AM1 in $O(n^{k^2+k-1})$ time and $O(n^k)$ space.*

It is natural to ask whether it is possible to extend the dynamic program described above to the models AM2 and AM3. With some modifications and at the expense of another polynomial factor in the running time we can extend algorithm $\mathcal{A}_k$ to the activity model AM2. The important difference between AM1 and AM2 is that presence intervals can be truncated at their right side if there is an active conflicting witness label causing the truncation. Hence, we need to add for each presence interval, all possible subintervals to Ψ that might be contained in an optimal solution. Moreover special care needs to be taken to ensure the witness condition of AM2 for all truncated intervals. A more detailed discussion of this extension can be found in the full version [7].

Theorem 6. *k-RESTRICTEDMAXTOTAL in AM2 can be solved in polynomial time.*

It remains open whether k-RESTRICTEDMAXTOTAL can be solved in polynomial time in AM3. Another extension of the dynamic programming algorithm is unlikely, since in AM3 the left and right subinstances created by a separating k-tuple $\boldsymbol{v}$ may have dependencies and thus cannot be solved independently any more. This is because a single original presence interval I can have subintervals both in $L_k[\boldsymbol{v}]$ and $R_k[\boldsymbol{v}]$, which cannot simultaneously be active.

Since the running time of our algorithms are, even for small k, prohibitively expensive in practice, we propose an approximation algorithm for k-RESTRICTEDMAXTOTAL based on GREEDYMAXTOTAL, which can be found in the full version [7].

Theorem 7. *There exists an $O(n^2)$-time approximation algorithm for for k-RESTRICTEDMAXTOTAL with unit squares for AM1–AM3 with approximation ratios $1/\min\{3+3k, 27\}$, $1/\min\{3+2k, 19\}$, $1/\min\{3+k, 11\}$, respectively.*

Acknowledgment. This work was supported by the 'Concept for the Future' of KIT under grant YIG 10-209 and by a Google Research Award.

References

1. P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory & Appl.*, 11(3-4):209–218, 1998.
2. K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Trans. Visualization and Computer Graphics*, 12(5):773–780, 2006.
3. K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Comput. Geom. Theory & Appl.*, 43(3):312–328, 2010.
4. M. C. Carlisle and E. L. Lloyd. On the k-coloring of intervals. *Discr. Appl. Math.*, 59(3):225–235, 1995.
5. P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *ACM-SIAM Symp. Discr. Algorithms (SODA'09)*, pp. 892–901, 2009.
6. R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.
7. A. Gemsa, B. Niedermann, and M. Nöllenburg. Trajectory-based dynamic map labeling. *CoRR*, arXiv:1309.3963, 2013.
8. A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In F. Dehne, J. Iacono, and J.-R. Sack (eds.), *Algorithms & Data Structures (WADS'11)*, vol. 6844 of *LNCS*, pp. 451–462. Springer, 2011.
9. J. Y. Hsiao, C. Y. Tang, and R. S. Chang. An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. *Inform. Process. Lett.*, 43(5):229–235, 1992.
10. D. Marx. Efficient approximation schemes for geometric problems?. In G. Brodal and S. Leonardi (eds.), *European Symp. Algorithms (ESA'05)*, vol. 3669 of *LNCS*, pp. 448–459. Springer, 2005.
11. B. Niedermann. Consistent labeling of dynamic maps using smooth trajectories. Master's thesis, Karlsruhe Institute of Technology, June 2012.
12. M. Sester and C. Brenner. Continuous generalization for visualization on small mobile devices. In P. F. Fisher (ed.), *Spatial Data Handling (SDH'04)*, pp. 355–368. Springer, 2004.
13. F. Wagner and A. Wolff. A practical map labeling algorithm. *Comput. Geom. Theory Appl.*, 7:387–404, 1997.
14. F. Wagner, A. Wolff, V. Kapoor, and T. Strijk. Three rules suffice for good label placement. *Algorithmica*, 30:334–349, 2001.