

An optimization approach for mapping and measuring the divergence and correspondence between paths

Shane T. Mueller · Brandon S. Perelman · Elizabeth S. Veinott

Published online: 4 March 2015
© Psychonomic Society, Inc. 2015

Abstract Many domains of empirical research produce or analyze spatial paths as a measure of behavior. Previously, approaches for measuring the similarity or deviation between two paths have either required timing information or have used ad hoc or manual coding schemes. In this paper, we describe an optimization approach for robustly measuring the area-based deviation between two paths we call ALCAMP (Algorithm for finding the Least-Cost Areal Mapping between Paths). ALCAMP measures the deviation between two paths and produces a mapping between corresponding points on the two paths. The method is robust to a number of aspects in real path data, such as crossovers, self-intersections, differences in path segmentation, and partial or incomplete paths. Unlike similar algorithms that produce distance metrics between trajectories (i.e., paths that include timing information), this algorithm uses only the order of observed path segments to determine the mapping. We describe the algorithm and show its results on a number of sample problems and data sets, and demonstrate its effectiveness for assessing human memory for paths. We also describe available software code written in the R statistical computing language that implements the algorithm to enable data analysis.

Keywords Path correspondence · Path mapping · Optimization

S. T. Mueller (✉) · B. S. Perelman
Department of Cognitive and Learning Sciences, Michigan Technological University, Houghton, MI, USA
e-mail: shanem@mtu.edu

E. S. Veinott
Applied Research Associates, Fairborn, OH, USA

Behavioral paths and trajectories

Paths are multi-dimensional spatial data series that represent an ordered sequence of locations in space. In contrast to trajectories (which refer to paths as a function of time), a path typically ignores or lacks timing information, making it both more general and at times less informative. Importantly, two trajectories that mismatch in their timing might be judged as very different, even if their paths are nearly identical. Nevertheless, both paths and trajectories have become important data sources in scientific research and machine intelligence applications, and their use is only bound to increase as research and applications take advantage of mobile GPS-enabled devices and other automated means of recording location. Computational research in this area has typically focused on trajectories (Yanagisawa et al., 2003; Chen et al., 2005) where timing is available, but in many cases the timing is either not known or is irrelevant, and so there remains a gap in algorithms and tools for analyzing paths and measuring path similarity and deviation based on their shapes alone.

Potential applications of a general-purpose path mapping technique

In this paper, our main objective is to present a set of algorithms and software tools that analyze paths produced in laboratory and real-world settings, allowing measurement of the deviation between paths, as well as determining how points on one path correspond to points on a second path. In behavioral science, these tools have potential applications in a number of domains, including spatial reasoning, memory, and communication. Specific examples include: the analysis of path data produced in studies using variations of

the HCRC Map task (Anderson et al., 1991; Brown et al., 1984; Veinott et al., 1999), in which one person must verbally communicate a path to a second person who cannot see the path, but must reproduce it; spatial search tasks, in which a person or animal must search for a target or reward (Mueller et al., 2010; 2013); the traveling salesman problem (MacGregor & Ormerod, 1996; Pizlo et al., 2006), in which an efficient path must be planned that visits a set of targets; eye-tracking research (van der Stigchel et al., 2006), in which the eye fixation pattern is recorded; mouse-tracking studies (Freeman & Ambady, 2010), in which a computer mouse path is recorded in a dynamic task; motor control research (Abend et al., 1982), in which the physical path of an effector is tracked; and human–computer interaction, for example in research that compares navigation through virtual and physical space (Ruddle & Lessels, 2009; Zhang et al., 2012).

Although applications in behavioral science are our primary concern, our methods may also have utility in any domain for which path data is available. Examples include: neuroscience research examining neural information pathways (Tuch et al., 2003); the domains of geography and anthropology which study the migration and movement of nomadic pastoral groups (Erdenebaatar & Humphrey, 1996; Istomin & Dwyer, 2009), especially to the extent that they compare simulated paths to observed data (Kennedy et al., 2010); zoology, in the study of migration paths of animals (Croxall et al., 2005; Egevang et al., 2010; Guilford et al., 2009; Weimerskirch & Wilson, 2000; Ceriani et al., 2012); and earth sciences including the measurement and assessment of how river routes (Fisk, 1944), ocean surface currents (Gawarkiewicz et al., 2012), atmospheric jet streams (Barton & Ellis, 2009), and the paths of hurricanes and tropical storms (Demuth et al., 2006) differ over time or events. Consequently, the potential applications of these methods may extend far beyond human behavioral science.

Goals of a path mapping algorithm and related problems

In any of these problem domains, two common goals emerge: first, one would like to measure how similar two paths are to one another (to score performance, give feedback, perform a clustering analysis, evaluate an experimental manipulation, etc.); second, one may want to determine the point on one path that best corresponds to a point on another path. This correspondence could be used to judge, for example, whether path memory errors produce a bowed serial position function, whether a remembered path was incompletely recalled, whether particular features of one path (e.g., loops, non-rectilinear corners, etc.) are reproduced accurately, or whether a set of paths are evidence for a single common strategy.

Non-ordered mappings Even though paths are ordered data sequences, a simple and fast approach for measuring path similarity (or divergence) might be to ignore the order of points and use methods for determining multi-point similarity. For example, one might simply map each point on one path to its nearest neighbor on another path (cf. Dasarathy, 1991), or use some other general point-to-set optimization (Zangwill, 1969; Hogan, 1973). Similar approaches have been successful in many non-spatial domains (e.g., Littman, Dumais, & Landauer, 1998), and may be sufficient for simple paths as well. However, it is easy to identify conditions under which this would produce suboptimal or illogical results. For example, the grey hashed triangle in Panel C of Fig. 1 indicates an area of one path that might be mapped out-of-sequence to the single (closest) point on the second path. This may be adequate for some applications, but by taking advantage of the ordinal nature of paths and constraining the mapping to follow this order, we may be able to induce better inferences about how two paths correspond.

Measuring trajectory similarity A number of algorithms have been developed for measuring similarity of trajectories (i.e., paths with timing information). These have proven useful for machine vision applications that track objects, people, or animals, and for storing and indexing these in databases to enable later retrieval by similarity. For example, Yanagisawa et al. (2003) developed a shape-based similarity and indexing scheme for trajectories, and similarly Chen et al. (2005) compared a number of methods for measuring distance between trajectories, and introduced a measure called Edit Distance on Real Sequence (EDR). These methods rely to some extent on the time-based nature of the trajectories, which are implicitly ordinal, to help determine how elements from one path map onto elements on the second path. Trajectories sampled at regular time intervals produce a natural division of a path into equal-duration path segments, and even with irregular sampling, trajectory times can be interpolated to produce equal-duration segments. The EDR method relies on timing information and its measure of similarity is a generalization of the edit distance, which counts the number of operations that are required to make one trajectory match the other in terms of timing and position. These approaches have also focused on the practical problems related to efficiently storing, indexing, and retrieving trajectories, which involves other tradeoffs we will not address here.

Such methods do not work directly for paths that lack timing information. In many cases, one or both paths lack a measured time-series component. This is especially true when one path is the experimental stimulus and the other path is generated by a subject as a measure of memory, search, or communication. Yet even when timing

information is available, the timescale of one trajectory may necessarily be different from the timescale of another trajectory. For example, it may take only seconds to remember and draw a flown flight path that took minutes or hours to complete; similarly, flying a planned flight route may take hours, even if the plan was generated in seconds. And in both of these cases, it is not always the case that the timing information is relevant. Thus, even when some timing information is available, it may not be helpful or possible to use it to compare paths.

Shape similarity algorithms A third relevant domain is the study of shape and object representation and matching (see, e.g., Latecki and Lakämper (2000), Belongie et al. (2001), Belongie et al. (2002), and Gorelick et al. (2006)), and related work on human movement analysis (Gorelick et al. 2007). Although there is little fundamental difference between paths and polygons, the goals for object and shape matching typically differ from those of path mapping. For example, shape algorithms typically ignore absolute coordinates, orientation, scale, and arbitrary affine transformations, in an attempt to identify the best mapping between two polygons based on their overall shape. These allow automated analysis of imagery that present similar objects from different perspectives, in different sizes and with variations on shape, as might be expected from natural objects and natural views (Pizlo, 2010; Mueller, 2010). These details that are specifically ignored by shape-matching algorithms can be critical for measuring the similarity between two paths, because here location and scale typically do matter. For shape matching, the value that is optimized is not typically related to the distance between shapes, but often involves statistics related to the relative angles between consecutive line segments (Latecki & Lakämper, 2000). Thus, although shape similarity algorithms may not be directly useful, their basic approach of using optimization to find a mapping between corresponding contours is quite relevant.

In summary, although there are several related existing approaches that could be used to map paths (non-ordered mappings, trajectory similarity, and shape similarity), none will be completely adequate for comparing paths. In the next section, we will identify some of the specific challenges a path mapping algorithm must be able to handle.

Challenges for path mapping and alignment algorithms

To help define the ideal properties of a path mapping algorithm, we have identified a number of potential challenges that such an algorithm must deal with, illustrated in Fig. 1. Once we review these challenges, we will examine the extent to which several candidate methods address these challenges. Then, we will describe a method designed

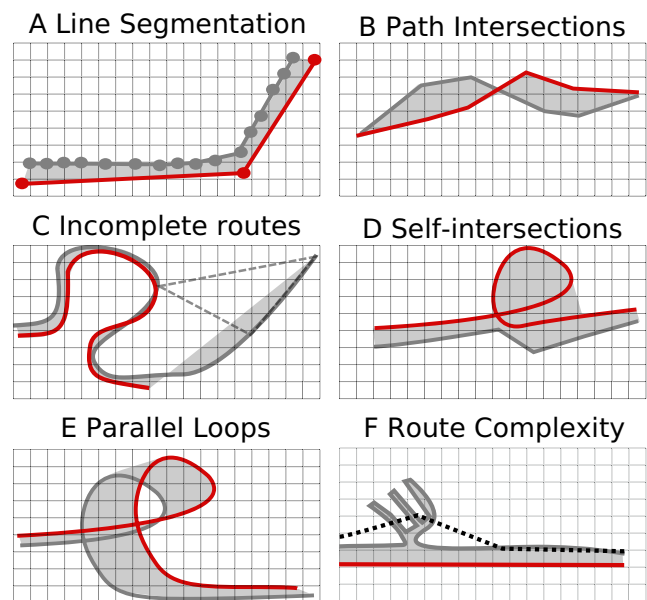


Fig. 1 Potential challenges for a path correspondence algorithm include that it should: **A** be relatively insensitive to how a path is segmented; **B** handle intersecting paths; **C** accommodate paths of different overall lengths; **D** handle paths that intersect themselves and form loops; **E** handle parallel loops on both paths, and **F** provide reasonable measures that account for differences in path complexity

to robustly measure path correspondence in the face of these challenges, and introduce a free software package that implements the algorithm.

When measuring the deviation between two paths, one issue is that two paths are not guaranteed to be partitioned into equal-length segments (a problem that trajectories may not have, because the time-based partitioning is an important part of the trajectory). Panel A shows an example of this. These paths might be produced by a memory test in which a given path is defined by just a few line segments, but a recalled path may be a digitized line which has hundreds of small elements. To handle this, an algorithm must be able to map multiple points and segments of one path onto a single segment of another path. A second issue (Panel B) is that two similar paths often intersect one another. As we will see later, not all measures handle intersections easily. A third issue (Panel C) is that one path may be substantially shorter in length than another. In the case of the paths in Panel C, the shorter path should probably be mapped onto the similar section of the longer path rather than being mapped proportionally from start to end onto the other path. At the same time, paths that are perhaps simplified versions of a more complex path (Panel D) should map one to the other in a reasonable way. Another issue is that a path may loop or intersect *itself* (also Panel D), and the algorithm should handle these situations robustly, including (as in Panel E) where both paths loop in parallel. Finally, as we

will discuss in greater detail later, there are deviations such as in Panel F that are not measured well by area deviation alone, as they may represent large differences that produce a small-area polygon, and an algorithm should be sensitive to these.

Candidate methods for measuring divergence between paths

Although no widely used methods exist for measuring the divergence and correspondence between paths, an intuitive solution is to measure the area *between* two paths. In this section, we will review a number of alternatives for computing area-based divergence, as well as several based on path shape, before introducing our approach.

Area-based measures

Many potential methods for finding the similarity between paths rely on measuring the area subtended by the polygon formed by connecting the paths. In previous behavioral research, the typical approach for measuring the area deviation between two paths (cf. Veinott et al., 1999) has been to overlay the paths with transparent graph paper and count the grid cells within their deviation. A polygon whose area can be measured can be formed from two paths by connecting the starting point of each path to the ending point of the other path. Such an area-based measure has the advantage of being identical regardless of whether a path has been segmented or digitized. Thus, area-based measures can be invariant to the details of path creation that may impact path segmentation. Furthermore, an area-based measure is interpretable by dividing the area by the mean path length, which gives the mean deviation per unit length. Many trajectory similarity algorithms are area-based in some way (Yanagisawa et al., 2003; Chen et al., 2005), but rely on the fact that a time-based natural segmentation exists, enabling one to use the area-based difference between two segments to determine the best mapping between segments. Without timing information, area-based methods are still applicable. Next we will review and evaluate several methods that can be used to measure the area of a polygon.

The graph planimeter, Green's theorem, and the surveyor's formula Physical computers called planimeters have been available for measuring the area of bounded regions since Johann Hermann introduced a device in 1814 (see Care 2010, Chapter 2). The most famous of these was developed by Amsler (1856), as a mechanical implementation

of Green's theorem.¹ Green's theorem shows how a double integral over a region (i.e., its area) can be computed via a line integral over its boundary (see Gatterdam, 1981, for an accessible explanation of this device). For polygons (i.e., area domains that are bounded by line segments), the solution offered by Green's theorem can be simplified to an algorithm relying on determinants of consecutive line segments known as the *surveyor's formula*, or the simplified but algebraically identical *shoelace formula* (Braden, 1986). These are perhaps more convenient for digitized paths, where a path can easily be transformed into a series of line segments and the area can be automatically computed via software.

These methods are fairly simple and can be used to efficiently compute the area of the polygon formed by two paths that do not intersect. However, they will not provide the correct answer in a number of cases, and so cannot be used as a general solution. For example, when two paths cross, this creates regions of negative area that need to be detected and handled separately. We will consider this type of strategy next.

Surveyor's formula applied to segmented polygons The surveyor's formula approach could be salvaged by searching for the points where a line segment from one path intersects with a segment from the other path. Then, using this information to isolate each non-intersected sub-polygon, one could compute areas using the surveyor's formula and find the sum. However, this method still has a number of problems. First, the approach still presents difficulties when paths intersect *themselves*. For example, consider Panel D of Fig. 1. Here, one path loops, while the other does not. The isolated polygons produce one area, but a better deviation might be the grey area indicated in the figure. But more importantly, this hypothetical algorithm now requires substantial search and analysis to use, and so the goal of a simple solution has been lost. Consequently, we might consider digital approaches akin to the graph-paper solution based on flood-fill algorithms.

Flood-fill algorithms Using a flood-fill algorithm (e.g., Asundi & Wensen, 1998; Shaw, 2004) might be a way to avoid some of the complexity involved in the surveyor's formula. In this approach, one would render the two paths on a pixel-based background, connect the ends, and use flood-fill to 'paint' everything outside the resulting polygon. Then, one could simply count the number of unpaired

¹Green's theorem had been known for some time, but had only recently been proven by Riemann (1851) in his dissertation.

pixels and determine an area-based deviation. This method could be very accurate (limited only by the pixel size) and it may avoid the need for identifying and isolating each sub-polygon. However, this approach also faces some of the same problems as the earlier approaches, especially when paths intersect themselves. For example, if a pair of closely intertwined paths formed a larger loop that intersected itself, a simple flood-fill algorithm would mismeasure the correspondence unless the area in the center of the loop was subtracted. Attempts to improve on flood-fill would require identifying and isolating parts of paths and larger loops, and so again the hope of a simple algorithm is lost.

General limitations of area-based solutions In general, simple area-based solutions appear to be untenable, primarily because of the need to handle path intersections, which requires additional analysis. However, even if simple crossovers and intersections can be identified, other problems persist. For example, in Panel E of Fig. 1, two paths may match one another fairly accurately, but together they form a set of parallel loops. A naive area-based solution might count the entire loop as a mismatch, as it might be ambiguous what corresponds to what. Clearly, if the two paths deviate from one another very little, the deviation of the paths should be close to 0, but substantial logic could be necessary to know which areas between line segments should or should not be counted in this deviation. Furthermore, suppose a pair of paths looped several times (e.g., if a pair of aircraft circled over a target)—this might be very difficult for a simple area-based measure to handle. Next, even for a simple polygon, there are situations where total area may not be the best measure of path deviation. For example, consider Panel F of Fig. 1. Here, one path is linear, but the second path is jagged. Because of the zigzag followed by this second path mapping, there is relatively little area within this polygon. In cases such as this, a simple polygon area approach is probably not even appropriate, and the dissimilarity measure should be greater than what would be obtained by area alone. For comparison, consider the third (dashed) path in Panel F, which has an overall larger distance deviation from the straight path than the wavy path, but is subjectively and objectively more similar in most ways. Finally, area-based solutions in general can provide dissatisfying results, because they do not determine how two paths actually correspond or map onto one another. This information could be very useful, and we would like to obtain this mapping as a consequence of measuring the deviation between paths.

For these reasons, we argue that simple area-based measures must be abandoned. However, the optimization approaches used in both shape-matching and trajectory-mapping algorithms offer an alternative that might avoid the need to analyze the topology of crossing paths and

determine which areas should or should not count. In this approach, if we can determine a measure of the deviation between any segment on one path and any point or segment on a second path, we may be able to apply standard mathematical programming algorithms that find a mapping having the smallest overall deviation. Thus, the smallest possible deviation between two paths will be identified, and a correspondence that maps points on one path to points on the second will be produced as a consequence. We will describe this approach next.

An algorithm for finding the least-cost areal mapping between Paths (ALCAMP)

The goal of finding point-to-set and set-to-set mappings (sometimes called set-valued maps or set-valued functions) has been studied in a number of domains of applied and pure mathematics, and general proofs for optimizing such mappings date back at least to Zangwill (see Zangwill, 1969; Hogan, 1973). Importantly, any mapping can be thought of as incurring a cost defined by a cost function or objective function, and the goal of optimization is to find the mapping that incurs the least cost. In our approach, we use optimization to find a mapping between paths that produces the least total cost, where cost is the area between corresponding points and segments. This approach shares some of the same logic as shape-based similarity measures (Latecki & Lakämper, 2000) and trajectory-mapping algorithms (Yanagisawa et al., 2003; Chen et al., 2005), in that it finds an optimal mapping by applying a cost function to deviations between paths. Formally, we use the same general dynamic programming algorithms used to compute sequence distance in many contexts, perhaps most widely for aligning sequential data such as letter strings or DNA sequences (the so-called edit or Levenshtein distance; cf. Levenshtein, 1966). For a string of letters, the goal of the edit distance is to find the minimum number of changes needed to transform one string into another, permitting additions or subtractions, each having an equal cost. For the EDR trajectory-mapping approach of Chen et al. (2005), the goal is to find the minimum number of changes of one path (in either timing or position) required to transform it into another path. There, each change also incurs a cost, but the cost depends on the distance each segment must be moved, and is not fixed for each operation. This distance, if weighted by segment length, is analogous to an area-based cost function, which is roughly the same as our approach.

The primary goal of our algorithm is to determine a mapping between two paths that minimizes a cost function. To begin with, we must first define three things: the specific meanings of paths, mappings, and cost functions. Once this

is complete, we will apply a dynamic programming optimization algorithm to efficiently identify the cost of the best mapping.

Defining a path

A path is an ordered series of points and the line segments connecting those points. A path can be completely defined by the points alone:

$$A_{points} = \{A_1, A_2, A_3, A_4, \dots\}, \quad (1)$$

or by the edges in the path

$$A_{edges} = \{A_{12}, A_{23}, A_{34}, \dots\}, \quad (2)$$

or more redundantly by the interleaved series of points and edges:

$$A_{merged} = \{A_1, A_{12}, A_2, A_{23}, A_3, A_{34}, A_4, \dots\} \quad (3)$$

or, alternately,

$$A_{merged} = \{A_{11}, A_{12}, A_{22}, A_{23}, A_{33}, A_{34}, A_{44}, \dots\} \quad (4)$$

where each single-number (or doubled-number) subscript element indicates a point, and each two-subscript element indicates a line segment. Note that although A_{points} is unconstrained, the end point of each element of A_{edges} must be connected to the starting point of the next element. Within A_{merged} , we will generically refer to points and segments as nodes, using point to refer to odd nodes and segment to refer to even nodes.

Determining a mapping between two paths

We define a mapping as an indicator function $M(A_i, B_j) \in \{0, 1\}$ that identifies which nodes of path A correspond to which nodes of path B .² In general, one might place no constraints on $M()$, allowing any point or segment on one path to map onto any or all points or segments on another path. However, because paths are ordered sets, many of the possible mappings are incompatible. Thus, we define the subset of *proper mappings* between two paths A and B as mappings that satisfy the following conditions: each *point* of A corresponds to at least one *node* of B , each *point* of B corresponds to at least one *node* of A , and the correspondences are strictly non-decreasing, so that the correspondence map does not move ‘backward’ on either path. More formally, a proper mapping satisfies the following constraints:

- $\forall x \in X_{points}, y \in Y_{merged}, \sum_y M(X_i, Y_j) \geq 1$

²In the notation of Hogan (1973), a point-to-set map Ω is defined as the mapping $X \rightarrow 2^X$; i.e., the power set. For any node of A (corresponding to a row of the matrix implied by $M(A_i, B_j)$), the power set 2^X is just all possible combinations of 1s and 0s on that row. Similarly, each column of $M(A_i, B_j)$ represents a point-to-set mapping between a node of B and the nodes of A .

- $\forall y \in Y_{points}, x \in X_{merged}, \sum_x M(Y_i, X_j) \geq 1$
- if $M(A_i, B_j) = 1$, then $M(A_s, B_t) = 0$ if $s > j$ and $t < i$, or if $s < j$ and $t > i$

Note that we do not require or consider the mapping between two edges. This is mainly done to simplify the optimization problem, and it is typically possible to infer whether two edges correspond by using a ‘sandwich’ rule (i.e., an edge is sandwiched between its endpoints which always have an explicit mapping to the other path), and similarly to identify how any point lying on one path corresponds to the other path (even when it lies in the middle of a segment).

A proper mapping is illustrated in Fig. 2, where a dashed line between points or segments indicates that $M(A, B) = 1$. Here, A has four points and three segments, for a total of seven nodes; B has seven points and six segments, for a total of 13 nodes. Starting at the left, points A_1 and B_1 are mapped onto one another so that $M(A_1, B_1) = 1$. Next, $M(A_1, B_{12}) = 1$, and so on, so that eventually each point on each path is connected to a point or segment on the other path, and the connections do not move backward to a previous node. However, multiple adjacent nodes (edges or points) on one path may be mapped onto a single node on another path. The mapping illustrated in Fig. 2 is shown in matrix format in Table 1, which shows each pair of $M(A, B)$ for which $M(A, B) = 1$. Note that any proper mapping must have a non-zero value in each odd row and each odd column, and the path through M must be non-decreasing in both row and column.

Computing the area-based cost for any mapping Measuring the deviation between paths can be translated into the network (a planar graph) shown in Fig. 3. Each node corresponds to one entry in the matrix in Table 1. To understand a mapping, we can conceive of it as a process by which we link two paths together, starting at the two end-points, and moving to the adjacent nodes on one or the other path, until we arrive at the endpoints of the two paths. The process of moving between adjacent nodes can be framed as moving along an arrow connecting nodes in the graph. Movement of the mapping along the nodes of path A corresponds to following rightward arrows in the graph; movement along the nodes of path B corresponds to following downward arrows in the graph, and a concomitant movements along paths A and B together correspond to a diagonal arrow in the graph.

As discussed earlier, there are no nodes corresponding to edge-edge connections (indicated by a small unfilled circle). Moving from one node-node connection to an adjacent node-node connection incurs a cost equal to the area deviation of the polygon formed by this movement. These

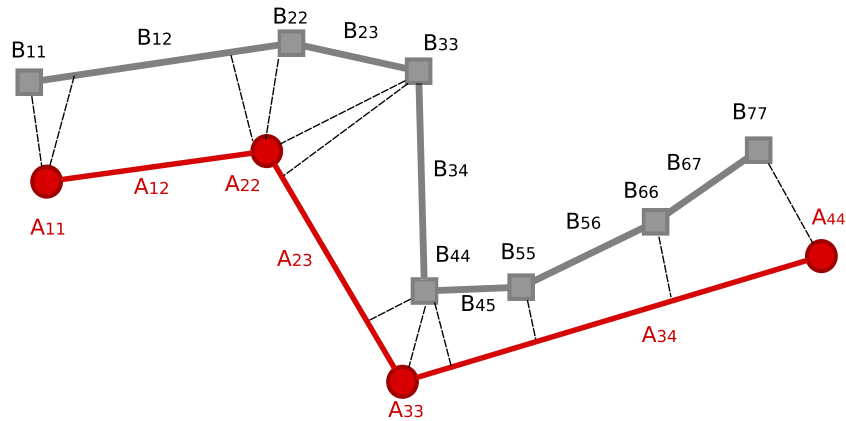


Fig. 2 Example alignments between two paths. Each point must map onto either an edge or a point in the corresponding path

transitions form either 0-area polygons, triangles, or quadrilaterals whose area-based costs can be computed by geometric formulas, or (less efficiently) by the Surveyor’s formula, or (hypothetically) by other cost functions. Consequently, we will use $S()$ to represent the area-based cost, with the assumption that it could be computed in various ways, or perhaps substituted with another cost function. Before a final cost of a path can be computed, we first must determine how to measure the area for consecutive points on one path that are mapped onto the same segment on another path.

One way to measure the cost between consecutive points on one path that are mapped onto a single line segment on a second path is to select the points on the segment that are closest to the points, and find the area of that quadrilateral. Figure 4 shows three example points: (P_1 , P_2 , and P_3), and how they might be connected to the segment RS . RS lies on line t , and unless P_i lies on t , there is another line (t') orthogonal to t that passes through P_i , and thus through point Q on line t . If the point Q is on segment

RS , we refer to the point as being ‘opposite’ the segment. Whether a point is ‘opposite’ a segment can be precomputed by calculating the length of the line segment RP projected onto line RS via a dot product, and then dividing by the length of RS to determine the proportional length z (where $z = ((S - R) \cdot (P - R)) / (|(S - R)|)$). If z is less than 0, P is attached to R ; if z is greater than 1, P is attached to S , and otherwise P is attached to $Q = R + z(S - R)$.

To compute the area corresponding to these mappings, it is convenient to define o_i :

$$o_i = \begin{cases} 0 & \text{if } z_i \leq 0, \\ z_i & \text{if } 0 \leq z_i \leq 1, \\ 1 & \text{if } z_i \geq 1. \end{cases} \tag{5}$$

Here, o_i represents the proportion along the segment’s length to which a point is mapped: a value of 0 indicates one end of the segment, a value of 1 indicates the other end,

Table 1 The proper mapping between path A and B shown in Fig. 2. Connected nodes are indicated with a 1; a ‘.’ indicates nodes are unconnected

	A11	A12	A22	A23	A33	A34	A44
B11	1
B12	1	.	1
B22	.	.	1
B23
B33	.	.	1	1	.	.	.
B34
B44	.	.	.	1	1	1	.
B45
B55	1	.
B56
B66	1	.
B67
B77	1

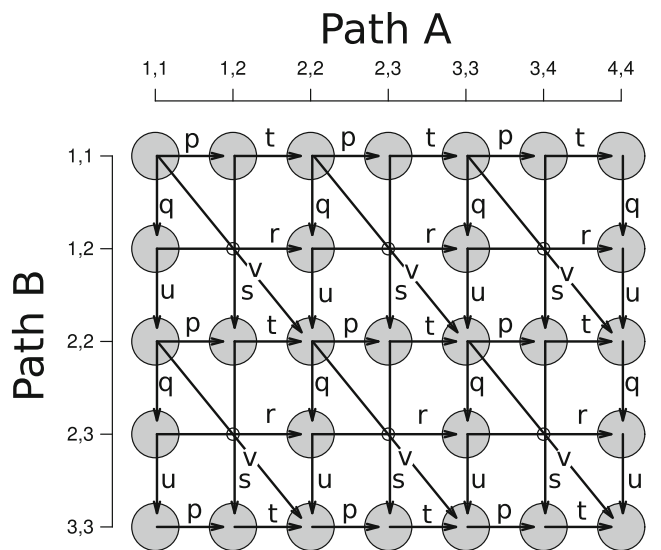


Fig. 3 The connection graph describing the path alignment problem. In the ALCAMP algorithm, transitions are only allowed along arrows, and each transition incurs a cost related to the areal deviation between the two nodes (i.e., either a line with area 0, a triangle, or a quadrilateral)

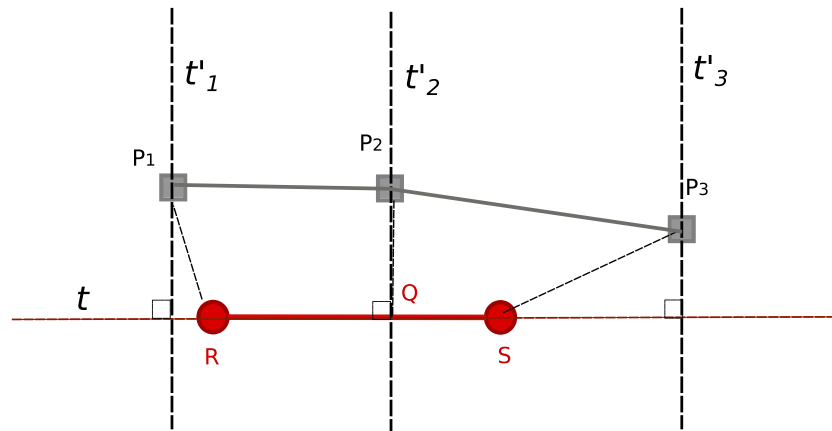


Fig. 4 Three points connected to their closest points on RS . The line perpendicular to t passing through P_1 (t'_1) falls to the left of R , and so P_1 gets attached to R . The line t'_3 falls to the right of S on t , and so gets attached to S . The line t'_2 falls within RS at point Q , and so gets attached there

and a value between 0 and 1 indicates an intermediate point. This can be used to compute the distance of one side of a quadrilateral or triangle represented by a transition, and it allows the areal deviation to be carved into multiple polygons whenever a single point on one path is mapped to a segment on the other path.

If several consecutive points are mapped onto a single segment, we can simply map each point onto its nearest point on the segment (as shown in Fig. 4). However, this has the potential for reversing direction of the mapping. For example, if comparing a straight segment to a loop whose points are all opposite the segment, the mapping from each point on the loop to the closest point on the segment will march forward, reverse, and forward again. The proper way to handle this is to break the long segment into subsegments and solve the mapping problem using the same optimization approach we use to solve the entire problem. This will obtain the smallest cost independent of path segmentation. We provide tools for doing this, but it can come at the cost of a substantially more complex (and time-consuming) optimization, because paths may be segmented into dozens or hundreds of sub-segments if one of the paths has many segments. In our tests using this approach, adding these additional segments typically had little or no impact on the final mapping.

Now, we can compute formulas for each type of transition in Fig. 3. Note that there are seven distinct transition types, labeled p, q, r, s, t, u , and v . Each of these transitions represents a different transition in the mapping. We will show how each cost is computed, considering the incoming transitions for any graph node. First, consider a point-point mapping. The previous mapping could be the previous pair of points on each path, which skips back two nodes in the network. Here, v is the cost of a quadrilateral defined by the four points being considered (where A_i is used as a shorthand for point $A_{i,i}$),

$$v = S(A_i, A_{i-1}, B_{j-1}, B_j) \tag{6}$$

For v , the two paths may in fact cross one another, and a naive cost function might not be able to detect or compensate for this. Under our approach, we handle this by detecting crossovers in a pre-processing step, and divide both crossing segments into two line segments at the cross-over point.

Other possible transitions to point–point mappings come from moving from the previous *segment* on one or the other path. Consider first the case where an adjacent segment and point on B are both mapped onto $A_{i,i}$. This cost is related to the triangle whose sides are defined by segment $B_{j-1,j}$ and point $A_{i,i}$. However, the actual starting point of the triangle depends on the value of o_{i-1} . These areas are defined by t and u , respectively, for triangles that move along either path A (right in the planar graph) or path B (down in the planar graph):

$$t = S(A_{i-1} + o_{i-1} \times (A_i - A_{i-1}), A_{i-1}, B_j) \tag{7}$$

$$u = S(A_i, B_{j-1} + o_{j-1} \times (B_j - B_{j-1}), B_j) \tag{8}$$

Next, consider mappings between a segment $A_{i-1,i}$ on path A and a point B_j on path B . Transitions to this mapping are obtained either by moving from a point–segment mapping (between B_{j-1} and $A_{i-1,i}$; the previous point of B mapped to the same segment of A), or from a point–point mapping (between B_j and A_{i-1} ; the current point of B and the end point of the A segment). These transitions incur costs s and p , respectively. If we consider the corresponding case of segment $B_{j-1,j}$ mapped onto point A_i , the transition costs are q and r . In these formulas, r and s both represent the cost associated with two consecutive points mapped onto a single segment, whereas p and q both represent the cost associated with a consecutive point and edge on one path mapped onto a single point on the other path.

The cost of r and s transitions only makes sense when both of the points in the transition are opposite the line segment. If $o_i \leq 0$ or $o_i \geq 1$, the transition is really a move

to the point–point mapping, and so the value of r or s is given the cost value of ∞ to force the mappings to use the point–point route and simplify the optimization. The formulas for r and s are essentially identical but with the role of A and B swapped. In each case, the polygon area formed by the transition actually depends on the value of o for each point–segment pair:

$$r = S(B_i, B_{j-1}, A_{i-1} + o_{i-1} \times (A_i - A_{i-1}), \\ A_{i-1} + o_i \times (A_j - A_{i-1})) \\ \text{iff } o_{i-1}, o_i \in (0, 1). \quad (9)$$

$$s = S(A_i, A_{i-1}, B_{j-1} + o_{j-1} \times (B_j - B_{j-1}), \\ B_{j-1} + o_j \times (B_j - B_{j-1}),) \\ \text{iff } o_{j-1}, o_j \in (0, 1). \quad (10)$$

Finally, costs p and q both involve transitions from the immediately previous point–point mapping to a point–segment mapping. Again, p and q are identical with the roles of A and B swapped:

$$p = S(A_i, B_j + o_j \times (B_j - B_{j-1})) \quad (11)$$

$$q = S(B_j, A_i + o_i \times (A_i - A_{i-1})) \quad (12)$$

These seven formulas provide the complete cost function for all possible transitions within the graph. Together, the cost of any proper mapping, which is a route through the graph from the upper left to lower right, can be computed as the sum of the costs along that route. Of all the possible routes, a subset will incur the minimal cost, and our goal is to identify that cost and the mappings that produce it. Before we introduce an algorithm that will optimize this, we will first consider several preprocessing steps that might be used.

Pre-processing steps

Prior to applying ALCAMP to a pairing of paths, three distinct pre-processing steps might be needed. We offer specific solutions to two of these, although one might use a number of alternate approaches.

So far, we have assumed that a path is a relatively true representation of the original behavior, sampled without considerable noise. If a path is known to be sampled with noise, one may wish to first pre-process the path with some time-series smoothing or interpolation algorithms. The details of this approach would need to be fairly specific to the type of data one is considering. A smoothed representation of a noisy data source may enable the path to be represented using fewer points or path segments, and thus could allow for a more efficient optimization. However, this step is not necessary to apply the algorithm, and it may not even typically provide a better measure of path similarity, given the current approach already optimizes a

cost function. Consequently, we have merely mentioned the possibility but do not explore it in this paper.

A second pre-processing step can be done to improve efficiency. One can remove redundant points, simplifying the path by merging segments that lie on the same line, or that do not contribute significantly to the shape of the path. A robust algorithm has been proposed by Latecki and Lakämper (2000), and we provide an implementation of their algorithm in our software library in a routine called `SimplifyPath`. This not only removes completely redundant points, but it also can remove points that only impact the shape of the overall path in small ways.

Finally, a third approach is to do further interpolation to ensure that a series of points on one path can map in a monotonic fashion to points on a second path. For example, we can add points implied by each Q point from Fig. 4, allowing each point to connect directly to another point, and eliminating any need to allow multiple consecutive points to map onto the same segment. This interpolation approach can reduce efficiency of the overall algorithm, but its advantage is that it enforces a strictly non-decreasing mapping along each path segment. The alternative approach, allowing consecutive least-area mappings of points onto a single segment, can produce a larger cost, and is thus sensitive to how paths are segmented, but typically has only a minor impact and can be substantially more efficient.

Finding the least-cost mapping

Once a pair of paths has undergone pre-processing, we can construct a graph such as in Fig. 3 and compute costs associated with each transition between nodes. Any legal route from the upper left to the lower right corresponds to a mapping between paths, and the sum of the transitions is the cost of that mapping. For paths with non-trivial curvature, some of these mappings will produce greater area costs than others, because they will create a map that produces overlapping regions whose areas are counted multiple times. However, there will typically be a set of mappings whose costs are identical and equal to the minimum cost.

For any network, the total number of possible paths is finite but large, and for paths with only a handful of nodes, one could perform exhaustive search to find the least-cost path. For complex mappings (e.g., for paths where at least one is smoothly generated and so includes hundreds of nodes), a complete search would be computationally infeasible. However, there exists a standard dynamic programming algorithm that allows the cost of the shortest path in such a graph to be computed in a time proportional to $M \times N$, where M and N refer to the number of nodes in paths A and B .

An efficient solution to finding the least-cost mapping for such a graph is closely related to the algorithm for

computing edit distance. The mechanics and proofs of the optimality of such algorithms are well understood and appear in many textbooks on dynamic programming (see Marzal and Vidal, 1993, for a detailed explanation using edit distance). Despite the fact that there are a large number of routes through the network, the cost of the best route(s) can be found by visiting each node only once, provided that all the incoming nodes to any node can be computed (i.e., there are no circuits in the directed graph). An intuitive explanation for this uses a recursive argument. Consider that any partial route through the network (e.g., starting at the upper left node but ending at an intermediate node) represents a partial mapping between the paths. Thus, suppose we have identified the least-cost route to the three intermediate nodes that precede the final (bottom right) node of the graph. Then the best route can be identified by computing the transition costs between each of these three previous nodes and the final node mapping, and choosing the minimum value. But finding the least-cost route to any of these previous nodes can likewise be computed if the minimum costs of the incoming nodes are also known. Applying the algorithm recursively, one can continue until one arrives at the upper-left node of the network, which has no input nodes and thus a cost of 0. Once this first node is known, the least-cost route through the first row can be computed easily, moving left to right on each row, and handling each row top to bottom in sequence, as the prior costs that need to be known have already been computed. Thus, each node of the network is visited once, and at the end the cost of the best mapping is available. The algorithm to find the smallest cost has computational complexity $O(n \times m)$, although an additional pass through the network is required to determine the set of mappings that produce this optimal cost. Importantly, multiple routes could (and in our case typically do) produce a least-cost mapping, and although this method can determine the cost of best mapping, it does not identify the set of all least-cost mappings.

Finding the family of least-cost mappings

This method is sufficient for computing the least-cost area between paths, and is robust to a number of problems for simpler polygon-based measures. However, it may not provide an unambiguous answer to the path correspondence problem, because there are typically many equivalent mappings that produce the same area. For example, consider the three mappings in Fig. 5, which each have the same cost because each exactly fills the area (in fact, almost every proper route through the cost network produces the same cost!). This network is shown in Fig. 6, with the three mappings highlighted. Here, the top-most route in Fig. 5 corresponds to the upper mapping in Fig. 6, the center route corresponds to the lower mapping, and the third route in

Fig. 6 corresponds to the diagonal mapping in the planar graph.

These three mappings have the same area-based cost, which illustrates how area-based measures alone often cannot identify a best mapping between paths. A number of additional constraints can be considered for this situation, but one we suggest aims to minimize the mean distance of the connections between paths. To do this, a second pass is made through the network to determine all possible minimal-cost mappings. Finally, this least-cost subnetwork is searched using the same dynamic programming approach used in the first pass to find the mapping that minimizes the sum of the node-node, node-segment, and segment-node distances (rather than areas). The third mapping in Fig. 5 (corresponding to the center route in Fig. 6) is a result of this minimization, which produces the smallest average distance between nodes among all mappings having the same area. Visually, this mapping is one that appears to directly connect each point on one path with its closest point on the opposite path.

This completes the basic description of the ALCAMP procedure. Software for implementing these methods is available in the `pathmapping` package for the R statistical computing language. Next, we will examine how this process fares on example toy and real-world problems.

Applications and example

Conceptually, the algorithms described here are fairly simple, but the implementation requires substantial programming. We have implemented the algorithm using the R statistical computing language (R Core Team 2013). The complete source code is available for download from <https://sites.google.com/a/mtu.edu/mapping/tasks>, and is available in the `pathmapping` package via the Comprehensive R Archive Network (CRAN). A description of the functions is provided in Appendix.

The algorithm produces reasonable answers in the face of all of the challenges identified in Fig. 1, as shown in the first two rows of Fig. 7. The third row shows example path pairs from a map communication task. In this task, one participant saw one reference path and communicated this verbally to a second participant who drew it. The left panel of row 3 shows a highly accurate communication; the center shows a communication with two distinct error patterns: one very costly error where the northward path was mis-drawn by about 300 pixels (despite the fact that this only used two placed points), and another region involving dozens of given and placed points that were also mis-specified badly, yet this cost was considerably less. The right panel of row 3 shows an example where a complex portion given path was approximated with a single line, with reasonably good results.

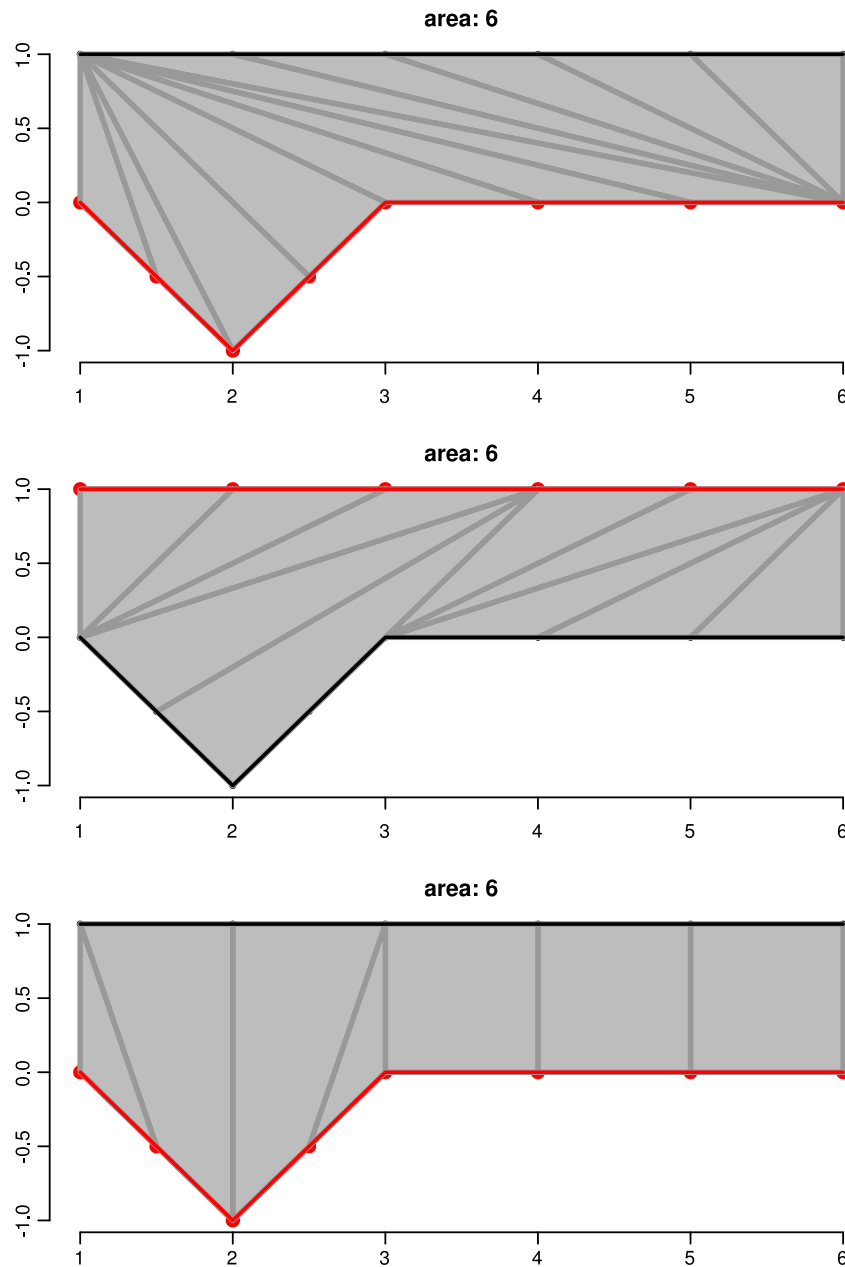


Fig. 5 Three equal-area mappings between two paths. The first mapping prefers transitions along the bottom path; the second mapping prefers transitions along the top path, and the third mapping minimizes

the sum of the distances between each node on one path and the other path. Colors are used to illustrate different segments of the mappings

However, it also highlights a limitation of our area-based approach: the two lower points on the simpler drawn path each miss the given point by about 50 pixels, and clearly were intended to correspond to one another. The least-area correspondence does not map these critical corners to one another.

Validating area-based deviation as an index of memory

The bottom row of Fig. 7 shows three trials from a task whose data and methods were originally reported by

Perelman and Mueller (2013). In this task, 21 undergraduate participants each took part in five trials of a simulated aerial search task³ implemented using the PEBL experimentation software platform (Mueller & Piper, 2014). In the task, participants were given a fixed amount of fuel (corresponding to 1,000 screen updates, or about 80 s) to fly a simulated aircraft over a map in search of specific targets located at half of 12 prespecified hot zones. Flight was

³This search task software is available at <https://sites.google.com/mtu.edu/aerialste/>.

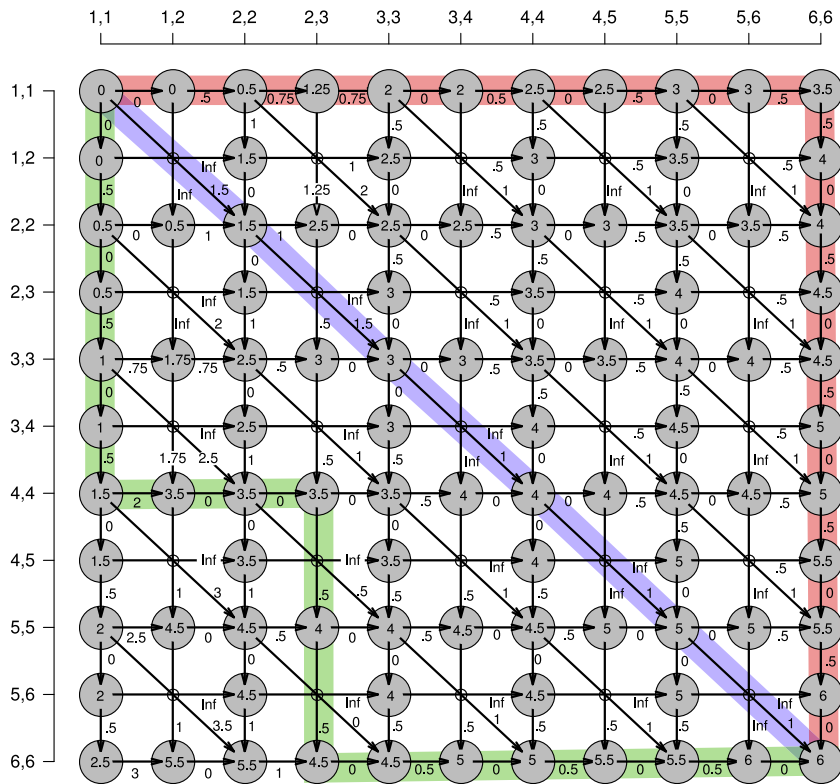


Fig. 6 The planar graph depicting the possible mappings between the two paths. Each route through the network corresponds to one mapping in Fig. 5

controlled using a touch screen monitor that changed the current destination of the aircraft. Once the aircraft reached that location, it smoothly circled that point until redirected (leading to circles in the flight route). To encourage participants to search efficiently, the fuel provided was not sufficient to search all targets, and participants found about half of the targets on any given trial. Once completed, two memory tests were performed: first, the participant drew their best memory of their flown path on the touch screen; then, they were asked to identify the locations of the found targets within the 12 unlabeled hot zones. For the second task, the hit rate (proportion of originally found targets remembered) and false alarm rate (proportion of foils incorrectly remembered) were both computed, and a simple difference score of HR-FAR served as a memory performance measure (Signal detection statistics such as d' were untenable because many participants had hit rates or false alarm rates of 1 or 0).

We wanted to examine whether individual differences in memory for the path would predict differences in target localization accuracy. To the extent that localization accuracy is negatively correlated with given-to-remembered path deviation, this would indicate that a common set of representations, processes, or strategies is responsible.

Because the flown distances were all roughly the same, total area-based deviation provided a good measure of path

dissimilarity, and thus memory precision. The mappings in the bottom row of Fig. 7 were fairly representative of performance in this task. Total area-based deviation ranged from about 35,000 pixels to around 80,000 pixels, with a few trials even larger than that.

We computed the average area-based deviation score for each participant (across five trials), and compared it to their target localization memory score. Results (see Fig. 8) showed a high negative correlation across participants ($R = -.759$, $t(19) = -5.08$, $p < .001$), indicating that participants who had better memory for where they found specific targets were able to redraw their flown path more precisely. Although several psychological explanations could account for this correlation (differences in alertness, attention, a common memory resource, etc.), this demonstrates that the area-based deviation score has convergent validity as a measure of memory, correlating with memory for other aspects of the flown path.

Discussion

The method we describe seeks to measure the deviation between two paths, and along the way identifies a mapping between the paths that enables additional analytics. We believe this algorithm will offer new ways to examine

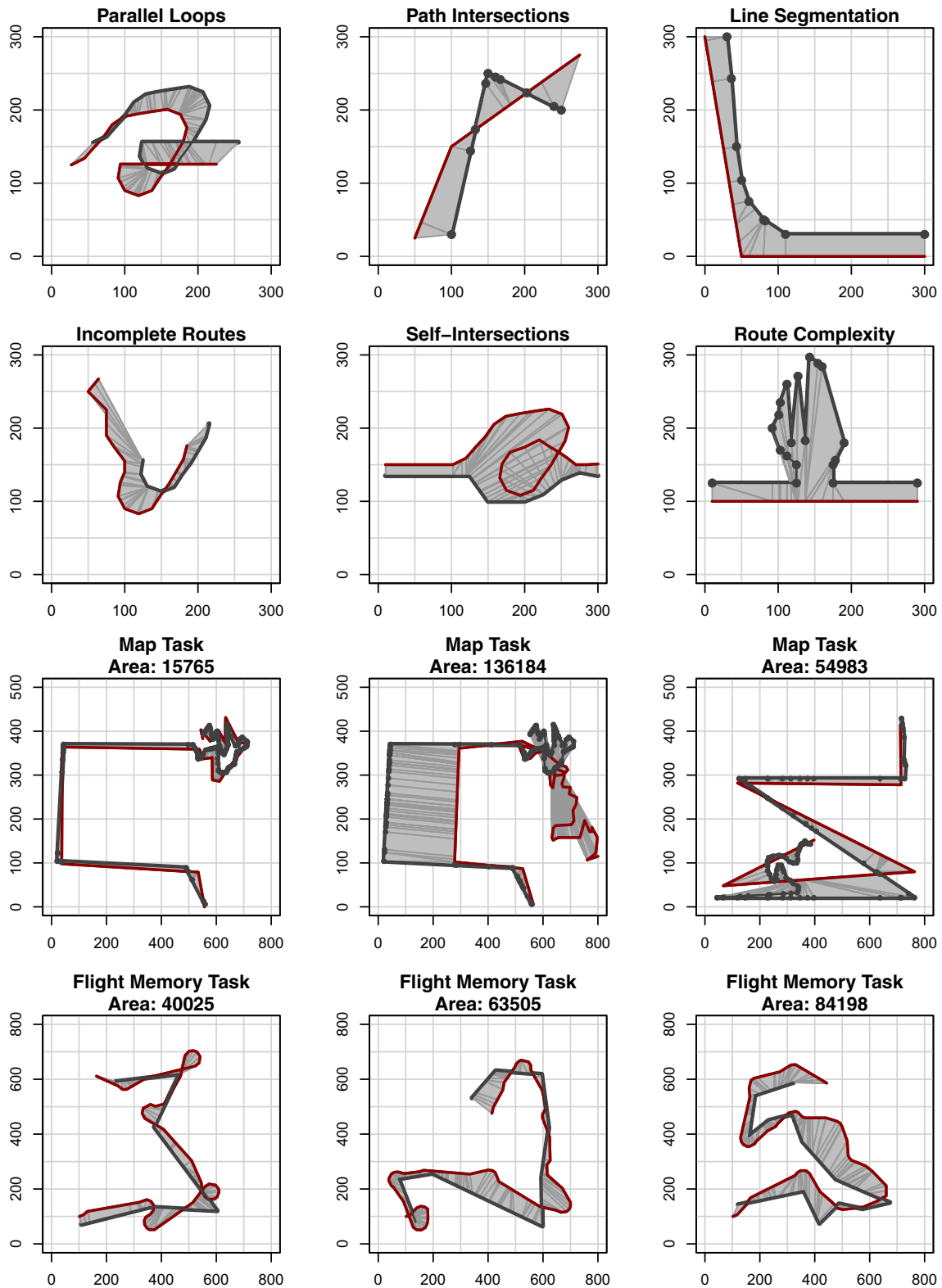


Fig. 7 Example mappings produced by the least-cost mapping algorithm. The *first two rows* show how the algorithm’s output for small sample problems. The *third row* shows output for three map route drawing data sets, and the *fourth row* shows output for three flight path memory trials

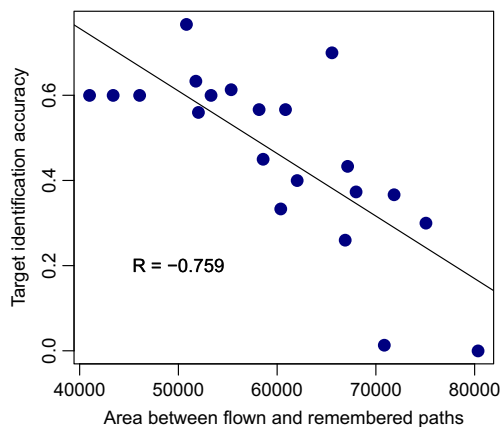


Fig. 8 Scatterplot showing relationship across participants of mean area-based deviation (*horizontal axis*) and memory score (target hit rate - false alarm rate)

current data, inspire new types of experiments enabled by the method, and provide new ideas for data mining of path data. As currently defined, the method is sufficient for many applications, although there are a number of limitations that should be recognized (and that could be the source of future development).

Limitations

Area-based costs The present algorithm uses the area between paths to measure the cost of a deviation. This has a drawback, in that there are often many ways in which the area between paths can be segmented, and so there is some ambiguity in determining a specific mapping. We have dealt with this by performing a second pass over the sub-network that produces the minimal area, and minimizing a second value that corresponds, roughly, to the sum of the distance between all the points on each path and the other path.

Area-based cost can be viewed as the integral of a constant cost function over the length of corresponding segments. It may be possible to use a different cost function to constrain the optimization in a way that tends to produce a single best mapping in one step. For any application, the cost of deviations may not map directly onto the area (and the cost might either be concave or convex with the absolute deviation). Any convex loss function, such as squared error, would satisfy Jensen's inequality and insure that the mapping prefers segment mappings that have smaller absolute distances, and this could be a reasonable alternative cost function to adopt. Now, the polygon-cost would no longer be computable with the Surveyor's formula, but Green's theorem would still hold and allow a relatively efficient means for computing this cost for each pair of segments. The pathmapping package

allows using alternative cost functions, but we have not systematically explored the consequences of these alternative costs.

An area-based cost function can also be insensitive to complexity at different scales. That is, a long, smooth or straight path may be easily captured by just a few line segments, whereas a path with detailed curvature in a local region may require dozens of line segments to capture it. Yet, moving or deleting one point of the large-scale region may have a large cost, but smoothing a detailed path may have only limited cost. For example, consider the center panel of the third row of Fig. 7. There, two mismatching points produced a large deviation.

One way to address this is to use the mapping obtained via ALCAMP and take an approach similar to the EDR algorithm of Chen et al. (2005). Here, one can identify a cost threshold, and count the number of edges that have a cost less than the threshold. In this case, the left half of the paths would result in two above-threshold points, whereas the right-half would result in dozens of above-threshold points. This may or may not be appropriate given a particular application.

Use of raw path data A second limitation that should be acknowledged regards the nature of the path data. The algorithm takes as input a path, defined as an ordered sequence of (x, y) locations. However, many paths may not be defined in this way. For example, if one were measuring how closely a GPS-tracked path followed a road drawn on a map, the GPS trail might be an ordered sequence of points, but the road might be simply a wavy line drawn on the map. Alternately, jet stream paths may be inferred from a complex vector field, and other paths may be similarly inferred from some other primary measure. These paths and others might also be more like a stream, with their widths varying along their extent. To use the algorithm, one must still (manually or automatically) translate the path into a single ordered sequence of points. Algorithms for doing this are beyond the scope of the current research, and might depend on the types of paths being investigated.

Practical efficiency for long paths Another related limitation is that for paths digitized or sampled from a continuous source, (e.g., GPS history trail, flight path, mouse tracking, etc.), the sampled path may have hundreds or thousands of points. Depending on computing resources, this may make the present algorithm impractical. In situations such as this, one could reduce the number of points in one or both paths using methods such as the shape evolution algorithm introduced by Latecki and Lakämper (1999). This would, at a minimal cost

of precision, allow more efficient calculation of distance between paths.

Additional spatial, time, and other dimensions Finally, an additional limitation is that the current algorithm only handles two spatial dimensions. This approach might be useful for three-dimensional data, such as search behaviors (in aerial, water-based, or virtual movements), or it could consider time as a third dimension (producing movement trajectories such as a football running route or military attack), or incorporate both (comparing landing or take-off paths of pilots). The present implementation is limited to two dimensions, but in principle could be extended to larger dimensional spaces.

Summary and conclusions

Researchers in behavioral sciences often obtain spatial paths in which they wish to derive a measure of similarity to another path. Previous methods adopted have been ad hoc and have required tedious hand-coding. Here, we propose an algorithm that (1) finds the distance between two paths, which can be used as a measure of dissimilarity; and (2)

determines an optimal correspondence between elements of each path, mapping each point to a point or segment on the other path. An implementation of the algorithm using the R statistical computing language is available, and can provide useful metrics both within behavioral research and in other domains.

Notes The authors gratefully acknowledge the advice of and discussions with the late Prof. Arthur F. Veinott, Jr., who pointed us toward the dynamic programming solutions to this problem.

Appendix

Overview of the pathmapping package for R

The R software code we provide contains a number of functions that enable a user to determine the distance and a mapping between paths. Along with a number of basic functions to compute distances, costs, and closest points between points and segments, the following functions are available

Table 2 Table A1 documents the main user-accessible functions. e, a path refers to a two-column matrix of x,y coordinates; a mapping refers to the data structure produced by the CreateMap function

Function name	arguments	Return value	Description
CreateMap	path1, path2, plotgrid =F, Cost=Cost.area, nondecreasingos=F, verbose=F, insertopposites=T	mapping	Finds a minimum-cost mapping between two paths
GetMinMap	mapping, leftbias=T, verbose=F	mapping	Finds the minimum-area mapping that also minimizes linear distance between points on paths
PlotMap	mapping, cols, linecol, xlim, ylim		Plots the paths and the mapping between paths
Insert Intersections	path1, path2, insertopposites=T, verbose=F	mapping	Inserts points on one path that are opposite points on a second path.
SimplifyPath	path, tolerance=.075, faster=T, verbose=F, plot=F	path	Removes points that do not significantly impact the shape of the path, using method by Latecki and Lakämper (1999)
surveyors	polygon	An area	Computes area inside polygon
PathDist	path	A distance	Computes length of a path
SummarizeMapping	map	A matrix	Returns a common matrix showing the direct correspondence between two paths.
PathOverlap	map	A proportion	Proportion of paths that overlap in their mapping.

for computing and displaying path mappings. Complete documentation and further examples are provided within the package, which can be installed and accessed by typing the following within R:

```
> install.packages("pathmapping")
> library(pathmapping)
> help(pathmapping)
```

First, each path must be defined as two columns of x,y values, for example:

```
pathA <- rbind(c(0,0),c(5,0),c(10,0))
pathB <- rbind(c(1,1),c(2,-1),
               c(3,4),c(5,1),c(10,-3))
```

The main function that computes the basic mapping is called `CreateMap`, which takes the two paths as arguments, and several optional arguments: `plotgrid`, which plots the grid network seen in Fig. 9; `nondecreasingos`, which will force the mapping to be monotonic when multiple points are mapped onto a single segment, and `verbose`, which prints out status information during computation. Importantly, the `CreateMap` function returns a data structure that contains a minimum-cost mapping as well as many other aspects of the problem.

```
answer<- CreateMap(pathA,pathB,F)
```

The returned value includes the following elements which can be accessed via the `$` accessor (e.g., `dist` is accessed using `answer$dist`). Values include:

- `path1`, the first path values
- `path2`, the second path values
- `linkcost`, a matrix showing the linear distance between each node and the other path
- `leastcost`, a matrix showing the minimum cumulative area-based costs at each node.
- `bestpath`, an $N \times M \times 2$ matrix that records the least-cost area-based path to each node. Note that the output of `CreateMap` does not choose a path that minimizes the linear distance between paths.
- `opposite`, a matrix computing o_{ij} for each node-segment pair.
- `dist`, the areal deviation between paths

The `InsertIntersections` function will first insert points on paths where the paths intersect, and then (optionally) insert points on segments of one path that are opposite points on the second path. This function is called by the `CreateMap` function, and so does not need to be called by the user directly.

This data structure can be displayed using `PlotMap()` function, which draws a polygon representing each mapping along the path, as shown in the top panel of Fig. 10. The data structure produced by `CreateMap` can be re-analyzed by

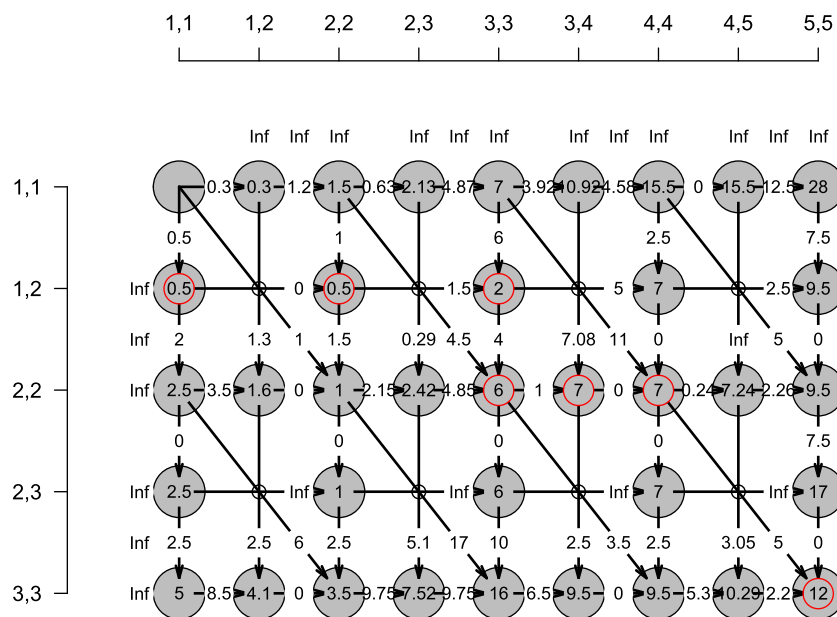


Fig. 9 Example grid plotted using the `CreateMap` function

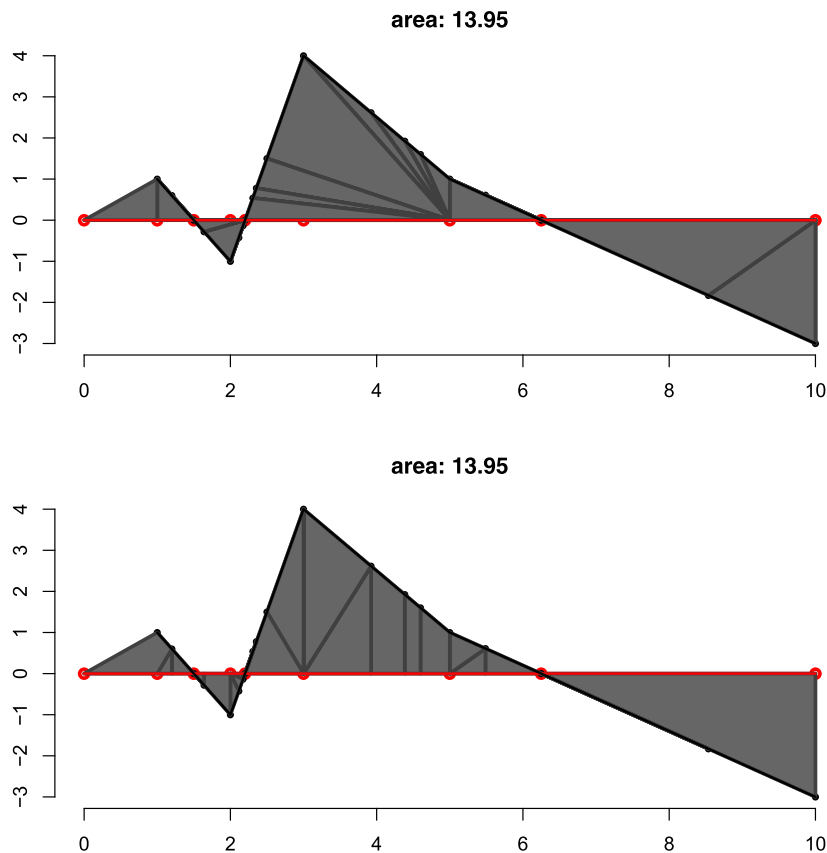


Fig. 10 Example paths, including default mapping produced by the `CreateMap` function, and distance-minimized mapping produced by `GetMinMap`

the `GetMinMap` function, which itself returns a mapping object:

```
answer2 <- GetMinMap(mapping)
```

This function adds an additional value to the mapping object, `$leastcostchain`, which records the cumulative cost network for the linear distance optimization. The result of this function is plotted in the bottom panel of Fig. 10.

The basic mapping function can be fairly computationally intensive, given that it is not currently parallelized, it is running in interpreted R code, and it is currently not optimized for speed. For paths with relatively few points (fewer than 100), the mapping completion time is tolerable, but when mapping one or more complex paths containing hundreds of points, the time to complete the mapping can take much longer. Any individual path can

be reduced to a smaller number of critical points using the `SimplifyPath` function:

```
path1b <- SimplifyPath(path1)
```

The default tolerance value (0.075) has produced reasonable results for fairly long and complex paths. A value very close to 0 will only remove points that lie on a segment containing its direct neighbors; a value larger will be more likely to distort smooth curves.

Finally, the function `PathDist()` computes the overall length of a path, by summing the length of each segment connecting adjacent points. This can be useful for computing an average deviation score (by dividing the area deviation by the path length), or for other metrics.

Details of flight path memory example

For the flight memory example in the paper, each flown path (`flownpath`) involved 1,001 points, represented as

a two-column matrix of x and y coordinates. The following code represents the logical steps (with brief comments) used to perform this analysis (although we omit the logic to

read in files from each individual file and separate these into different distinct trials).

```

library(pathmapping)
flownpath <- read.csv("flownpath.csv")
fpath2 <- SimplifyPath(flownpath,tolerance=.1)
mempath <- read.csv("mempath.csv")
map0 <- CreateMap(mempath,fpath2,insertopposites=F)
map <- GetMinMap(map0)
PlotMap(map)
cat("Path deviation: " , map$deviation, "\n")

```

```

#load library
#read in flown path
#simplify flown path
#read in recalled path
#create mapping
#find best mapping
#plot mapping
#Print out deviation

```

References

- Abend, W., Bizzi, E., & Morasso, P. (1982). Human arm trajectory formation. *Brain: A Journal of Neurology*, *105*(2), 331–348.
- Amsler, J. (1856). Über die mechanische Bestimmung des Flächeninhaltes, der statischen Momente und der Trägheitsmomente ebener Figuren insbesondere über einen neuen Planimeter (On the mechanical assessment of the area, the static moments, and the moment of inertia of figures in a plane, in particular on a new planimeter) Schaffhausen: Beck. available from. <http://books.google.com/books?id=qpQ5AAAACAAJ>
- Anderson, A. H., Bader, M., Bard, E. G., Boyle, E., Doherty, G., & Garrod, S. (1991). The HCRC map task corpus. *Language and Speech*, *34*(4), 351–366.
- Asundi, A., & Wensen, Z. (1998). Fast phase-unwrapping algorithm based on a gray-scale mask and flood fill. *Applied Optics*, *37*(23), 5416–5420.
- Barton, N. P., & Ellis, A. W. (2009). Variability in wintertime position and strength of the North Pacific jet stream as represented by re-analysis data. *International Journal of Climatology*, *29*(6), 851–862.
- Belongie, S., Malik, J., & Puzicha, J. (2001). Matching shapes. In *Proceedings of the Eighth IEEE International Conference on Computer Vision (ICCV-2001)* (Vol. 1, pp. 454–461).
- Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(4), 509–522.
- Braden, B. (1986). The surveyor's area formula. *The College Mathematics Journal*, *17*(4), 326–337.
- Brown, G., Anderson, A. H., Shillcock, R., & Yule, G. (1984). *Teaching talk*. Cambridge University Press.
- Care, C. (2010). *Technology for modelling: Electrical analogies, engineering practice, and the development of analogue computing, Chapter 2*. Berlin: Springer. doi:10.1007/978-1-84882-948-0
- Ceriani, S. A., Roth, J. D., Evans, D. R., Weishampel, J. F., & Ehrhart, L. M. (2012). Inferring foraging areas of nesting loggerhead turtles using satellite telemetry and stable isotopes. *PLoS One*, *7*(9), e45335.
- Chen, L., Özsu, M. T., & Oria, V. (2005). Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (pp. 491–502).
- Croxall, J. P., Silk, J. R., Phillips, R. A., Afanasyev, V., & Briggs, D. R. (2005). Global circumnavigations: Tracking year-round ranges of nonbreeding albatrosses. *Science*, *307*(5707), 249–250.
- Dasarathy, B. V. (1991). *Nearest neighbor (NN) norms: NN pattern classification techniques*. IEEE Computer Society Press.
- Demuth, J. L., DeMaria, M., & Knaff, J. A. (2006). Improvement of advanced microwave sounding unit tropical cyclone intensity and size estimation. *Journal of Applied Meteorology and Climatology*, *45*(11), 1573–1581.
- Egevang, C., Stenhouse, I. J., Phillips, R. A., Petersen, A., Fox, J. W., & Silk, J. R. (2010). Tracking of Arctic terns *Sterna paradisaea* reveals longest animal migration. *Proceedings of the National Academy of Sciences*, *107*(5), 2078–2081.
- Erdenebaatar, B., & Humphrey, C. (1996). Socio-economic aspects of the pastoral movement patterns of mongolian herders. In C. Humphrey & D. Sneath (Eds.), (pp. 58–110). White Horse Press.
- Fisk, H. N. (1944). Geological investigation of the alluvial valley of the lower Mississippi River. *War Department, Corps of Engineers*. Retrieved from. <http://biotech.law.lsu.edu/climate/mississippi/fisk/fisk.htm>
- Freeman, J. B., & Ambady, N. (2010). MouseTracker: Software for studying real-time mental processing using a computer mouse-tracking method. *Behavior Research Methods*, *42*(1), 226–241.
- Gatterdam, R. W. (1981). The planimeter as an example of Green's theorem. *American Mathematical Monthly*, 701–704.
- Gawarkiewicz, G. G., Todd, R. E., Plueddemann, A. J., Andres, M., & Manning, J. P. (2012). Direct interaction between the Gulf stream and the shelfbreak south of New England. *Scientific Reports*, *2*.
- Gorelick, L., Blank, M., Shechtman, E., Irani, M., & Basri, R. (2007). Actions as space-time shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(12), 2247–2253.
- Gorelick, L., Galun, M., Sharon, E., Basri, R., & Brandt, A. (2006). Shape representation and classification using the Poisson equation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(12), 1991–2005.
- Guilford, T., Meade, J., Willis, J., Phillips, R. A., Boyle, D., & Roberts, S. (2009). Migration and stopover in a small pelagic seabird, similar point, for: Manx shearwater *Puffinus puffinus*: Insights from machine learning. *Proceedings of the Royal Society B: Biological Sciences*. rspb–2008.
- Hogan, W. W. (1973). Point-to-set maps in mathematical programming. *SIAM Review*, *15*(3), 591–603.
- Istomin, K. V., & Dwyer, M. J. (2009). Finding the way. *Current Anthropology*, *50*(1), 29–49.
- Kennedy, W. G., Hailegiorgis, A., Rouleau, M., Bassett, J., Coletti, M., & Balan, G. (2010). MASON HerderLand: Modeling the origins of conflict in East Africa. *Proceedings of the First Annual Conference of the Computational Social Science Society*.

- Latecki, L. J., & Lakämper, R. (1999). Convexity rule for shape decomposition based on discrete contour evolution. *Computer Vision and Image Understanding*, 73(3), 441–454.
- Latecki, L. J., & Lakämper, R. (2000). Shape similarity measure based on correspondence of visual parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10), 1185–1190.
- Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10, 707.
- Littman, M. L., Dumais, S. T., & Landauer, T. K. (1998). Automatic cross-language information retrieval using latent semantic indexing. In *Cross-language information retrieval* (pp. 51–62). Springer.
- MacGregor, J. N., & Ormerod, T. (1996). Human performance on the traveling salesman problem. *Perception & Psychophysics*, 58(4), 527–539.
- Marzal, A., & Vidal, E. (1993). Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9), 926–932.
- Mueller, S. T. (2010). A partial implementation of the BICA cognitive decathlon using the Psychology Experiment Building Language (PEBL). *International Journal of Machine Consciousness*, 2(2), 273–288.
- Mueller, S. T., Perelman, B. S., & Simpkins, B. G. (2013). Pathfinding in the cognitive map: Network models of mechanisms for search and planning. *Biologically Inspired Cognitive Architectures*, 5, 94–111.
- Mueller, S. T., & Piper, B. J. (2014). The Psychology Experiment Building Language (PEBL) and PEBL test battery. *Journal of Neuroscience Methods*, 222, 250–259.
- Mueller, S. T., Price, O. T., McClellan, G. E., Fallon, C. K., Simpkins, B., & Cox, D. A. (2010). Cognitive Performance Prediction with the T3 Methodology, (Tech. Rep.). Dayton, OH: DTRA Technical report, HDTRA-1-08-C-0025.
- Perelman, B. S., & Mueller, S. T. (2013). Examining memory for search using a simulated aerial search and rescue task. In *Proceedings of the 17th International Symposium on Aviation Psychology (ISAP17)* (pp. 302–309). Dayton, OH.
- Pizlo, Z. (2010). *3D shape: Its unique place in visual perception*. MIT Press.
- Pizlo, Z., Stefanov, E., Saalweachter, J., Li, Z., Haxhimusa, Y., & Kropatsch, W. G. (2006). Traveling salesman problem: A foveating pyramid model. *The Journal of Problem Solving*, 1(1), 8.
- R Core Team (2013). R: A Language and Environment for Statistical Computing [computer software manual]. Vienna, Austria. <http://www.R-project.org/>
- Riemann, B. (1851). *Grundlagen für eine allgemeine Theorie der Functionen einer veränderlichen complexen Grösse* (Doctoral dissertation, EA Huth). See reference at: <http://www.emis.ams.org/classics/Riemann/Grund.pdf>
- Ruddle, R. A., & Lessels, S. (2009). The benefits of using a walking interface to navigate virtual environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(1), 5.
- Shaw, J. (2004). QuickFill: An efficient flood fill algorithm. Online article Retrieved from: <http://www.codeproject.com/Articles/6017/QuickFill-An-efficient-flood-fill-algorithm>
- Tuch, D. S., Reese, T. G., Wiegell, M. R., & Wedeen, V. J. (2003). Diffusion MRI of complex neural architecture. *Neuron*, 40(5), 885–895.
- van der Stigchel, S., Meeter, M., & Theeuwes, J. (2006). Eye movement trajectories and what they tell us. *Neuroscience & Biobehavioral Reviews*, 30(5), 666–679.
- Tuch, D. S., Reese, T. G., Wiegell, M. R., & Wedeen, V. J. (2003). Diffusion MRI of complex neural architecture. *Neuron*, 40(5), 885–895.
- Veinott, E. S., Olson, J., Olson, G. M., & Fu, X. (1999). Video helps remote work: Speakers who need to negotiate common ground benefit from seeing each other. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (pp. 302–309).
- Weimerskirch, H., & Wilson, R. P. (2000). Oceanic respite for wandering albatrosses. *Nature*, 406(6799), 955–956.
- Yanagisawa, Y., Akahani, J., & Satoh, T. (2003). Shape-based similarity query for trajectory of mobile objects. In *Mobile Data Management* (pp. 63–77). Springer.
- Zangwill, W. (1969). *Nonlinear programming: A unified approach*. Prentice-Hall, Englewood Cliffs, NJ.
- Zhang, R., Nordman, A., Walker, J., & Kuhl, S. A. (2012). Minification affects verbal-and action-based distance judgments differently in head-mounted displays. *ACM Transactions on Applied Perception (TAP)*, 9(3), 14.