String Tightening as a Self-Organizing Phenomenon

Bonny Banerjee, *Student Member, IEEE*

***Abstract*—The phenomenon of self-organization has been of special interest to the neural network community throughout the last couple of decades. In this paper, we study a variant of the self-organizing map (SOM) that models the phenomenon of self-organization of the particles forming a string when the string is tightened from one or both of its ends. The proposed variant, called the string tightening self-organizing neural network (STON), can be used to solve certain practical problems, such as computation of shortest homotopic paths, smoothing paths to avoid sharp turns, computation of convex hull, etc. These problems are of considerable interest in computational geometry, robotics path-planning, artificial intelligence (AI) (diagrammatic reasoning), very large scale integration (VLSI) routing, and geographical information systems. Given a set of obstacles and a string with two fixed terminal points in a 2-D space, the STON model continuously tightens the given string until the unique shortest configuration in terms of the Euclidean metric is reached. The STON minimizes the total length of a string on convergence by dynamically creating and selecting feature vectors in a competitive manner. Proof of correctness of this anytime algorithm and experimental results obtained by its deployment have been presented in the paper.**

***Index Terms*—Convex hull, homotopy, neural network, self-organization, shortest path, smooth path, tighten string.**

I. Introduction

SELF-ORGANIZATION, as a phenomenon, has received considerable attention from the neural network community in the last couple of decades. Several attempts have been made to use neural networks to model different self-organization phenomena. One of the most well known of such attempts is that of Kohonen's who proposed the self-organizing map (SOM) [1] inspired by the way in which various human sensory impressions are topographically mapped into the neurons of the brain. SOM possesses the capability to extract features from a multidimensional data set by creating a vector quantizer by adjusting weights from common input nodes to M output nodes arranged in a 2-D grid. At convergence, the weights specify the clusters or vector centers of the set of input vectors such that the point density function of the vector centers tend to approximate the probability density function of the input vectors. Several authors in different contexts reported different dynamic versions of SOM [1]–[10].

Manuscript received November 18, 2005; revised February 16, 2006 and July 12, 2006; accepted December 27, 2006. This work was supported by participation in the Advanced Decision Architectures Collaborative Technology Alliance sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAD19-01-2-0009.

The author is with the Laboratory for AI Research, Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: banerjee@cse.ohio-state.edu).

Digital Object Identifier 10.1109/TNN.2007.891192

In this paper, assuming a string is composed of a sequence of particles, we claim that the phenomenon undergone by the particles of the string, when the string is pulled from one or both ends to tighten it, is that of self-organization, by modeling the phenomenon using a variant of SOM, called the string tightening self-organizing neural network (STON). We further use the proposed variant to solve a few well-known practical problems—computation of shortest path in a given homotopy class, smoothing paths to avoid sharp turns, and computation of convex hull. Other than theoretical considerations in computational geometry [11], computation of shortest homotopic paths is of considerable interest in robotics path-planning [12], AI (diagrammatic reasoning) [13], VLSI routing [14], and geographical information systems. Smooth paths are required for navigation of large robots incapable of taking sharp turns, and also for handling unexpected obstacles. To generate a path that is smooth, shorter, collision-free, and is homotopic to the original path requires generation of the configuration space of a robot which is computationally expensive and difficult to represent [15]. Computation of a convex hull finds numerous applications in computational geometry algorithms, pattern recognition, image processing, and so on. The aim of this paper is to study the properties of STON and how it might be applied to solve some practical problems as aforementioned.

The remainder of this paper is organized as follows. In Section II, the STON algorithm is described assuming the given string is sampled at a frequency of at least $d/2$ where d is the minimum distance between the obstacles. Thereafter, an analysis of the algorithm is presented along with proof of its important properties and correctness. Section IV discusses how STON might be extended when the aforementioned constraint on sampling is not met. The extension is used for computation of shortest path in a given homotopy class. Proof of correctness and complexity analysis of the extension are also included. Finally, simulation results from real life and synthesized data sets are presented for the problems—computation of shortest homotopic path, smooth path, and convex hull—using both the original and extended algorithms. The paper concludes with a general discussion.

II. STON Algorithm

A. Homotopy

A string π in a 2-D space ($\Re^2$) might be defined as a continuous mapping $\pi : [0,1] \rightarrow \Re^2$, where $\pi(0)$ and $\pi(1)$ are the two terminal points of the string. A string is simple if it does not intersect itself; otherwise, it is nonsimple. Let π_1 and π_2 be two strings in $\Re^2$ sharing the same terminal points, i.e., $\pi_1(0) = \pi_2(0)$ and $\pi_1(1) = \pi_2(1)$, and avoiding a set of obstacles $P \subset \Re^2$. The strings π_1 and π_2 are considered to be homotopic to each other or to belong to the same homotopy class, with respect to the set of obstacles P, if there exists a continuous function $\Psi : [0,1] \times [0,1] \rightarrow \Re^2$ such that the following hold:

1) $\Psi(0,t) = \pi_1(t)$ and $\Psi(1,t) = \pi_2(t)$, for $0 \leq t \leq 1$;
2) $\Psi(\lambda,0) = \pi_1(0) = \pi_2(0)$ and $\Psi(\lambda,1) = \pi_1(1) = \pi_2(1)$, for $0 \leq \lambda \leq 1$;
3) $\Psi(\lambda,t) \notin P$, for $0 \leq \lambda \leq 1$ and $0 \leq t \leq 1$.

Informally, two strings are considered to be homotopic with respect to a set of obstacles, if they share the same terminal points and one can be continuously deformed into the other without crossing any obstacle. Thus, homotopy is an equivalence relation.

Given a string π_i, specified in terms of sampled points, and a set of obstacles P, STON computes a string π_s such that π_i and π_s belong to the same homotopy class, and the Euclidean distance covered by π_s is the shortest among all strings homotopic to π_i. It is noteworthy that π_s is unique and has some canonical form [16].

B. Objective

Assume a string wound around obstacles in $\Re^2$ with two fixed terminal points. A shorter configuration of the string can be obtained by pulling its terminals. The unique shortest configuration can be obtained by pulling its terminals until they cannot be pulled any more. The proposed algorithm models this phenomenon as a self-organized mapping of the points forming a given configuration of a string into points forming the desired shorter configuration of the string. Let us consider a set of n data points or obstacles, $P = \{p_1, p_2, \dots p_n\}$, representing the input signals, and a sequence of variable (say, k) processors $\langle q_1, q_2, \dots q_k \rangle$ each of which (say q_i) is associated with a weight vector $w_i(t)$ at any time t. A weight vector represents the position of its processor in $\Re^2$. If the k processors are placed on a string in $\Re^2$, the STON is an anytime algorithm for tuning the corresponding weights to different domains of the input signals such that, on convergence, the processors will be located in such a way that they minimize a distance function $\phi(w)$ given by

$$\phi(w) = \sum_{i=1}^{k-1} \|w_{i+1}(t) - w_i(t)\|^2 \tag{1}$$

where q_i and q_{i+1} are two consecutive processors on the string with corresponding weights $w_i(t)$ and $w_{i+1}(t)$ at any time t. The algorithm further guarantees that the final configuration of the string formed by the sequence of processors at convergence lies in the same homotopy class as the string formed by the initial sequence of processors with respect to P. Thus, assuming fixed q_1 and q_k, the STON defines the shortest configuration of the k processors in an unsupervised manner. The phenomenon undergone by the particles forming the string is modeled by the processors in the neural network.

C. Initialization of the Network

The STON is initialized with a given number of connected processors, the weight corresponding to each of which is initialized at a unique point on the given configuration of a string. A feature vector, presented to the STON, is an attractor point in $\Re^2$ and is either created dynamically or chosen selectively from the given set of input obstacles P. The weight vectors are updated iteratively on the basis of the created and chosen feature space $S(t)$; $S(t) = \{x_1(t), x_2(t), \dots x_k(t)\}$ being the set of feature vectors at any time t. It is noteworthy that $x_i(t)$ is not necessarily unique. Unlike SOM and many of its variants, randomized updating of weights does not yield better results in the STON model; updating weights sequentially or randomly both yield the same result in terms of output quality as well as computation time. On convergence, the location of the processors representing the unique shortest configuration homotopic to the given configuration of the string is obtained.

D. Creating/Choosing Feature Vectors

A feature vector $x_i(t)$ is created if the triangular area spanned by three consecutive processors q_{m-1}, q_m, and q_{m+1} does not contain any obstacle $p_j \in P$. In that case

$$x_i(t) = \frac{w_{m-1}(t') + w_{m+1}(t'')}{2} \tag{2}$$

where t' and t'' assume the value t if the weight has not yet been updated in the current iteration or *sweep* and the value $t+1$ if the weight has been updated in the current sweep, and $1 < m < k$. If an obstacle, say $p_j \in P$, lies within the triangular area spanned by three consecutive processors q_{m-1}, q_m, and q_{m+1}, the feature vector $x_i(t)$ is chosen to be

$$x_i(t) = p_j. \tag{3}$$

For this algorithm, we assume the given string is sampled in such a way that there cannot exist multiple nonidentical obstacles within the triangular area spanned by any three consecutive processors (see Appendix I for how to achieve such sampling).

E. Updating Weights

The STON evolves by means of a certain processor evolution mechanism, given by [17]

$$w_m(t+1) = w_m(t) + \alpha(t)\,[x_j(t) - w_m(t)] \tag{4}$$

where $\alpha(t)$ is the gain term or learning rate which might vary with time and $0 \leq \alpha(t) \leq 1$. $\alpha(t)$ might be unity only when feature vectors are created according to (2). All the weight vectors are updated exactly once in a single sweep, indexed by t.

In this process, modification of weights is continued and the processors tend to produce shorter configurations at the end of each sweep. The weight vectors converge when

$$\|w_{i+1}(t) - w_i(t)\| < \epsilon \qquad \forall i \tag{5}$$

where ϵ is a predetermined sufficiently small positive quantity.

III. Analysis of STON

STON is a variant of SOM. As in SOM, each feature vector in STON pulls the selected processors in a neighborhood towards itself as a result of updating in a topologically constrained manner, ultimately leading to an ordering of processors that minimizes some residual error. Neighborhood, in SOM, is

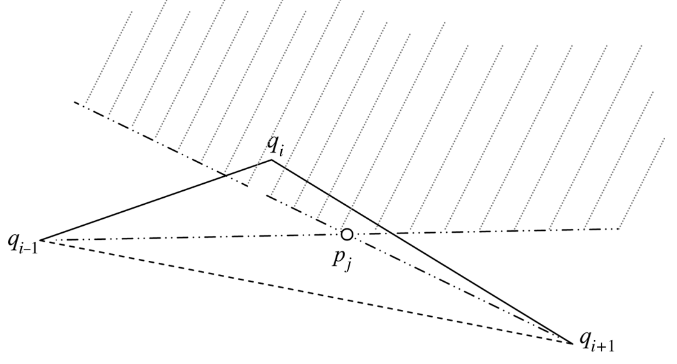

Fig. 1. If p_j lies inside the triangle formed by q_{i-1}, q_i, and q_{i+1}, then q_i has to lie in the shaded region.

something like a smoothing kernel over a 2-D grid often taken as the Gaussian which shrinks over time. In STON, all those processors are included within the neighborhood of a feature vector that form triangles with their adjacent processors such that the feature vector lies within their triangles. Such a neighborhood is conceptually similar to that proposed in [18] where the neighborhood is not dependent on time but on the nature of the input signals. STON incorporates competitive learning as the weights adapt themselves to specific chosen features of the input signal defined in terms of the obstacles. The residual error is defined in SOM in terms of variance while in STON, in terms of Euclidean distance. From the inputs, the net adapts itself dynamically in an unsupervised manner to acquire a stable structure at convergence, thereby manifesting self-organization [19]. The STON possesses certain key properties which are discussed in this section that eventually lead to the proof of correctness of the algorithm.

Property 1: The configuration of a string formed by the sequence of processors at initialization and the same at convergence is homotopic.

Proof: We start by noting that, given a fixed set of obstacles and fixed terminal points, the homotopy class of a string can be altered only by crossing any obstacle. The configuration of a string formed by the sequence of processors at any time t is obtained by updating the weights with respect to the selected feature vectors at time $t-1$, the feature vectors being selected according to (2) or (3). In the first case, creation of a feature vector and updating the weight does not change the homotopy class of the string as there was no obstacle in the triangular area, hence updating the weight did not result in crossing any obstacle.

In the second case, a feature vector is selected by (3) from the set of obstacles. A processor is pulled towards the feature vector by updating its weight. It requires to be proven that by such selection of feature vectors and updating of weights, a string cannot cross any obstacle. Let q_{i-1}, q_i, and q_{i+1} be three consecutive processors on a string with corresponding weights being $w_{i-1}(t')$, $w_i(t)$, and $w_{i+1}(t'')$ and $t', t'' \in \{t, t+1\}$, and $x_i(t) = p_j$ be the selected feature vector (see Fig. 1). In order to complete the proof, we need to show that p_j will never be crossed if the following is true: 1) weight for processor q_i is updated and 2) weight for one of the neighbors of q_i (i.e., q_{i-1} or q_{i+1}) is updated.

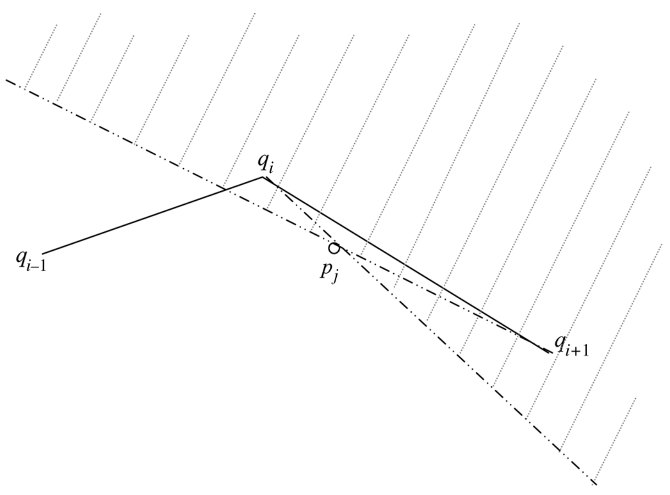

Fig. 2. Feature vector for updating the weight for q_{i+1}, if not p_j, can lie only in the shaded region.

First, we note that in order for p_j to be inside the triangle formed by q_{i-1}, q_i, and q_{i+1}, the processor q_i has to be in the region bounded by extensions of the lines joining q_{i-1} and q_{i+1} to p_j (i.e., the shaded region in Fig. 1). Since updating the weight for processor q_i pulls it towards the obstacle p_j along a straight line, q_i continues to remain within the shaded region after updating, thereby never letting the segments $\overline{q_{i-1}q_i}$ and $\overline{q_iq_{i+1}}$ cross the obstacle p_j. That proves condition 1).

Now, let us consider the case when the weight for a neighbor of q_i, say q_{i+1} without loss of generality, is updated. Let q_{i+2} be the other neighbor of q_{i+1}. Then, either the obstacle p_j lies inside the triangle formed by q_i, q_{i+1}, and q_{i+2}, or it does not. If p_j lies inside, it is the selected feature vector that pulls q_{i+1} towards itself and it will never be crossed [due to condition 1)]. If p_j does not lie inside, then there lies either some other obstacle, say $p_{j'}$, inside the triangle formed by q_i, q_{i+1}, and q_{i+2}, or there does not exist any obstacle inside. In the former case, $p_{j'}$ is the selected feature vector while in the later, the feature vector is created according to (2). In any case, the feature vector can lie only in the partition, bounded by the extensions of the lines joining q_i and q_{i+1} to p_j, in which q_{i+1} lies (shaded region in Fig. 2). Thus, updating the weight for q_{i+1} will not make the string cross the obstacle p_j. In general, updating the neighbors of q_i will not make the string cross p_j, proving condition 2).

From this, we conclude that updating a weight vector with respect to a created or selected feature vector does not change the homotopy class of a string. Hence, the configurations of a string at the end of consecutive sweeps are homotopic. But homotopy is a transitive relation. This concludes the proof that the configurations of a string at initialization and at convergence are homotopic. ■

Property 2: The Euclidean distance covered by the configuration of a string formed by the sequence of processors at time $t+1$ is less than the same at time t.

Proof: Let us consider a triangle formed by three consecutive processors q_{i-1}, q_i, and q_{i+1} with corresponding weights $w_{i-1}(t')$, $w_i(t)$, and $w_{i+1}(t'')$; $t', t'' \in \{t, t+1\}$, on a configuration of a string (see Fig. 3). Let $w_i(t+1)$ be the weight

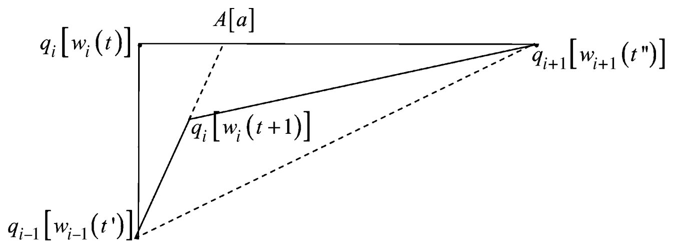

Fig. 3. Euclidean distance covered by a configuration of the string from $q_{i-1}[w_{i-1}(t')]$ to $q_{i+1}[w_{i+1}(t'')]$ via $q_i[w_i(t+1)]$ is less than the same via $q_i[w_i(t)]$.

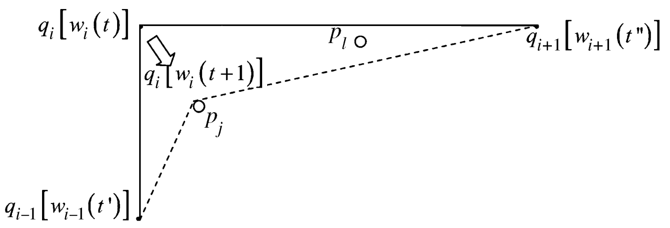

Fig. 4. STON algorithm might fail to perform correctly if the given path is sparsely sampled. The configuration formed by $q_{i-1}[w_{i-1}(t')]$, $q_i[w_i(t)]$, and $q_{i+1}[w_{i+1}(t'')]$ and that formed by $q_{i-1}[w_{i-1}(t')]$, $q_i[w_i(t+1)]$, and $q_{i+1}[w_{i+1}(t'')]$ are not homotopic to each other as the obstacle p_l has been crossed.

vector after updating $w_i(t)$ with respect to a feature vector either created according to (2) or chosen according to (3). We are required to prove that the Euclidean distance covered by a configuration of the string from $q_{i-1}[w_{i-1}(t')]$ to $q_{i+1}[w_{i+1}(t'')]$ via $q_i[w_i(t+1)]$ is less than the same via $q_i[w_i(t)]$.

In order to prove that, we extend the line segment $\overline{q_{i-1}[w_{i-1}(t')]q_i[w_i(t+1)]}$ to intersect the line segment $\overline{q_i[w_i(t)]q_{i+1}[w_{i+1}(t'')]}$ at a point, say A located at a. Then, from Fig. 3, using the triangle inequality, we get

$$\begin{aligned}&\|w_{i-1}(t') - w_i(t+1)\| + \|w_i(t+1) - a\|\\ &\quad < \|w_{i-1}(t') - w_i(t)\| + \|w_i(t) - a\|\\ &\|w_i(t+1) - w_{i+1}(t'')\|\\ &\quad < \|w_i(t+1) - a\| + \|a - w_{i+1}(t'')\|.\end{aligned}$$

From the previous inequalities, we get

$$\begin{aligned}&\|w_{i-1}(t') - w_i(t+1)\| + \|w_i(t+1) - w_{i+1}(t'')\|\\ &\quad < \|w_{i-1}(t') - w_i(t)\| + \|w_i(t) - a\| + \|a - w_{i+1}(t'')\|.\end{aligned}$$

Thus, at any time t, after updating a weight $w_i(t)$, we have

$$\begin{aligned}&\|w_{i-1}(t') - w_i(t+1)\| + \|w_i(t+1) - w_{i+1}(t'')\|\\ &\qquad < \|w_{i-1}(t') - w_i(t)\| + \|w_i(t) - w_{i+1}(t'')\|\end{aligned}$$

i.e., updating $w_i(t)$ contributes to the minimization of the sum of lengths of the segments $\overline{q_{i-1}[w_{i-1}(t')]q_i[w_i(t)]}$ and $\overline{q_i[w_i(t)]q_{i+1}[w_{i+1}(t'')]}$. Each weight is updated exactly once in every sweep. Thus, updating a weight contributes to the minimization of a length of the current configuration of the string at every sweep, thereby minimizing $\phi(w)$. ■

Property 1 shows that the STON algorithm guarantees that the final configuration π_s of the string belongs to the same homotopy class as its initial configuration π_i. From Property 2, it can be seen that if sufficient number of sweeps are computed, the shortest configuration in the given homotopy class can be reached. Thus, the proposed algorithm is correct with respect to the goal of obtaining the shortest homotopic configuration of a string, as defined in Section II-B. This, however, does not guarantee that the optimum solution will always be reached. It is possible for the algorithm to get stuck at suboptimal solutions, a situation that can be averted by choosing α much less than unity when feature vectors are selected by (3). We will discuss this issue further in Section V.

IV. Computation of the Shortest Homotopic Paths

The solution to the problem of computing the shortest homotopic path can be viewed as an instance of pulling a string to tighten it, where the given path corresponds to the initial configuration of the string while the shortest homotopic path corresponds to the tightened configuration of the same string. The STON assumes a string to be sampled such that there exists at most one obstacle in the triangle formed by any of the three consecutive processors. In this section, given a path π_i, we propose an extension of STON to do away with that assumption and apply the extension for computing the shortest homotopic path π_s with respect to a given set of obstacles P, where π_i might be simple or nonsimple. The set of obstacles P is specified as a set of points in $\Re^2$ with no assumption being made about their connectivity. The input path π_i is specified in terms of a pair of terminal points and either a mathematical equation or a sequence of points in $\Re^2$. In the former case, the path is sampled to obtain the sequence of points in $\Re^2$.

A. Extending STON for Sparsely Sampled Paths

When a path is sampled sparsely, it can no longer be guaranteed that at the time of choosing a feature vector, there will exist only one obstacle within the triangular area spanned by any of the three consecutive processors. Hence, the algorithm might fail to perform correctly as Property 1 no longer holds true (see Fig. 4).

Let us assume that the given path is very sparsely sampled. In that case, whenever more than one obstacle point is encountered within a triangle formed by three consecutive processors q_{i-1}, q_i, and q_{i+1}, the centroid, say C, of the obstacle points lying within the triangle is computed. The line segments joining the processors q_{i-1} and q_{i+1} to C partitions the obstacle space within the triangle into two disjoint parts. The convex hull of the obstacle points lying within the partition adjacent to the processor q_i is computed [see Fig. 5(a)]. A new processor is introduced and initialized near each vertex of the convex hull[1] [see Fig. 5(b)]. The indices of all processors and their corresponding weights are updated. The processor q_i is considered as "useless" and is deleted. A similar notion of useless units has been used in [10].

We claim that this extension of STON works correctly in all cases. Let us, for a contradiction, assume that there exists an

[1]See Appendix II for details of how we introduce processors near each vertex of the convex hull.

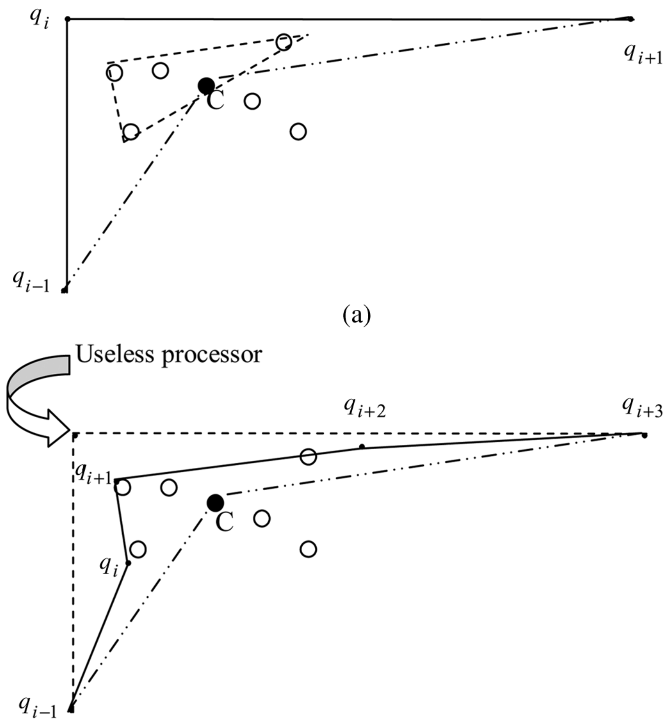

Fig. 5. C is the centroid of all obstacle points lying within the triangle formed by q_{i-1}, q_i, and q_{i+1}. (a) Convex hull of the obstacle points lying in the partition, formed by the line segments joining the processors q_{i-1} and q_{i+1} to C, adjacent to q_i, is shown. (b) Introduction of the new processors q_i, q_{i+1}, and q_{i+2} near the vertices of the convex hull.

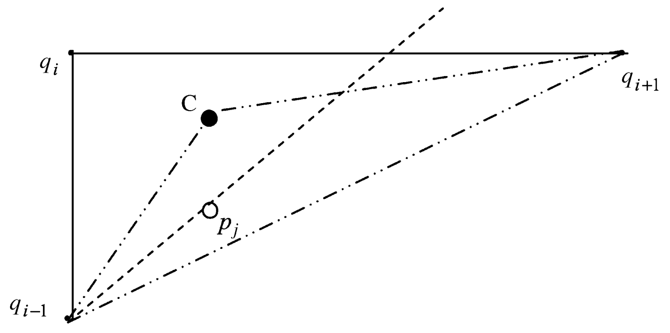

Fig. 6. Shortest homotopic path cannot pass through any obstacle point lying in the partition, formed by line segments joining processors q_{i-1} and q_{i+1} to C, not adjacent to q_i.

obstacle in the partition not adjacent to the processor q_i [see Fig. 5(a)] at which a processor has to be introduced in order to obtain the shortest homotopic path. That is, the shortest homotopic path will pass through an obstacle in the partition not adjacent to the processor q_i. Let p_j be such an obstacle point (see Fig. 6). The line from q_{i-1} through p_j partitions the triangle formed by q_{i-1}, q_i, and q_{i+1} into two disjoint parts. If the shortest homotopic path passes through p_j, there cannot exist any obstacle point in the partition, formed by the line from q_{i-1} through p_j, adjacent to q_i. But in that case, the centroid C cannot lie in the partition, formed by the line from q_{i-1} through p_j, adjacent to q_i; hence, a contradiction. If the shortest homotopic path passes through p_j, the centroid C will lie in the partition, formed by the line from q_{i-1} through p_j, not adjacent to q_i. In that case, the obstacle point p_j lies in the partition, formed by line segments joining processors q_{i-1} and q_{i+1} to C, adjacent to q_i. Hence, the claim follows.

B. Algorithm: Extension of STON

```
1.  initialize the weights
2.  $t \leftarrow 0$
3.  while convergence criteria (5) not satisfied, do
4.    $t \leftarrow t + 1$
5.    for each processor on path, do
6.      $z \leftarrow$ number of obstacles inside triangle formed with neighboring processors
7.      if $z = 0$
8.        create feature vector (2)
9.        update weight (4)
10.     if $z = 1$
11.       select feature vector (3)
12.       update weight (4)
13.     if $z > 1$
14.       compute convex hull of the selected obstacles
15.       introduce new processors and update their weights.
```

First, we note that the algorithm for the original version of STON comprised of steps 1–12 of the previously described algorithm. Computational complexity of step 6 is $O(\log n + m)$ where n is the number of obstacle points and m is the number of obstacle points inside the triangle formed by a processor and its adjacent neighbors, $0 \leq m \leq n$. This complexity can be achieved by a one-time construction of a 2-D range tree of the obstacle points in $O(n \log n)$ time. Querying the tree requires $O(\log n + m)$ time using fractional cascading [20]. On average, $m = (n/k)$ where k is the number of processors. Thus, complexity of STON is $O(\log n + n/k)$ per processor per sweep, assuming the input path has been sampled at half the minimum distance between the obstacles. The purpose of extending STON is to eliminate the constraint on sampling. As a result, steps 13–15 had to be introduced which use the algorithm recursively for computing convex hull. Let $T(n)$ be the complexity of extension of STON for each sweep and k be the number of processors at the end of a sweep. Then, from the previous algorithm, we get

$$T(n) = k\left(r_c T(m) + \log n + m\right) \tag{6}$$

where r_c is the number of sweeps required to compute convex hull. The convex hull is computed to determine the number and locations of new processors that have to be introduced so that there does not exist more than one obstacle in any triangle formed by three consecutive processors. For this purpose, it is sufficient to compute just one sweep of the convex hull instead of a tight convex hull. This strategy saves computational costs as the newly added processors will eventually not remain on the convex hull of the obstacles but will remain on the path. Therefore

$$T(n) = k\left(T\left(\frac{n}{k}\right) + \log n + \frac{n}{k}\right) = O\left(n(\log k + \log_k n)\right) \tag{7}$$

Thus, the complexity of extension of STON is $O((n/k)(\log k + \log_k n))$ per processor per sweep. As the number of processors (k) increases, $m \rightarrow 1$ and the complexity

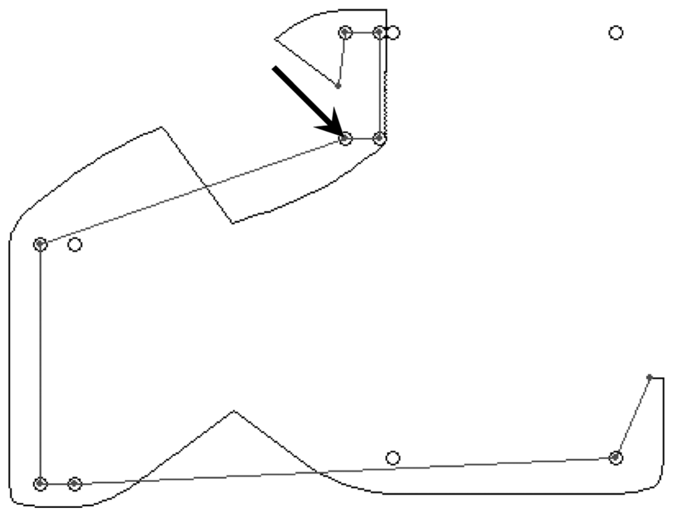
Fig. 7. STON works incorrectly for large values of α. Circles represent point obstacles, while dark and light lines represent initial and tightened configurations of a path respectively.

of extension of STON becomes $O(\log n)$. Thus, the extension of STON starts with a complexity of $O((n/k)(\log k + \log_k n))$ and reaches a complexity of $O(\log n)$ when no more processors are required to be introduced. It is interesting to note that the complexity of STON is comparable with that of some of the recently proposed variants of SOM [9], [21].

In computational geometry, many researchers have proposed algorithms to solve this problem with the primary goal of minimizing computational complexity (refer to [11] for a detailed review). Efrat *et al.* [22] and Bespamyatnikh [23] have independently proposed output-sensitive algorithms for the problem. Their algorithms tackle the problem for simple and nonsimple paths in different ways, with the one for nonsimple paths having higher complexity. Bespamyatnikh's algorithm for nonsimple paths achieves $O(\log^2 n)$ time per output vertex. If the terminal points of a given path are not fixed, the resulting problem is NP-hard [24] which has not been dealt with in this paper.

V. Experimental Results

In this section, we present experimental results obtained by deploying STON to different data sets for different purposes. The extension of STON has been used to compute shortest homotopic paths, smooth paths, and convex hulls. In Fig. 7, the performance of STON is illustrated assuming α is assigned a large value close to unity when feature vectors are chosen according to (3). In that case, the algorithm might fail to perform optimally as the final path might cling to undesired obstacles, as shown by the arrow in Fig. 7. This happens because once a processor falls on an obstacle, which might happen for some processors before convergence if α is large, the processor fails to let the obstacle go as the obstacle continues to remain within its triangle and the processor has no memory of which direction it proceeded from. Such performance from STON can only be averted by choosing α much smaller than unity when feature vectors are selected by (3). This provides ample time for the processors to distribute themselves along the path before coming

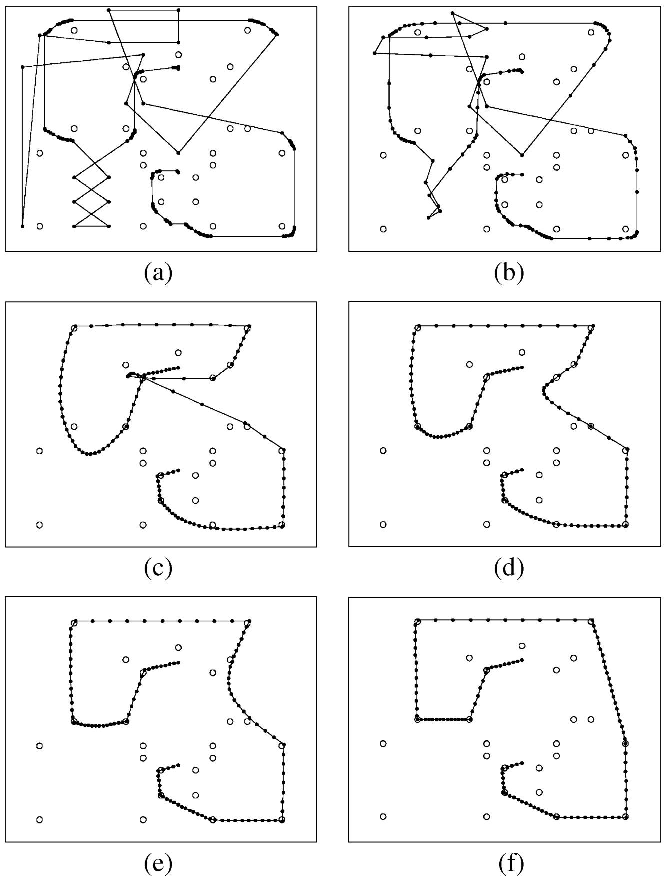

Fig. 8. STON applied to a nonsimple path that turns out to be simple when tightened. Circles denote point obstacles while dots denote locations of the processors on the path. (a) At initialization. (b) After first sweep. (c) After 5 sweeps. (d) After 10 sweeps. (e) After 15 sweeps. (f) After 20 sweeps.

close to any obstacle. For our experiments, α was chosen as follows:

$$\alpha(t) = \begin{cases} 1.0, & \text{if feature vector is created} \\ \beta\left(1 + \frac{t}{T}\right), & \text{if feature vector is selected} \end{cases} \quad (8)$$

where β is the learning constant, $0 < \beta < 0.5$, and T is the total number of sweeps that STON is expected to converge within. Typically, β and T are assigned values 0.01 and 5000, respectively. Thus, initially, a processor proceeds slowly towards the chosen obstacle but the rate of the proceeding increases as more and more sweeps are computed. This prevents STON from converging at suboptimal solutions. Throughout our experiments, ϵ is chosen to be 0.001% of the maximum distance covered along any one dimension by the obstacles.

Fig. 8(a) illustrates a complicated configuration of a path that has not been sampled uniformly. STON was applied to shorten this path and the configurations reached after 1, 5, 10, 15, and 20 sweeps are shown in Fig. 8. It can be seen that the processors gradually distribute themselves evenly along the path due to their weights being updated with respect to created feature vectors (2). This and the assignment of α according to (8) help STON avoid suboptimal convergence which could have been the case after 10 sweeps [see Fig. 8(d)]. The correct shortest configuration was reached within 20 sweeps.

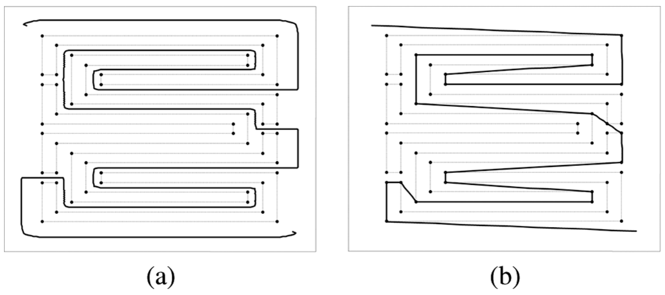

Fig. 9. Performance of STON in a structured obstacle environment. Dotted lines represent the contours of the obstacles while firm lines represent paths. (a) At initialization. (b) After 15 sweeps.

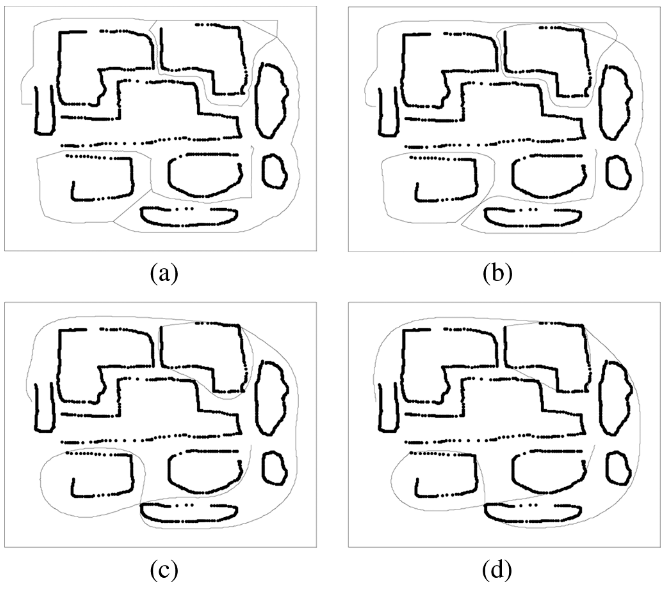

Fig. 10. STON finds a smooth homotopic path with respect to a large set (thousands) of obstacles. Dots represent point obstacles while firm lines represent paths. (a) At initialization. (b) After first sweep. (c) After 10 sweeps. (d) After 20 sweeps.

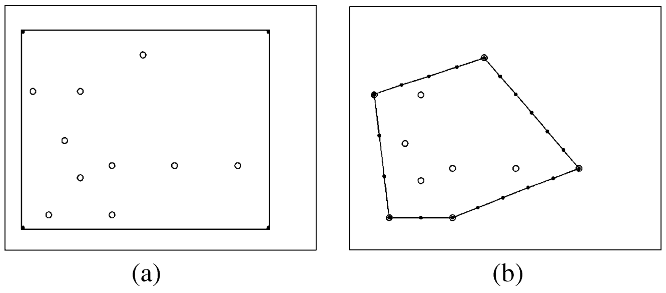

Fig. 11. STON computes the convex hull of a set of points represented by circles. Lines represent the contour of the convex hull while dots on the contour represent locations of the processors. (a) At initialization. (b) After 10 sweeps.

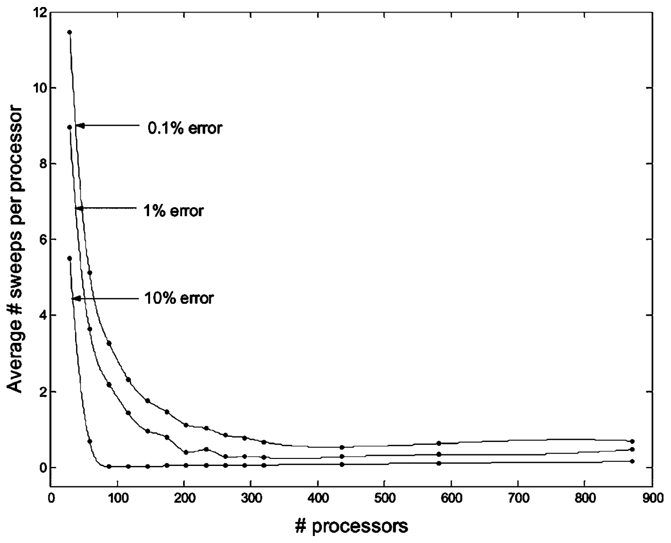

Fig. 12. Experiments show that average number of sweeps per processor required for convergence using STON decreases with the increase in number of processors.

Fig. 9 illustrates the performance of STON in a structured environment where obstacles are not point objects but are 2-D shapes. The algorithm converged within 15 sweeps. In a structured environment, STON considers the points forming the 2-D shapes as point obstacles and does not use their connectivity information. The algorithm performs equally well in the structured as well as unstructured environments.

In real-world applications, such as navigational planning of mobile robots [15] or route formation for military planning [13], the absolute shortest path is not always desired; a suboptimal path that is devoid of sharp turns is often more desirable in such cases. Fig. 10 shows the capability of STON to produce such paths by appropriately adjusting the parameter ϵ. In this case, ϵ was chosen to be 0.1%. The illustration in Fig. 10 further demonstrates the capability of STON to handle a large number of obstacles, in the range of a few thousand.

STON can be used to compute the convex hull of a set of points, as shown in Fig. 11. A path has a starting and an ending point which are fixed and common for all paths belonging to the same homotopy class. To exploit this information for computation of the shortest homotopic paths, the first and the last processors, q_1 and q_k, on a path with k processors were assumed to be fixed and their corresponding weights were never updated. Computation of convex hull, however, does not require any fixed processors, so weights corresponding to all processors were updated. The starting and ending points were assumed to be the same. Such a modification of STON makes it functionally similar to an elastic band or a snake [25].

Experiments with a number of different data sets, a few of which are shown in Figs. 7–11, assuming the learning constant β to be 10^{-2}, reveal certain characteristics of STON. It is expected that the total number of sweeps required for convergence increases with the increase in number of processors. Outcomes of our experiments satisfy such expectations but they also show that average number of sweeps per processor required for convergence decreases with the increase in number of processors (see Fig. 12). This is important in determining how many processors to sample a path with as one should choose the optimum number of processors for minimizing computational costs. The errors in Fig. 12 refer to the ratio of the length of the shortened path at convergence with respect to the length of the shortest path.

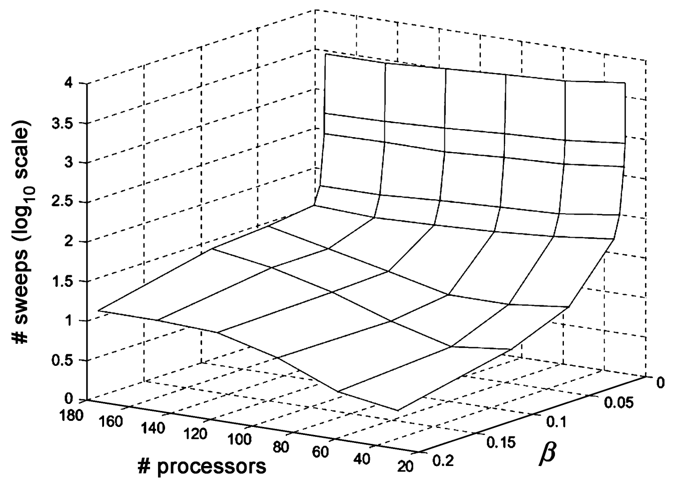

Fig. 13. As the learning constant decreases, the number of sweeps required for convergence using STON increases.

The learning rate is an important factor for ensuring faster convergence. For a fixed number of processors, the total number of sweeps required for convergence increases with decrease in learning constant β (see Fig. 13). We experimented with a number of data sets including those shown in Figs. 7–11, sampling the paths with 30, 59, 88, 117, 146, and 175 processors at different locations and varying the learning constant β from 10^{-4} to 0.2 for each data set. Choosing a very high learning constant might lead to suboptimal results, as has been illustrated in Fig. 7. It is interesting to note from Fig. 13 that for the low learning constants, such as 10^{-2} or lower, the total number of sweeps required for convergence is more for the lower number of processors. This observation only reinforces the fact that average number of sweeps per processor decreases with the increase in number of processors for low learning constants.

VI. Conclusion

A self-organizing neural network algorithm STON is proposed that models the phenomenon undergone by the particles forming a string when the string is tightened from one or both of its ends amidst obstacles. Discussions of the properties and correctness of this anytime algorithm is presented assuming the given string is sampled at a frequency of at least $d/2$ where d is the minimum distance between the obstacles. It is shown how STON might be extended for tightening strings when the previous constraint on sampling is not met. This extension is applied to compute the shortest homotopic path with respect to a set of obstacles. Proof of correctness and computational complexity of the extension of STON are included. Experimental results show that the proposed algorithm works correctly with both simple and nonsimple paths in reasonable time as long as the constraints for correctness are met. STON is used to generate smooth and shorter homotopic paths, a problem that can be modeled as the phenomenon of tightening a string. STON is also used as an elastic band for computing convex hulls. Future research aims at improving the computational complexity of the extension of STON and using it to solve more problems that can be mapped into the problem of tightening a string or an elastic band.

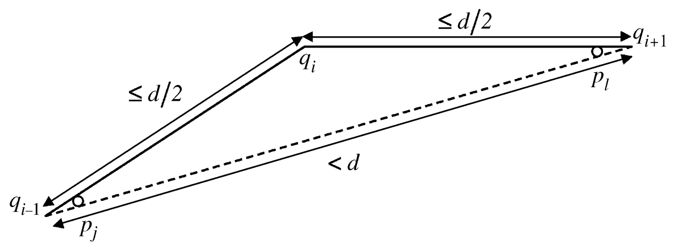

Fig. 14. Sampling a path uniformly at half the minimum distance between the obstacles guarantees at most one obstacle in any triangle.

Appendix I
A Finite Sampling Theorem

A string in $\Re^2$, wound around point obstacles, can be finitely sampled in such a way so as to guarantee only one obstacle within the triangular area spanned by any of the three consecutive points on the string. The following theorem states the constraint necessary to be imposed on the sampling.

Theorem: Sampling a string at half the minimum distance between the obstacles guarantees at most one obstacle in any triangle formed by three consecutive points on the string.

Proof: Let d be the minimum distance between any two unique obstacles in P, P being the set of point obstacles. Let us sample the string such that the distance between any two consecutive points on the string is at most $d/2$. Since d is finite, clearly this leads to a finite sampling of the string. The theorem claims that this sampling ensures that a triangle formed by any of the three consecutive points on the string will never contain more than one unique obstacle.

For contradiction, let us assume that there exist two obstacle points in a triangle formed by three consecutive points q_{i-1}, q_i, and q_{i+1} (see Fig. 14). The segments $\overline{q_{i-1}q_i}$ and $\overline{q_iq_{i+1}}$ included in the string that form two sides of the triangle are each of length at most $d/2$. Thus, the maximum distance between any two points lying within the triangle is less than d. But the distance between any two obstacle points is at least d. Hence, a contradiction, and the claim follows. ■

Appendix II

Here, we describe the procedure for introducing processors near each vertex of the convex hull in the extension of STON. The newly introduced processor, say q_i, should be placed at a location such that connecting it with the neighboring processors q_{i-1} and q_{i+1}does not alter the homotopy class of the path, i.e., the path in which the new processors are being introduced should remain in the same homotopy class as the given path. This is not a trivial task, as illustrated in Fig. 15, where all the processors are introduced near the convex hull but the new path π_{new} is not homotopic to the old path π_{old}.

In order for the new path to remain in the same homotopy class as the old one, processors cannot be introduced within the convex hull and lines joining consecutive processors cannot intersect the edges of the convex hull. We claim that if processors are introduced outside the convex hull in the regions bounded by the extended adjacent edges of the convex hull, then the lines joining the consecutive processors will not intersect with the edges of the convex hull.

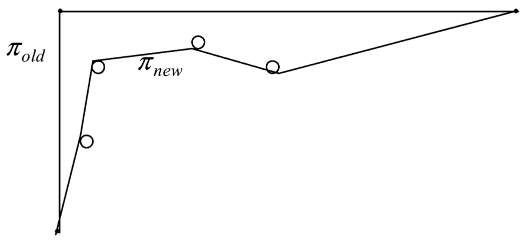

Fig. 15. Introducing processors anywhere near the vertices (shown by circles) of the convex hull does not guarantee that the new path will be homotopic to the old one.

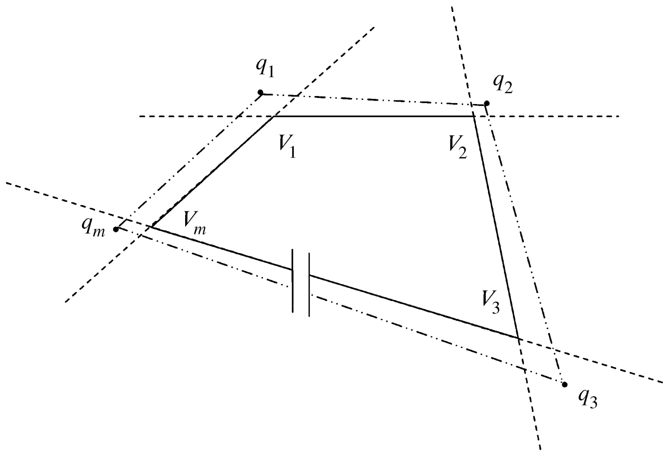

Fig. 16. Introduction of processors outside the convex hull in the regions bounded by the extended adjacent edges of the convex hull guarantees that the lines joining the consecutive processors will not intersect the edges of the convex hull.

Proof: Let $V_1V_2V_3\ldots V_m$ be an m-sided polygon which is the convex hull for a set of obstacles under consideration (see Fig. 16). Let q_i be a processor in the region formed by extensions of adjacent edges $\overline{V_{i-1}V_i}$ and $\overline{V_{i+1}V_i}$, $\forall i$, $1 \leq i \leq m$. The claim states that there cannot be an intersection between the line segment $\overline{q_jq_{j+1}}$ and any edge of the convex hull.

Let us assume, for a contradiction, that there exists at least one intersection between the line segment $\overline{q_jq_{j+1}}$ and an edge, say $\overline{V_kV_{k+1}}$, of the convex polygon $V_1V_2V_3\ldots V_m$. Then, q_j and q_{j+1} must lie on the opposite sides of the extended line segment $\overline{V_jV_{j+1}}$. But by construction, the processors q_j and q_{j+1} lie on the same side of the extended line segment $\overline{V_jV_{j+1}}$. Hence, the contradiction; and the claim follows. ■

REFERENCES

[1] T. Kohonen, *Self-Organizing Maps*. Berlin, Germany: Springer-Verlag, 2001.

[2] J. A. Kangas, T. Kohonen, and J. Laaksonen, "Variants of self-organizing maps," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 93–99, Jan. 1990.

[3] B. Fritzke, "Growing cell structures—a self-organizing network for unsupervised and supervised learning," *Neural Netw.*, vol. 7, no. 9, pp. 1441–1460, 1994.

[4] D. Choi and S. Park, "Self-creating and organizing neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 4, pp. 561–575, Jul. 1994.

[5] L. Schweizer, G. Parladori, L. Sicuranza, and S. Marsi, "A fully neural approach to image compression," in *Artificial Neural Networks*, T. Kohonen, K. Makisara, O. Simula, and J. Kangas, Eds. Amsterdam, The Netherlands: North-Holland, 1991, pp. 815–820.

[6] K. Obermayer, H. Ritter, and K. Schulten, "Large-scale simulations of self-organizing neural networks on parallel computers: Application to biological modeling," *Parallel Comput.*, vol. 14, pp. 381–404, 1990.

[7] F. Favata and R. Walker, "A study of the application of Kohonen-type neural networks to the traveling salesman problem," *Biol. Cybern.*, vol. 64, pp. 463–468, 1991.

[8] H. J. Ritter and T. Kohonen, "Self-organizing semantic maps," *Biol. Cybern.*, vol. 61, pp. 241–254, 1989.

[9] H. Kusumoto and Y. Takefuji, "$O(log_2M)$ self-organizing map algorithm without learning of neighborhood vectors," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1656–1661, Nov. 2006.

[10] B. Fritzke, "A self-organizing network that can follow non-stationary distributions," in *Proc. Int. Conf. Artif. Neural Netw.*, 1997, pp. 613–618.

[11] J. S. B. Mitchell, "Geometric shortest paths and network optimization," in *Handbook on Computational Geometry*, J. R. Sack and J. Urrutia, Eds. New York: Elsevier, 2000, pp. 633–702.

[12] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, MA: MIT Press, 2005.

[13] B. Chandrasekaran, U. Kurup, B. Banerjee, J. R. Josephson, and R. Winkler, "An architecture for problem solving with diagrams," in *Diagrammatic Representation and Inference, Lecture Notes in Artificial Intelligence*, A. Blackwell, K. Marriott, and A. Shimojima, Eds. Berlin, Germany: Springer-Verlag, 2004, vol. 2980, pp. 151–165.

[14] S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, and W. Rlling, "On continuous homotopic one layer routing," in *Proc. Symp. Comput. Geometry*, 1988, pp. 392–402.

[15] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and robot control," in *Proc. IEEE Int. Conf. Robot. Autom.*, Atlanta, GA, 1993, vol. 2, pp. 802–807.

[16] D. Grigoriev and A. Slissenko, "Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane," in *Proc. Int. Symp. Symbolic Algebraic Comput.*, Rostock, Germany, 1998, pp. 17–24.

[17] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.

[18] E. Berglund and J. Sitte, "The parameterless self-organizing map algorithm," *IEEE Trans. Neural Netw.*, vol. 17, no. 2, pp. 305–316, Mar. 2006.

[19] T. DeWolf and T. Holvoet, "Emergence versus self-organisation: Different concepts but promising when combined," in *Engineering Self Organising Systems: Methodologies and Applications*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2005, vol. 3464, pp. 1–15.

[20] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*. New York: Springer-Verlag, 1997.

[21] S. Pal, A. Datta, and N. R. Pal, "A multilayer self-organizing model for convex-hull computation," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1341–1347, Nov. 2001.

[22] A. Efrat, S. G. Kobourov, and A. Lubiw, "Computing homotopic shortest paths efficiently," in *Proc. 10th Ann. Eur. Symp. Comput. Geometry*, 2002, pp. 411–423.

[23] S. Bespamyatnikh, "Computing homotopic shortest paths in the plane," *J. Algorithms*, vol. 49, no. 2, pp. 284–303, 2003.

[24] D. Richards, "Complexity of single-layer routing," *IEEE Trans. Comput.*, vol. C-33, no. 3, pp. 286–288, Mar. 1984.

[25] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes active contour models," *Int. J. Comput. Vis.*, pp. 321–331, 1988.

Bonny Banerjee (S'06) received the B.E. degree in electronics and telecommunication engineering (with honors) from Jadavpur University, Calcutta, India, in 2000, the M.S. degree in electrical engineering, and the Ph.D. degree in computer science and engineering from The Ohio State University, Columbus, in 2002 and 2007, respectively.

From 2001 to 2007, he was a Graduate Research Associate at the Laboratory for AI Research at The Ohio State University. His current research interests include AI, neural networks, and computer vision.

Dr. Banerjee was a recipient of the M. V. Chauhan Merit Certificate awarded by the IEEE India Council in 1999, the J. N. Tata Scholarship for higher studies abroad in 2000–2001, and the Edward J. Ray Travel Award for scholarship and service from The Ohio State University in 2004 and 2006. He received the National Scholarship, India, in 1994 and 1996.