Aggregating Time Partitions

Taneli Mielikäinen
tmielika@cs.helsinki.fi

Evimaria Terzi
terzi@cs.helsinki.fi

Panayiotis Tsaparas
tsaparas@cs.helsinki.fi

Basic Research Unit, Helsinki Institute for Information Technology
Department of Computer Science, University of Helsinki, Finland

ABSTRACT

Partitions of sequential data exist either per se or as a result of sequence segmentation algorithms. It is often the case that the same timeline is partitioned in many different ways. For example, different segmentation algorithms produce different partitions of the same underlying data points. In such cases, we are interested in producing an aggregate partition, i.e., a segmentation that agrees as much as possible with the input segmentations. Each partition is defined as a set of continuous non-overlapping segments of the timeline. We show that this problem can be solved optimally in polynomial time using dynamic programming. We also propose faster greedy heuristics that work well in practice. We experiment with our algorithms and we demonstrate their utility in clustering the behavior of mobile-phone users and combining the results of different segmentation algorithms on genomic sequences.

1. INTRODUCTION

Analyzing sequential data has received considerable attention in the data mining community. To that aim many algorithms for extracting different kinds of useful information and representation of sequential data have been proposed. For example, in time-series mining and analysis, a major trend is towards the invention of *segmentation algorithms*. These are algorithms that take as input a sequence of points in $\mathbb{R}^d$, and give as output a partition of the sequence into contiguous and non-overlapping pieces that are called *segments*. The idea is that the variation of the data points within each segment is as small as possible, while at the same time the variation of the data across different segments is as large as possible. The points in each segment can then be concisely summarized, producing a compact representation of the original sequence that compresses the data at hand, and reveals their underlying structure. The resulting representation depends obviously on the definition of the measure of variation. There are many different such measures, resulting in different variants of the basic segmentation problem. Numerous segmentation algorithms have appeared in the literature and they have proved useful in time-series mining [4, 16, 21, 34], ubiquitous computing [19] and genomic sequence analysis [14, 27, 32].

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

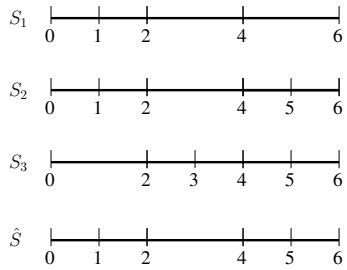

Figure 1: Segmentation aggregation that takes into consideration only the segment information.

The multitude of segmentation algorithms and variation measures raises naturally the question, given a specific dataset, what is the segmentation that best captures the underlying structure of the data? We try to answer this question by adopting a democratic approach that assumes that all segmentations found by different algorithms are correct, each one in its own way. That is, each one of them reveals just one aspect of the underlying true segmentation. Therefore, we aggregate the information hidden in the segmentations by constructing a consensus output that reconciles optimally the differences among the given inputs. We call the problem of finding such a segmentation, the *segmentation aggregation* problem.

The key idea of this paper lies in proposing a different view on sequence segmentation. We segment a sequence via aggregation of already existing, but probably contradicting segmentations. Therefore, the input to our problem is m different segmentations $S_1, \ldots, S_m$. The objective is to produce a single segmentation $\hat{S}$ that agrees as much as possible with the given m segmentations. We define a disagreement between two segmentations S and S' as a pair of points (x, y) such that S places them in the same segment, while S' places them in different segments, or vice versa. If $D_A(S, S')$ denotes the total number of disagreements between S and S', then the segmentation aggregation asks for segmentation $\hat{S}$ that minimizes $\sum_{i=1}^{m} D_A(S_i, \hat{S})$. Our definitions generalize to the continuous case, where each segmentation is a partition of a continuous timeline into segments. The discrete case can be mapped to the continuous by mapping points to elementary intervals of unit length.

As a concrete example, consider a sequence of length 6 and three segmentations of this sequence: S_1, S_2 and S_3 as shown in Figure 1. Each segmentation is defined by a set of boundaries. For example, segmentation S_1 has boundaries $\{0, 2, 4, 6\}$. A pair of boundaries i, and $i + 1$ defines a segment that contains all points in $(i, i + 1]$. For segmentation S_1 the first and second segments con-

tain only a single point (point 1 and 2 respectively), the third segment contains points 3 and 4, and the last segment contains points 5 and 6. The segmentation $\hat{S}$ in the bottom is the optimal aggregate segmentation for S_1, S_2 and S_3. The total cost of $\hat{S}$ is 3, since it has one disagreement with segmentation S_1 and two disagreements with segmentation S_3.

In this paper, we study the segmentation aggregation problem both theoretically and experimentally. Our contributions can be summarized as follows.

- We formally define the *segmentation aggregation* problem. Our definitions are general enough to include the partition of both discrete and continuous timelines. We consider various distance functions, and we study their properties.

- We show that the problem can be solved optimally in polynomial time using a dynamic-programming algorithm. We prove that the complexity of the algorithm depends on the number of unique segmentation points used by all segmentations, and not on the number of input points (the length of the timeline). We also propose greedy heuristics that although not exact, they are significantly faster, allowing us to deal efficiently with large datasets. Experimental evidence shows that the loss of accuracy due to those heuristics is negligible in most of the cases.

- We apply the segmentation aggregation framework to several problem domains and we demonstrate its practical significance. We experiment with haplotype data for producing a consensus haplotype block structure. We also experiment with real data of mobile phone users, and we use segmentation aggregation for clustering and profiling these users. Furthermore, we demonstrate that segmentation aggregation can be used to alleviate errors in segmentation algorithms that are caused due to errors or missing values in some of the dimensions of multidimensional time series.

In the next section we give some candidate application domains for segmentation aggregation. In Section 3 we formally define the segmentation aggregation problem and the disagreement distance between segmentations, and in Section 4 we describe exact and heuristic algorithms for solving it. Section 5 provides experimental evidence of the framework's utility. In Section 6 we give alternative formulations of the segmentation aggregation problem and in Section 7 we discuss the related work. We conclude the paper in Section 8.

2. APPLICATION DOMAINS

Segmentation aggregation can prove useful in many scenarios. We list some of them below.

Analysis of genomic sequences: A motivating problem of important practical value is the *haplotype block structure* problem. The "block structure" discovery in haplotypes is considered one of the most important discoveries for the search of structure in genomic sequences [7]. To explain this notion, consider a collection of DNA sequences over n marker sites for a population of individuals. Consider a marker site to be a location on the DNA sequence associated with some value. This value is indicative of the genetic variation of individuals in this location. The "haplotype block structure" hypothesis states that the sequence of markers can be segmented in blocks, so that, in each block most of the haplotypes of the population fall into a small number of classes. The description of these haplotypes can be used for further knowledge discovery, e.g., for associating specific blocks with specific genetic-influenced diseases [17].

From the computational point of view, the problem of discovering haplotype blocks in genetic sequences can be viewed as that of partitioning a multidimensional sequence into segments such that each segment demonstrates low diversity along the different dimensions. Different segmentation algorithms have been applied to good effect on this problem. However, these algorithms either assume different generative models for the haplotypes or optimize different criteria. As a result, they output block structures that are, to some extend (small or great) different. In this setting, the segmentation aggregation assumes that all models and optimization criteria contain useful information about the underlying haplotype structure, and aggregates their results to obtain a single block structure that is hopefully a better representation of the underlying truth.

Segmentation of multidimensional categorical data: The segmentation aggregation framework gives a natural way of segmenting multidimensional categorical data. Although the problem of segmenting multidimensional numerical data is rather natural, the segmentation problem of multidimensional categorical sequences has not been considered widely, mainly because such data are not easy to handle. Consider an 1-dimensional sequence of points that take nominal values from a finite domain. In such data, we can naturally define a segment as consecutive points that take the same value. For example, the sequence *a a a b b b c c*, has 3 segments (*a a a*, *b b b* and *c c*). When the number of dimensions in such data increases the corresponding segmentation problems becomes more complicated, and it is not straightforward how to segment the sequence using conventional segmentation algorithms. Similar difficulties in using off-the-shelf segmentation algorithms appear when the multidimensional data exhibit a mix of nominal and numerical dimensions. However, each dimension has its own clear segmental structure. We propose to segment each dimension individually, and aggregate the results.

Robust segmentation results: Segmentation aggregation provides a concrete methodology for improving segmentation robustness by combining the results of different segmentation algorithms, which may use different criteria for the segmentation, or different initializations of the segmentation method. Note also that most of the segmentation algorithms are sensitive to erroneous or noisy data. Such data though are very common in practice. For example, sensors reporting measurements over time may fail (e.g., run out of battery), genomic data may have missing values (e.g., due to insufficient wet-lab experiments). Traditional segmentation algorithms show little robustness to such scenarios. However, when their results are combined, via aggregation, the effect of missing or faulty data in the final segmentation is expected to be alleviated.

Clustering segmentations: Segmentation aggregation gives a natural way to cluster segmentations. In such a clustering, each cluster is represented by the aggregate segmentation, in the same way that the mean represents a set of points. The cost of the clustering is the sum of the aggregation costs within each cluster. Commonly used algorithms such as k-means can be adapted in this setting. Furthermore, the disagreements distance is a metric. Hence, we can apply various distance-based data-mining techniques to segmentations, and provide approximation guarantees for many of them.

Summarization of event sequences: An important line of research has focused on mining event sequences [1, 18, 22, 29]. An event sequence consists of a set of events of certain type that occur at certain points on a given timeline. For example, consider a user accessing a database at time points $t_1, t_2, \ldots, t_k$ within a day. Or

a mobile phone user making phone calls, or transferring between different cells. Having the activity times of the specific user for a number of different days one could raise the question: *How does the user's activity on an average day look like?* One can consider the time points at which events occur as segment boundaries. In that way, forming the profile of the user's daily activity is mapped naturally to a segmentation aggregation problem.

Privacy-preserving segmentations: Consider the scenario where there are multiple parties, each having a sequence defined over the *same* timeline. The parties would like to find a joint segmentation of the timeline, but they are not willing to share their sequences. (Alternatively, each party might have a segmentation method that is too sensitive to be shared.) A privacy-preserving segmentation protocol for such a scenario is as follows: each party computes a segmentation of their sequence locally and sends the segmentation to one party that aggregates the local segmentations.

3. THE SEGMENTATION AGGREGATION PROBLEM

3.1 Problem Definition

Let T be a timeline of bounded length. In order to make our definitions as general as possible, we consider the continuous case. We will assume that T is the real unit interval $(0, 1]$. For the purpose of exposition we will some times talk about discrete timelines. A discrete timeline T of size N can be thought of as the unit interval discretized into N intervals of equal length.

A segmentation P is a partition of T into continuous intervals (segments). Formally, we define $P = \{p_0, p_1, \ldots, p_\ell\}$, where $p_i \in T$ are the *breakpoints* (or *boundaries*) of the segmentation and it holds that $p_i < p_{i+1}$ for all i's. We will always assume that $p_0 = 0$ and $p_\ell = 1$. We define the i-th segment $\bar{p}_i$ of P to be the interval $\bar{p}_i = (p_{i-1}, p_i]$. The length of P, defined as $|P| = \ell$, is the number of segments in P. Note that there is an one to one mapping between boundaries and segments. We will often abuse the notation and define a segmentation as a set of segments instead of a set of boundaries. In these cases we will always assume that the segments define a partition of the timeline, and thus they uniquely define a set of boundaries.

For a set of m segmentations $P_1, \ldots, P_m$ we define their *union segmentation* to be the segmentation with boundaries $U = \bigcup_{i=1}^{m} P_i$. Let $\mathcal{S}$ be the space of all possible segmentations, and let D be a distance function between two segmentations P and Q, with $D : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$. Assume that function D captures how differently two segmentations partition timeline T. Given such a distance function, we define the SEGMENTATION AGGREGATION problem as follows:

PROBLEM 1 (SEGMENTATION AGGREGATION). *Given a set of m segmentations $P_1, P_2, ..., P_m$ of timeline T, and a distance function D between them, find a segmentation $\hat{S} \in \mathcal{S}$ that minimizes the sum of the distances from all the input segmentations. That is,*

$$\hat{S} = \arg\min_{S \in \mathcal{S}} \sum_{i=1}^{m} D(S, P_m).$$

We define $C(\hat{S}) = \sum_{i=1}^{m} D(S, P_m)$ to be the cost *of the aggregate segmentation.*

Note that Problem 1 is defined independently of the distance function D used between segmentations. We focus our attention on the disagreement distance D_A, which we formally describe in the next subsection. Other natural alternative distance functions are discussed in Section 6. The results we prove for D_A hold for those alternatives as well.

3.2 The Disagreement distance

In this section we formally define the notion of distance between two segmentations. Our distance function is based on similar distance functions proposed for clustering [15] and ranking [8]. The intuition for the distance function is drawn from the discrete case. Given two discrete timeline segmentations, the disagreement distance is the total number of pairs of points that are placed into the same segment in one segmentation, while placed in different segments in the other. We now generalize the definition to the continuous case.

Let $P = \{p_1, \ldots, p_{\ell_p}\}$ and $Q = \{q_1, \ldots, q_{\ell_q}\}$ be two segmentations. Let $U = P \cup Q$ be their union segmentation with segments $\{\bar{u}_1, \ldots, \bar{u}_n\}$. Note that by definition of the union segmentation, for every $\bar{u}_i$ there exist segments $\bar{p}_k$ and $\bar{q}_t$ such that $\bar{u}_i \subseteq \bar{p}_k$ and $\bar{u}_i \subseteq \bar{q}_t$. We define $P(\bar{u}_i) = k$ and $Q(\bar{u}_i) = t$, to be the *labeling* of interval $\bar{u}_i$ with respect to segmentations P and Q respectively. Similar to the discrete case, we define a disagreement when two segments $\bar{u}_i$, and $\bar{u}_j$ receive the same label in one segmentation, but different in the other. The disagreement is weighted by the product of the segment lengths $|\bar{u}_i||\bar{u}_j|$. Intuitively, the length captures the number of points contained in the interval, and the product the number of disagreements between the points. This notion can be made formal using integrals, but we omit the technical details. In the discrete case points can be thought of as unit intervals.

Formally, the disagreement distance of P and Q on segments $\bar{u}_i, \bar{u}_j \in U$ is defined as follows.

$$d_{P,Q}(\bar{u}_i, \bar{u}_j) := \begin{cases} |\bar{u}_i||\bar{u}_j|, & \text{if } P(\bar{u}_i) = P(\bar{u}_j) \text{ and } Q(\bar{u}_i) \neq Q(\bar{u}_j) \\ & \text{or } Q(\bar{u}_i) = Q(\bar{u}_j) \text{ and } P(\bar{u}_i) \neq P(\bar{u}_j) \\ 0, & \text{otherwise.} \end{cases}$$

Naturally, the overall disagreement distance between two segmentations is defined as follows.

$$D_A(P, Q) = \sum_{(\bar{u}_i, \bar{u}_j) \in U \times U} d_{P,Q}(\bar{u}_i, \bar{u}_j)$$

It is rather easy to prove that the distance function D_A is a metric. This property is significant for applications such as clustering, where good worst-case approximation bounds can be derived in metric spaces.

3.3 Computing disagreement distance

For two segmentations P and Q with ℓ_p and ℓ_q number of segments respectively, the distance $D_A(P, Q)$ can be computed trivially in time $O\left((\ell_p + \ell_q)^2\right)$. Next we show that this can be done even faster in time $O\left(\ell_p + \ell_q\right)$. Furthermore, our analysis helps in building intuition on the general aggregation problem. This intuition will be useful in the following sections.

We first define the notion of *potential energy*.

DEFINITION 1. *Let $\bar{v} \subseteq T$ be an interval in timeline T that has length $|\bar{v}|$. We define the* potential energy *of the interval to be:*

$$E(\bar{v}) = \frac{|\bar{v}|^2}{2}. \quad (1)$$

Let $P = \{p_0, \ldots, p_\ell\}$ be a segmentation with segments $\{\bar{p}_1, \ldots, \bar{p}_\ell\}$. We define the potential energy *of P to be*

$$E(P) = \sum_{\bar{p}_i \in P} E(\bar{p}_i)\ .$$

The potential energy computes the potential disagreements that the interval $\bar{v}$ can create. To better understand the intuition behind it we resort again to the discrete case. Let $\bar{v}$ be an interval in the discrete time line, and let $|\bar{v}|$ be the number of points in $\bar{v}$. There are $|\bar{v}|(|\bar{v}|-1)/2$ distinct pairs of points in $\bar{v}$, all of which can potentially cause disagreements with other segmentations.

Each of the discrete points in the interval can be thought of as a unit-length elementary subinterval and there are $|\bar{v}|(|\bar{v}|-1)/2$ pairs of those in $\bar{v}$, all of which are potential disagreements. Considering the continuous case is actually equivalent to focusing on very small (instead of unit length) subintervals. Let their length be ε with $\varepsilon << 1$. In this case the potential disagreements caused by all the ε-length intervals in $\bar{v}$ are:

$$\frac{|\bar{v}|/\varepsilon\,(|\bar{v}|/\varepsilon - 1)}{2} \cdot \varepsilon^2 \quad = \quad \frac{|\bar{v}|^2 - \varepsilon|\bar{v}|}{2} \rightarrow \frac{|\bar{v}|^2}{2}$$

when $\varepsilon \rightarrow 0$.[1]

The potential energy of a segmentation P is the sum of the potential energies of the segments it contains. Intuitively, it captures the number of potential disagreements due to points placed in the same segment in P, while in different segments in other segmentations. Given this definition we can show the following basic lemma.

LEMMA 1. *Let P and Q be two segmentations and U be their union segmentation. The distance $D_A(P,Q)$ can be computed by the following closed formula*

$$\begin{aligned} D_A(P,Q) &= E(P) - E(U) + E(Q) - E(U) \qquad (2)\\ &= E(P) + E(Q) - 2E(U). \end{aligned}$$

PROOF. For simplicity of exposition we will present the proof in the discrete case and talk in terms of points (rather than intervals), though the extension to intervals is straightforward. Consider the two segmentations P and Q and a pair of points $x, y \in T$. For some point x let $P(x)$, and $Q(x)$ be the index of the segment that contains x in P and Q respectively. By definition, the pair (x, y) introduces a disagreement if one of the following two cases is true:

Case 1: $P(x) = P(y)$ and $Q(x) \neq Q(y)$,

Case 2: $Q(x) = Q(y)$ and $P(x) \neq P(y)$.

In Equation 2, we can see that the term $E(P)$ gives all the pairs of points that are in the same segments in segmentation P. Similarly, the term $E(U)$ gives the pairs of points that are in the same segments in the union segmentation U. Their difference gives the number of pairs that are in the same segment in P but not in the same segment in U. However, if for two points x, y it holds that $P(x) = P(y)$ and $U(x) \neq U(y)$, then it has to be that $Q(x) \neq Q(y)$, since U is the union segmentation of P and Q. Therefore, the potential difference $E(P) - E(U)$ counts all the disagreements due to Case 1. Similarly, the disagreements due to Case 2 are counted by the term $E(Q) - E(U)$. Therefore, Equation 2 gives the total number of disagreements between segmentations P and Q. □

Lemma 1 allows us to compute the disagreements between two segmentations P and Q of size ℓ_p and ℓ_q respectively in time $O(\ell_p + \ell_q)$.

4. AGGREGATION ALGORITHMS

In this section we give optimal and heuristic algorithms for the SEGMENTATION AGGREGATION problem. First, we show that the optimal segmentation aggregation contains only segment boundaries in the union segmentation. That is, no new boundaries are introduced in the aggregate segmentation. Based on this observation we can construct a dynamic-programming algorithm (DP) that solves the problem exactly even in the continuous setting. If n is the size of the union segmentation, the dynamic-programming algorithm runs in time $O(n^2 m)$. We also propose faster greedy heuristic algorithms that run in time $O\left(n(m + \log n)\right)$ and, as shown in the experimental section, give high-quality results in practice.

4.1 Candidate segment boundaries

Let U be the union segmentation of the segmentations $S_1, \ldots, S_m$. The following theorem establishes the fact that the boundaries of the optimal aggregation are a subset of the boundaries appearing in U. The proof of the theorem appears in the full version of the paper.

THEOREM 1. *Let $S_1, S_2 \ldots S_m$ be the m input segmentations to the segmentation aggregation problem for the D_A distance, and let U be their union segmentation. For the optimal aggregate segmentation $\hat{S}$, it holds that $\hat{S} \subseteq U$, that is, all the segment boundaries in $\hat{S}$ belong in U.*

The consequences of the theorem are twofold. For the discrete version of the problem, where the input segmentations are defined over discrete sequences of N points, Theorem 1 restricts the search space of output aggregations. That is, only 2^n (instead of 2^N) segmentations are valid candidate aggregations. Furthermore, this pruning of the search space allows us to map the continuous version of the problem to a discrete combinatorial search problem, and to apply standard algorithmic techniques for solving it.

4.2 The DP algorithm

We now formulate the dynamic-programming algorithm that solves optimally the segmentation aggregation problem. We first need to introduce some notation. Let $S_1, \ldots, S_m$ be the input segmentations, and let $U = \{u_1, \ldots, u_n\}$ be the union segmentation. Consider a *candidate* aggregate segmentation $A \subseteq U$, and let $C(A)$ denote the cost of A, that is, the sum of distances of A to all input segmentations. We write $C(A) = \sum_i C_i(A)$, where $C_i(A) = D_A(A, S_i)$, the distance between A and segmentation S_i. The optimal aggregate segmentation is the segmentation $\hat{S}$ that minimizes the cost $C(\hat{S})$.

We also define a *j-restricted* segmentation A^j to be a candidate segmentation such that the next-to-last breakpoint is restricted to be the point $u_j \in U$. That is, the segmentation is of the form $A^j = \{a_0, \ldots, a_{\ell-1}, a_\ell\}$, where $a_{\ell-1} = u_j$. Segmentation A^j contains u_j, and does not contain any breakpoint $u_k > u_j$, except for the last point of the sequence. To avoid confusion, we note that although a j-restricted segmentation is restricted to select boundaries from the first j boundaries of U, it does not necessarily have length $j+1$, but rather, any length $\ell \leq j+1$ is possible. We use $\mathcal{A}^j$ to denote the set of all j-restricted segmentations, and $\hat{S}^j$ to denote the one with the minimum cost. Note that for $j = 0$, $\hat{S}^0 = \{u_0, u_n\}$ consists of a single segment. Abusing slightly the notation, for $j = n$, where the next-to-last and the last segmentation breakpoints coincide to be u_n, we have that $\hat{S}^n = \hat{S}$, that is, the optimal aggregate segmentation.

Let A be a candidate segmentation, and let $u_k \in U$ be a boundary point such that $u_k \notin A$. We define the *impact* of u_k to A to be the change (increase or decrease) in the cost that is caused by adding breakpoint u_k to the A, that is, $I(A, u_k) = C(A \cup \{u_k\}) - C(A)$. We have that $I(A, u_k) = \sum_i I_i(A, u_k)$, where $I_i(A, u_j) = C_i(A \cup \{u_k\}) - C_i(A)$.

[1] We can obtain the same result by integration, however, we feel that this helps to better understand the intuition of the definition.

We can now prove the following theorem.

THEOREM 2. *The cost of the optimal solution for the* SEGMENTATION AGGREGATION *problem can be computed using a dynamic-programming algorithm (*DP*) with the following recursion.*

$$C(\hat{S}^j) = \min_{0 \le k < j} \left\{ C(\hat{S}^k) + I(\hat{S}^k, u_j) \right\} \quad (3)$$

PROOF. For the proof of correctness it suffices to show that the impact of adding breakpoint u_j to a k-restricted segmentation is the same for all $A^k \in \mathcal{A}^k$. Recursion 3 calculates the minimum-cost aggregation correctly, since the two terms appearing in the summation are independent.

Formally, for some sequence $Q = \{q_0, \ldots, q_\ell\}$, and some boundary value b, let $\mathrm{Pre}(Q, b) = \{q_j \in Q : q_j < b\}$ be the set of breakpoints in Q that precede point b. Consider a k-restricted segmentation $A^k \in \mathcal{A}^k$ with boundaries $\{a_0, \ldots, a_{\ell-1}, a_\ell\} \subseteq U$. We will prove that the impact $I(A^k, u_j)$ is independent of the set $\mathrm{Pre}(A^k, u_k)$. Since the $a_{\ell-1} = u_k$ for all segmentations in $\mathcal{A}^k$, we have that $I(A^k, u_j)$ is invariant in $\mathcal{A}^k$.

For proving the above claim it is enough to show that $I_i(A^k, u_j)$ is independent of $\mathrm{Pre}(A^k, u_k)$ for every input segmentation S_i. Let U_i^k be the union of segmentation S_i with the segmentation A^k. Using Lemma 1 we have that $C_i(A^k) = E(A^k) + E(S_i) - 2E(U_i^k)$. Therefore, we need to show that the change in the potential of A^k, S_i and U_i^k is independent of $\mathrm{Pre}(A^k, u_k)$.

Adding boundary point u_j to A^k has obviously no effect on the potential of S_i. In order to study the effect on the potential of A^k, and U_i^k we consider the general question of how the potential of a segmentation changes when adding a new breakpoint. Let $Q = \{q_0, \ldots, q_\ell\}$ be a segmentation, and let $Q' = Q \cup \{b\}$ denote the sequence Q after the addition of breakpoint b. Assume that b falls in segment $\bar{q}_t = (q_{t-1}, q_t]$. The addition of b splits the interval $\bar{q}_t$ into two segments $\bar{\beta}_1 = (q_{t-1}, b]$ and $\bar{\beta}_2 = (b, q_t]$ such that $|\bar{q}_t| = |\bar{\beta}_1| + |\bar{\beta}_2|$. We can think of Q' as being created by adding segments $\bar{\beta}_1$, and $\bar{\beta}_2$ and removing segment $\bar{q}_t$. Since the potential of a segmentation is the sum of the potentials of its intervals, we have that

$$\begin{aligned} E(Q') - E(Q) &= -E(\bar{q}_t) + E(\bar{\beta}_1) + E(\bar{\beta}_2) \\ &= -\frac{|\bar{q}_t|^2}{2} + \frac{|\bar{\beta}_1|^2}{2} + \frac{|\bar{\beta}_2|^2}{2} \\ &= -|\bar{\beta}_1||\bar{\beta}_2|\,. \end{aligned} \quad (4)$$

Consider now the segmentation A^k. Adding u_j to segmentation A^k splits the last segment $\bar{a}_\ell$ into two sub-segments. From Equation 4 we know that the change in potential of A^k depends only on the lengths of these sub-segments. Since these lengths are determined solely by the position of the boundary $a_{\ell-1}$, the potential change is independent of $\mathrm{Pre}(A^k, u_k)$.

For segmentation U_i^k, we need to consider two cases. If $u_j \in S_i$, then the addition of u_j to A^k does not change U_i^k, since the boundary point was already in the union. Therefore, there is no change in potential. If $u_j \notin S_i$, then we need to add breakpoint u_j to segmentation U_i^k. Assume that the breakpoint u_j falls in $\bar{u}_t = (u_{t-1}, u_t]$. The change in the potential of U^k depends only on the lengths of sub-intervals into which the segment $\bar{u}_t$ is split. However, since $u_j > u_k$, and since we know that $u_k \in U_i^k$, we have that $u_{t-1} \ge u_k$. Therefore, the change in potential is independent of $\mathrm{Pre}(U_i^k, u_k)$, and hence independent of $\mathrm{Pre}(A^k, u_k)$ □

Computing the impact of every point can be done in $O(m)$ time (constant time is needed for each S_i) and therefore the total computation needed for the evaluation of the dynamic-programming recursion is $O(n^2 m)$.

4.3 The GREEDY algorithm

The dynamic-programming algorithm produces an optimal solution with respect to the disagreements distance, but it runs in time quadratic in the size n of the union segmentation. This makes it impractical for large datasets. We therefore need to consider faster heuristics.

In this section we describe a greedy bottom-up (GREEDYBU) approach to segmentation aggregation. (The idea in the top-down greedy algorithm GREEDYTD is similar but the description is omitted due to lack of space.) The algorithm starts with the union segmentation U. Let $A_1 = U$ denote this initial aggregate segmentation. At the t-th step of the algorithm we identify the boundary b in A_t whose removal causes the maximum decrease in the cost of the segmentation. By removing b we obtain the next aggregate segmentation A_{t+1}. If no boundary that causes cost reduction exists, the algorithm stops and it outputs the segmentation A_t.

At some step t of the algorithm, let $C(A_t)$ denote the cost of the aggregate segmentation A_t constructed so far. As in Section 4.2, we have that $C(A_t) = \sum_i C_i(A_t)$. For each boundary point $b \in A_t$, we need to store the *impact* of removing b from A_t, that is, the change in $C(A_t)$. This may be negative, meaning that the cost decreases, or positive, meaning that the cost increases. We denote this impact by $I(b)$ and as before, it can be written as $I(b) = \sum_i I_i(b)$.

We will now show how to compute and maintain the impact in an efficient manner. We will show that at any step the impact for a boundary point b can be computed by looking only at *local* information: the segments adjacent to b. Furthermore, the removal of b affects the impact only of the adjacent boundaries in A_t, thus updates are also fast.

For the computation of $I(b)$ we make use of Lemma 1. Let A_t be the aggregate segmentation at step t, and let S_i denote one of the input segmentations. Also, let U_i denote the union segments between A_t and S_i. We have that $C_i(A_t) = E(S_i) + E(A_t) - 2E(U_i)$. Similar to Section 4.2, we can compute the impact of removing boundary b by computing the change in potential. The potential of S_i remains obviously unaffected. We only need to consider the effect of b on the potentials $E(A_t)$ and $E(U_i)$.

Assume that $b = a_j$ is the j-th boundary point of A_t. Removing a_j causes segments $\bar{a}_j$ and $\bar{a}_{j+1}$ to be merged, creating a new segment of size $|\bar{a}_j| + |\bar{a}_{j+1}|$ and removing two segments of size $|\bar{a}_j|$ and $|\bar{a}_{j+1}|$. Therefore, the potential energy of the resulting segmentation A_{t+1} is

$$\begin{aligned} E(A_{t+1}) &= E(A_t) + \frac{(|\bar{a}_j| + |\bar{a}_{j+1}|)^2}{2} - \frac{|\bar{a}_j|^2}{2} - \frac{|\bar{a}_{j+1}|^2}{2} \\ &= E(A_t) + |\bar{a}_j||\bar{a}_{j+1}| \end{aligned}$$

The boundary b that is removed from A_t is also a boundary point of U_i. If $b \in S_i$, then the boundary remains in U_i even after it is removed from A_t; thus, the potential energy $E(U_i)$ does not change. Therefore, the impact is $I_i(b) = |\bar{a}_j||\bar{a}_{j+1}|$

Consider now the case that $b \notin S_i$. Assume that $b = u_k$ is the k-th boundary of U. Therefore, it separates the segments $\bar{u}_k$ and $\bar{u}_{k+1}$. The potential energy of U_i increases by $|\bar{u}_k||\bar{u}_{k+1}|$. Thus the impact of b is $I_i(b) = |\bar{a}_j||\bar{a}_{j+1}| - 2|\bar{u}_k||\bar{u}_{k+1}|$.

Therefore, the computation of $I_i(b)$ can be done in constant time with the appropriate data structure for obtaining the lengths of the segments adjacent to b. Going through all input segmentations we can compute $I(b)$ in time $O(m)$.

Computing the impact of all boundary points takes time $O(nm)$.

Updating the costs in a naive way would result in an algorithm with cost $O(n^2 m)$. However, we do not need to update all boundary points. Since the impact of a boundary point depends only on the adjacent segments, only the impact values of the neighboring boundary points are affected. If $b = a_j$, we only need to recompute the impact for a_{j-1} and a_{j+1}, which can be done in time $O(m)$.

Therefore, using a simple heap to store the benefits of the breakpoints, we are able to compute the aggregate segmentation in time $O(n(m + \log n))$.

5. EXPERIMENTS

In this section we experimentally evaluate our methodology. First, on a set of generated data we show that both DP and GREEDY algorithms give results of high quality. Next, we show the usefulness of our methodology in different domains.

5.1 Comparing aggregation algorithms

For this experiment we generate segmentation datasets as follows. First we create a random segmentation of a sequence of length 1000 by picking a random set of boundary points. Then, we use this segmentation as a basis to create a dataset of 100 segmentations to be aggregated. Each segmentation is generated from the basis as follows: each segment boundary of the basis is kept identical in the output segmentation with probability $(1 - p)$, or it is altered with probability p. There are two types of changes a boundary is subject to: *deletion* and *translocation*. In the case of translocation, the new location of the boundary is determined by the *variance level* (v). For small values of v the boundary is placed close to its old location, while for large values it is placed further.

Figure 2 shows the ratio of the aggregation costs achieved by GREEDYTD and GREEDYBU with respect to the optimal DP algorithm. It is apparent that in most of the cases the greedy alternatives give results with cost very close (almost identical) to the optimal. We mainly show the results for $p > 0.5$, since for smaller values of p the ratio is always equal to 1. Figure 3 shows the distance (measured using D_A) between the aggregation produced by DP, GREEDYTD and GREEDYBU and the basis segmentation used for generating the datasets. These results demonstrate that not only the quality of the aggregation found by the greedy algorithms is close to that of the optimal, but also that the *structure* of the algorithms' outputs is very similar. All the results are averages over 10 independent runs.

5.2 Experiments with haplotype data

The basic intuition of the haplotype-block problem as well as its significance in biological sciences and medicine have already been discussed in Section 2. Here we show how the segmentation-aggregation methodology can be applied in this setting. The main problem with the haplotype block-structure problem is that although numerous studies have confirmed its existence, the methodologies that have been proposed for finding the blocks are inconclusive with respect to the number and the exact positions of their boundaries.

The main line of work related to haplotype-block discovery consists of a series of segmentation algorithms. These algorithms usually assume different optimization criteria for block quality and segment the data so that blocks of good quality are produced. Although one can argue for or against each one of those optimization functions, we again adopt the aggregation approach. That is, we aggregate the results of the different algorithms used for discovering haplotype blocks by doing segmentation aggregation.

For the experiments we use the published dataset of [7] and we aggregate the segmentations produced by the following five different methods:

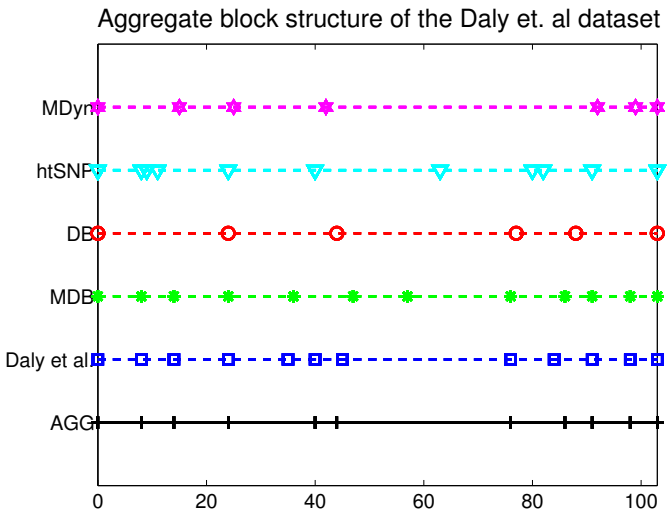

Figure 4: Block structure of real haplotype data

1. *Daly et al.*: This is the original algorithm for finding blocks used in [7].

2. *htSNP*: This is a dynamic-programming algorithm proposed in [35]. The objective function uses the htSNP criterion proposed in [30].

3. *DB*: This is again a dynamic-programming algorithm, though for a different optimization criterion. The algorithm is proposed in [35], while the optimization measure is the *haplotype-diversity* proposed by [20].

4. *MDB*: This is a Minimum Description Length (MDL) method proposed in [3].

5. *MDyn*: This is another MDL-based method proposed by [23].

Figure 4 shows the block boundaries found by each one of the methods. The solid line shows the block boundaries found by doing segmentation aggregation on the results of the aforementioned five methods. The aggregate segmentation has 11 segment boundaries, while the input segmentations have $12, 11, 6, 12$ and 7 segment boundaries respectively, with 29 of them being unique. Note that in the result of the aggregation, block boundaries that are very close to each other in some segmentation methods (for example htSNP) disappear and in most cases they are replaced by a single boundary. Additionally, the algorithm does not always find boundaries that are in the majority of the input segmentations. For example, the eighth boundary of the aggregation appears in only two input segmentations, namely the results of Daly et al. and htSNP.

5.3 Robustness experiments

In this experiment we demonstrate the usefulness of the segmentation aggregation in producing robust segmentation results, insensitive to the existence of outliers in the data. Consider the following scenario, where multiple sensors are sending their measurements to a central server. It can be the case that some of the sensors may fail at certain points in time. For example, they may run out of battery or report erroneous values due to communication delays in the network. Such a scenario causes outliers (missing or erroneous data) to appear. The classical segmentation algorithms are sensitive to such values and usually produce "unintuitive" results. We here

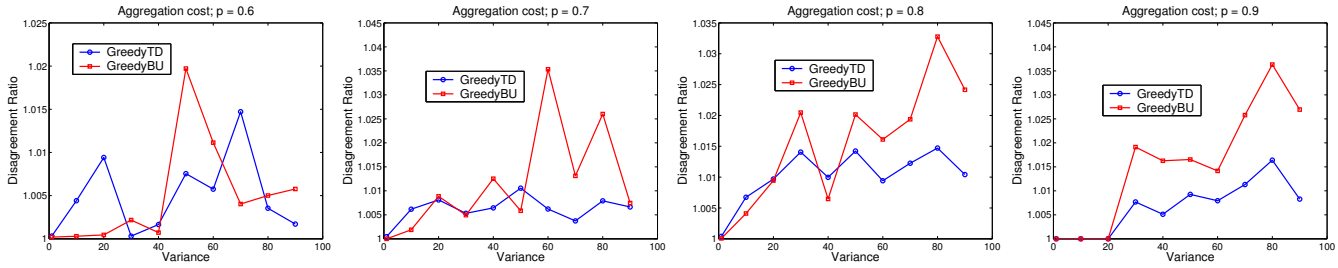

Figure 2: Performance ratio of greedy algorithms with respect to the optimal DP result.

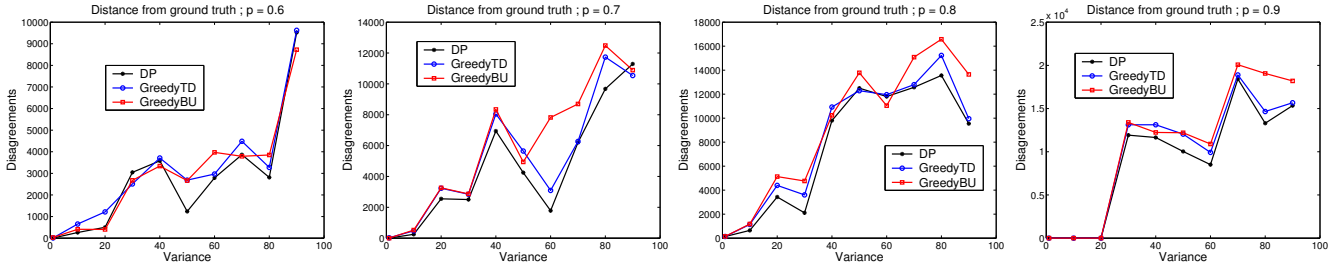

Figure 3: Disagreements of the aggregate segmentation with the ground truth segmentation, used for generating the data.

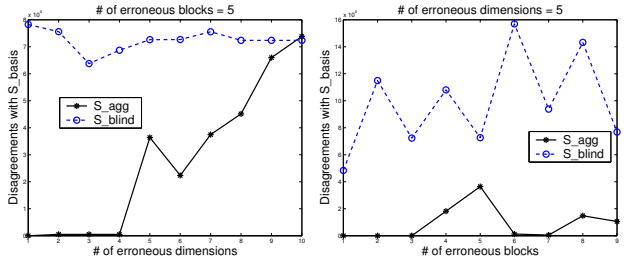

Figure 5: Disagreements of S_{agg} and S_{blind} with the true underlying segmentation S_{basis}.

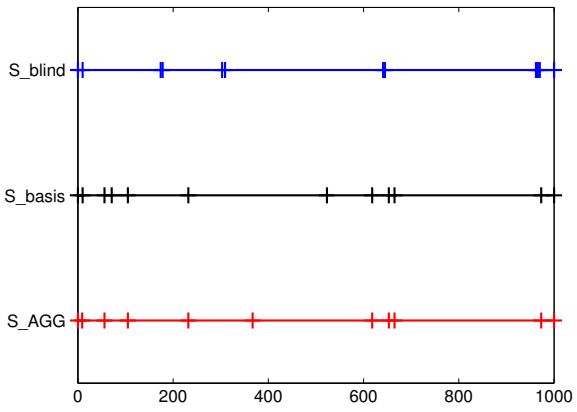

Figure 6: Anecdote illustrative of the insensitivity of the aggregation to the existence of outliers in the data.

show that the segmentation aggregation is insensitive to the existence of missing or erroneous data via the following experiment. First, we generate a multidimensional sequence of real numbers that has an a-priori known segmental structure. We fix the number of segments appearing in the data to be $k = 10$, and all the dimensions have the same segment boundaries. All the points in a segment are normally distributed around some randomly picked mean $\mu \in [9, 11]$. One can consider each dimension to correspond to data coming from a different sensor. We report the results from a dataset that has 1000 data points, and 10 dimensions.

Standard segmentation methods segment all dimensions together. We do the same using the variance of the segments to measure the quality of the segmentation. We segment all the dimensions together using the standard optimal dynamic-programming algorithm for sequence segmentation [4]. We denote by S_{basis} the segmentation of this data obtained by this dynamic-programming algorithm.

Then, we simulate the erroneous data as follows: first we pick a specific subset of dimensions on which we insert erroneous blocks of data. The cardinality of the subset varies from 1 to 10 (all dimensions). An erroneous block is a set of consecutive outlier values. Outlier values are represented by 0s in this example. We use small blocks of length at most 4 and we insert $1 - 10$ such blocks. This means that in the worst case we have at most 4% faulty data points.

We segment this noisy dataset using the standard segmentation algorithm described above that blindly segments all dimensions together. We denote by S_{blind} the segmentation produced by the dynamic-programming segmentation algorithm on this modified dataset. We also experiment with the aggregation approach. We segment each dimension separately in $k = 10$ segments and then we aggregate the results. We denote by S_{agg} the resulting aggregate segmentation.

Figure 5 reports the disagreements $D_A(S_{\text{agg}}, S_{\text{basis}})$ and $D_A(S_{\text{blind}}, S_{\text{basis}})$ obtained when we fix the number of erroneous blocks inserted in each dimension and vary the number of dimensions that are faulty, and vice versa. That is, we try to compare the number of disagreements between the segmentations produced by aggrega-

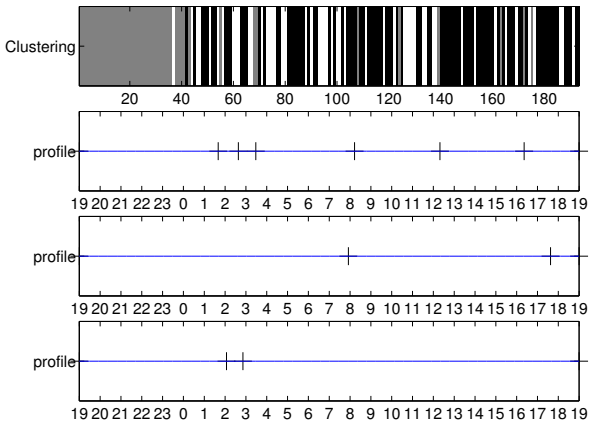

Figure 7: Clustering of a single user's logged days into three clusters and the cluster representatives.

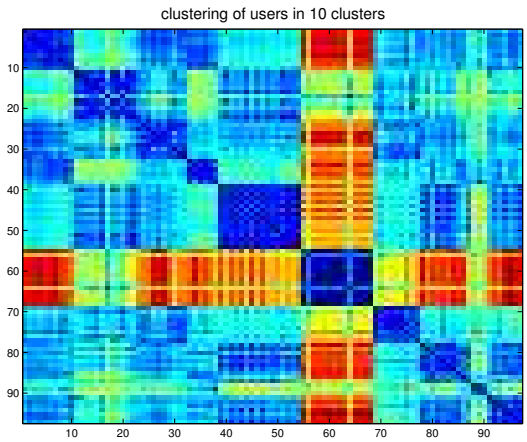

Figure 8: The clustering structure of the reality-mining user data

tion and by blindly segmenting all the dimensions together, to the segmentation that would have been obtained if the erroneous data were ignored. Our claim is that a "correct" segmentation should be as close as possible to S_{basis}. Figure 5 indeed demonstrates that the aggregation result is much closer to the underlying true segmentation, and thus the aggregation algorithm is less sensitive to the existence of outliers. Figure 6 further verifies this intuition by visualizing the segmentations S_{basis}, S_{agg} and S_{blind} for the case of 5 erroneous dimensions containing 5 blocks of consecutive outliers.

5.4 Experiments with reality-mining data

The reality-mining dataset[2] contains usage information of 97 mobile phone users [9]. A large percentage of these users are either students (masters students, freshmen, graduate students) or faculty (professors, staff) of the MIT Media Laboratory, while the rest are incoming students at the MIT Sloan business school, located adjacent to the laboratory. The collected information includes call logs, Bluetooth devices in proximity, cell tower IDs, application usage, and phone status (such as charging and idle) etc. The data spans a period from September 2004 to May 2005. We mainly focus our analysis on the data related to the *callspan* of each user. The callspan data has information related to the actual times each user places a phonecall.

From this data we produce segmentations as follows: for each user, and each day during which he has been logged, we take the starting times reported in the callspan and we consider them as segment boundaries on the timeline of the day. For example, a user that is logged for 30 days, produces 30 different segmentations, one for each day.

5.4.1 *Identifying single user's patterns*

In our first experiment, we cluster the days of a single user. Since each day is represented as a segmentation of the 24-hour timeline, clustering the days corresponds to clustering these segmentations. We use distance D_A for comparing the different days. The definition of segmentation aggregation allows naturally to define the "mean" of a segmentation cluster to be the aggregation of the segmentations that are grouped together in the cluster. We can then readily apply the classical k-means algorithm to the space of segmentations.

[2]The interested reader can find the datasets at `http://reality.media.mit.edu/`

Figure 7 shows the clustering of the days of a single user (who is classified as a professor in the dataset) over a period of 213 days starting from September 2004 to May 5th 2005 (not all days are recorded). The plot on the top shows the clustering of the days. The days are arranged sequentially and the different colors correspond to different clusters. It is apparent that at the beginning of the recorded period the patterns of the user are quite different from the patterns observed at later points in the study. More specifically, all the initial days form a single rather homogeneous cluster. From [9] we take the information that during this period the Media Lab subjects had been working towards the annual visit of the laboratory's sponsors. It had been previously observed that this had affected the subjects' schedules. It is possible that our methodology captures this pattern. The rest of Figure 7 shows the representatives of each cluster. We observe that the representatives are rather distinct consisting of profiles where the users use their phone either in morning hours, or in evening hours, or both.

5.4.2 *Finding groups of similar users*

In the second experiment we try to build clusters of users that show similar patterns in their activities. For this, we build the profile of each user, by aggregating all the days he has been logged for. Next, we cluster the user profiles, using the k-means algorithm for segmentations, as discussed in the previous paragraph. Figure 8 gives visual evidence of the existence of some clusters of users in the dataset. The plot shows the distances between user profiles, in terms of disagreement distance. The rows and the columns of the distance matrix have been rearranged so that users clustered together are put in consecutive rows (columns). The darker the coloring of a cell at position (i, j) the more similar users i and j are. There are some evident clusters in the dataset, like for example the one consisting of users at positions $1-10$, $33-38$, $39-54$, $55-68$ and $69-77$. Note that the cluster containing users $55-68$ is characterized not only by strong similarity between its members, but additionally a strong dissimilarity to almost every other user in the dataset.

From these groups, the third one, consisting of rows $39-54$, seems to be very coherent. We further looked at the people constituting this group and found out that most of them are related (being probably students) to the Sloan business school. More specifically, the academic/professional positions of the people in the cluster, as reported in the dataset, are as follows.

`sloan, mlUrop, sloan, sloan, sloan, sloan, 1styeargrad, sloan, 1styeargrad, mlgrad, sloan, sloan, sloan, staff, sloan, sloan`

Similarly, the relatively large and homogeneous group formed by lines $1-10$ consists mostly from staff and professors.

The cluster of users $55-68$, although quite homogeneous and distinct from the rest of the dataset, it contains a rather diverse set of people, at least with respect to their positions. However there may be another link that makes their phone-usage patterns similar, and separates them from the rest of the users.

6. ALTERNATIVE DISTANCE FUNCTIONS

In this section we discuss alternative formulations of the SEGMENTATION AGGREGATION problem. The differences are due to the alternative distance functions that can be used for comparing segmentations.

6.1 Entropy Distance

The *entropy distance* between two segmentations quantifies the information one segmentation reveals about the other. In general, the entropy distance between two random variables X and Y that take values in domains $\mathcal{X}$ and $\mathcal{Y}$ respectively is defined as

$$D_H(X,Y) \quad = \quad H(X \mid Y) + H(Y \mid X),$$

where $H(\cdot \mid \cdot)$ is the conditional entropy function and

$$H(X \mid Y) \quad = \quad -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} Pr[x,y]\log Pr[x \mid y]. \qquad (5)$$

For segmentations this can be applied as follows. Let $Q = \{q_0,\ldots,q_{\ell_q}\}$ be a segmentation that partions the unit real interval. Let i be the *label* of the interval $\bar{q}_i = (q_{i-1}, q_i]$ Abusing the notation, we define a random variable $Q : (0,1] \to \{1,\ldots,\ell_q\}$, where $Q(x) = i$ for $x \in \bar{q}_i$. Assuming that x is drawn uniformly at random, we have that $Pr[Q(x) = i] = |\bar{q}_i|$. In plain terms, each segment is chosen with probability proportional to its length. Given two segmentations Q and P we define the joint distribution of the random variables Q and P as $P[i,j] = |\bar{q}_i \cap \bar{p}_j|$, that is, the probability that a randomly chosen point falls in the intersection of the segments $\bar{q}_i$ and $\bar{p}_j$. We can now define the entropy distance $D_H(P,Q)$ between segmentations, using Equation 5.

The entropy distance is also a metric. Computing the entropy distance between two segmentations can be done in linear time. The main idea of this linear-time algorithm is a decomposition of $D_H(P,Q)$ similar to the decomposition of $D_A(P,Q)$ showed in Lemma 1 (Equation 2), using the concept of potential energy. For the entropy distance the potential energy of segmentation P is $E(P) = -H(P)$.

Furthermore, solving the SEGMENTATION AGGREGATION problem (Problem 1) for D_H can also be done optimally using a dynamic-programming algorithm. This algorithm is a variation of the DP algorithm discussed in Section 4. The recursion of the algorithm is again based on the fact that adding a new breakpoint has only a local effect on the cost of the segmentation. The details of the algorithms for the entropy distance are omitted due to lack of space.

6.2 Boundary Mover's Distance

The *Boundary Mover's Distance* (D_B)[3] compares two segmentations P and Q considering only the distances between their boundary points. Let the boundary points of P and Q be $\{p_0,\ldots,p_{\ell_p}\}$ and $\{q_0,\ldots,q_{\ell_q}\}$. We define the Boundary Mover's distance of P with respect to Q to be

$$D_B(P \mid Q) \quad = \quad \sum_{p_i\in P} \min_{q_j\in Q} |p_i - q_j|^r .$$

Two natural choices for r is $r = 1$ and $r = 2$. For $r = 1$ the Boundary Mover's distance employs the Manhattan distance between the segment boundaries, while for $r = 2$ it uses the sum-of-squares distance.

The SEGMENTATION AGGREGATION problem for distance D_B with m input segmentations $S_1,\ldots,S_m$ asks for an aggregate segmentation $\hat{S}$ of at most t boundaries such that:

$$\hat{S} \quad = \quad \arg\min_{S\in\mathcal{S}} D_B(S_i \mid S).$$

Note that in this alternative definition of the segmentation aggregation problem we have to restrict the number of boundaries that can appear in the aggregation. Otherwise, the optimal $\hat{S}$ will contain the union of boundaries that appear in P and Q – such a segmentation will have total cost equal to 0. One can easily see that this alternative definition of the segmentation aggregation problem can also be solved optimally in polynomial time. More specifically, the problem of finding the best aggregation with at most t segment boundaries is equivalent to one-dimensional clustering that can be solved using dynamic programming. For the mapping, consider the boundaries of the input segmentations to be the points to be clustered, and the boundaries of the aggregation to be the cluster representatives. We note that for the case $r = 1$ the boundaries of the aggregate segmentation are again a subset of the union segmentation.

7. RELATED WORK

Related work on segmentation algorithms and their practical utility has already been discussed in Section 1. Though these algorithms are related to our work, our goal here is not just to propose a new segmentation algorithm but to aggregate the results of existing algorithms.

There exists a considerable amount of work for building individual, system or network temporal profiles that can be used in anomaly or misuse detection, prediction of mobile-phone users etc., as for example in [11, 26, 24, 28]. These methods approach the problem from a different view point and segmentations are not a central concept in this approach. Though we explore the applicability of segmentation aggregation in clustering users and building user profiles, performing exhaustive experiments on profile-building is beyond the scope of this paper. Exploring the relationship of our method to other profiling methods is an interesting question for future work.

Most related to our work are other aggregation problems that have been extensively studied. The notion of aggregation has recently emerged in several data-mining tasks. The problem of aggregating clusterings has been studied under the names of clustering aggregation [15], consensus clustering [2, 25] and cluster ensembles [12, 33]. Ranking aggregation has been studied from the viewpoints of algorithmics [2], Web search [8], databases [10], and machine learning [13]. A third important group of aggregating data mining results is formed by voting classifiers such as bagging [5] and boosting [6].

Our work is similar in spirit, since we are also trying to aggregate results of existing data-mining algorithms. However, although the segmentation problem has received considerable attention by the data-mining community, to the best of our knowledge, the segmentation aggregation problem has not been previously studied.

[3]The name is inspired by the Earth Mover's Distance [31].

8. CONCLUDING REMARKS

We have presented a novel approach to sequence segmentation, that is based on the idea of aggregating existing segmentations. The utility of segmentation aggregation has been extensively discussed via a set of useful potential applications. We have formally defined the segmentation aggregation problem and showed some of its interesting properties. From the algorithmic point of view, we showed that we can solve it optimally in polynomial time using dynamic programming. Furthermore, we designed and experimented with greedy algorithms for the problem, which although not optimal, in practice they are both fast and give results of high quality (almost as good as the optimal). The practical utility of the problem and the proposed algorithms has been illustrated via a broad experimental evaluation that includes applications of the framework on genomic sequences and users' mobile-phone data. We additionally demonstrated that segmentation aggregation is a noise and error-insensitive segmentation method that can be used to provide trustworthy segmentation results.

Acknowledgements

We would like to thank Aris Gionis and Heikki Mannila for many useful discussions and suggestions.

9. REFERENCES

[1] AGRAWAL, R., AND SRIKANT, R. Mining sequential patterns. In *Proceedings of the IEEE International Conference on Data Engineering* (1995), pp. 3–14.

[2] AILON, N., CHARIKAR, M., AND NEWMAN, A. Aggregating inconsistent information: ranking and clustering. In *STOC* (2005), pp. 684–693.

[3] ANDERSON, E. C., AND NOVEMBRE, J. Finding haplotype block boundaries by using the minimum-description-length principle. *American Journal of Human Genetics 73* (2003).

[4] BELLMAN, R. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM 4*, 6 (1961).

[5] BREIMAN, L. Bagging predictors. *Machine Learning 24*, 2 (1996), 123–140.

[6] COLLINS, M., SCHAPIRE, R. E., AND SINGER, Y. Logistic regression, adaboost and bregman distances. *Machine Learning 48*, 1-3 (2002), 253–285.

[7] DALY, M. J., RIOUX, J. D., SCHAFFNER, S. F., HUDSON, T. J., AND LANDER, E. S. High-resolution haplotype structure in the human genome. *Nature Genetics 29* (2001), 229–232.

[8] DWORK, C., KUMAR, R., NAOR, M., AND SIVAKUMAR, D. Rank aggregation methods for the web. In *WWW* (2001), pp. 613–622.

[9] EAGLE, N. Machine perception and learning of complex social systems. *PhD Thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology* (2005).

[10] FAGIN, R., KUMAR, R., MAHDIAN, M., SIVAKUMAR, D., AND VEE, E. Comparing and aggregating rankings with ties. In *PODS* (2004).

[11] FAN, W., AND STOLFO, S. J. Ensemble-based adaptive intrusion detection. In *SDM* (2002).

[12] FRED, A. L., AND JAIN, A. K. Combining multiple clusterings unsing evidence accumulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 6 (2005), 835–850.

[13] FREUND, Y., IYER, R., SCHAPIRE, R. E., AND SINGER, Y. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research 4* (2003), 933–969.

[14] GIONIS, A., AND MANNILA, H. Finding recurrent sources in sequences. In *Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology* (Berlin, Germany, 2003), pp. 115–122.

[15] GIONIS, A., MANNILA, H., AND TSAPARAS, P. Clustering aggregation. In *Proceedings of the 21st International Conference on Research in Data Engineering* (Tokyo, Japan, 2005), pp. 51–60.

[16] GUHA, S., KOUDAS, N., AND SHIM, K. Data-streams and histograms. In *ACM Symposium on Theory of Computing* (2001), pp. 471–475.

[17] GUSFIELD, D. An overview of haplotyping via perfect phylogeny: Theory, algorithms and programs. In *ICTAI* (2003).

[18] GWADERA, R., ATALLAH, M. J., AND SZPANKOWSKI, W. Reliable detection of episodes in event sequences. *Knowledge and Information Systems 7*, 4 (2005), 415–437.

[19] HIMBERG, J., KORPIAHO, K., MANNILA, H., TIKANMÄKI, J., AND TOIVONEN, H. Time series segmentation for context recognition in mobile devices. In *Proceedings of the IEEE International Conference on Data Mining* (2001), pp. 203–210.

[20] JOHNSON, N., KOTZ, Z., AND BALAKRISHNAN, N. *Discrete multivariate distributions*. John Wiley & Sons, 1997.

[21] KEOGH, E., CHU, S., HART, D., AND PAZZANI, M. An online algorithm for segmenting time series. In *Proceedings of the IEEE International Conference on Data Mining* (2001), pp. 289–296.

[22] KLEINBERG, J. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery 7* (2003), 373–397.

[23] KOIVISTO, M., PEROLA, M., VARILO, T., ET AL. An MDL method for finding haplotype blocks and for estimating the strength of haplotype block boundaries. In *Pacific Symposium on Biocomputing* (2003), pp. 502–513.

[24] LANE, T., AND BRODLEY, C. E. Temporal sequence learning and data reduction for anomaly detection. *ACM Trans. Inf. Syst. Secur. 2*, 3 (1999), 295–331.

[25] LANGE, T., AND BUHMAN, J. M. Combining partitions by probabilistic label aggregation. In *KDD'05, August 21–24, 2005, Chicago, Illinois, USA* (2005), pp. 147–156.

[26] LEE, W., AND STOLFO, S. J. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur. 3*, 4 (2000), 227–261.

[27] LI, W. DNA segmentation as a model selection process. In *International Conference on Research in Computational Molecular Biology* (2001), pp. 204–210.

[28] LI, Y., WU, N., JAJODIA, S., AND WANG, X. S. Enhancing profiles for anomaly detection using time granularities. *Journal of Computer Security 10*, 1,2 (2002), 137–157.

[29] MANNILA, H., TOIVONEN, H., AND VERKAMO, I. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery 1*, 3 (1997), 259–289.

[30] PATIL, N., BERNO, A. J., HINDS, D. A., ET AL. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science 294* (2001), 1669–70.

[31] RUBNER, Y., TOMASI, C., AND GUIBAS, L. J. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision 40*, 2 (2000), 99–121.

[32] SALMENKIVI, M., KERE, J., AND MANNILA, H. Genome segmentation using piecewise constant intensity models and reversible jump MCMC. In *Proceedings of the European Conference on Computational Biology* (2002), pp. 211–218.

[33] STREHL, A., AND GHOSH, J. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research 3* (2002).

[34] TERZI, E., AND TSAPARAS, P. Efficient algorithms for sequence segmentation. In *SDM* (2006), p. To appear.

[35] ZHANG, K., CHENG, M., CHEN, T., WATERMAN, M., AND SUN, F. A dynamic programming algorithm for haplotype block partitioning. *PNAS 99* (2002), 7335–7339.