ELSEVIER

Journal of Systems Architecture 42 (1996/97) 743–760

JOURNAL OF SYSTEMS ARCHITECTURE

Sub-optimal solutions to track detection problem using graph theoretic concepts

Stavros D. Nikolopoulos *, George Samaras

Department of Computer Science, University of Cyprus, 75 Kallipoleos Street, CY-1678 Nicosia, Cyprus

Abstract

This paper demonstrates how the problem of tracking targets, which appear as either straight or curved lines in two-dimensional display images (or data images) can be formulated in terms of a directed weighted graph model and how dynamic programming techniques can be efficiently applied to reach an optimal or sub-optimal solution. In general, track detection algorithms providing optimal solutions have good detective ability, but most of them suffer from the inability to detect discontinuous lines or to resolve efficiently pairs of crossing lines. A sub-optimal solution is provided that efficiently overcomes these weaknesses. We focus on modeling the track detection problem in terms of a graph, formulating fast sequential/parallel sub-optimal track detection algorithms and testing them on simulated data in order to show their detective ability. Moreover, we specify the conditions under which sub-optimal algorithms can perform at least as well as their corresponding optimal algorithms. This is significant for the track detection problem where fast, accurate and real-time detection is considered a necessity.

Keywords: Track detection; Display images; Graph modeling; Shortest paths; Dynamic programming; Parallel implementation

1. Introduction

An important topic in diverse fields (radar, sonar, radio-astronomy, etc.) is the detection of targets, which appear as either straight or curved lines in two-dimensional display images or data images. The track detection problem is simply one of trying to locate particular line patterns embedded in a noisy background in display images. Several algorithms have been published in the literature looking at this problem from different points of view [2,4,11,12,16,18,20]. Recently, a number of papers have been published on this subject using graph theoretic approaches [5,10,13,21,22].

It is well-known that, any problem that can be expressed in terms of a graph can be solved using graph theoretic techniques. In signal images, one available aspect is the intensity of each pixel. Therefore, an obvious straightforward way to represent an image by a graph is as follows: the information of

* Corresponding author. Email: stavros@cs.uoi.gr

PII S1383-7621(96)00074-4

time history records are mapped onto a weighted graph such that the vertices of the graph would correspond to the individual pixels of the display image. Specifically, each pixel of the image becomes a vertex in the graph with edges between adjacent pixels. Under this formulation, the problem of detection tracks is reduced to the problem of finding a set of vertices in the graph which minimizes some cost function.

In this paper the track detection problem is formulated in the above mentioned way. The key to the solution is that any path through the graph, i.e., any set of vertices, optimizing some cost function generates a line. First, based on a well-known dynamic programming solution to this problem (Viterbi algorithm [3,4,16]), we developed a line detection algorithm, which we call OSP (Optimal Survivor Paths), for finding the k best completely unmerged paths through a graph. Both algorithms, Viterbi and OSP, are characterized by global optimality with respect to the cost function used. In general, the complete globally optimal algorithms based on the dynamic programming approach have good detective ability, but most of them suffer from the inability to detect discontinuous tracks or to resolve efficiently pairs of crossing tracks [21]. In these cases, instead of a complete globally optimal solution, a sub-optimal solution which may overcome the above weakness is preferable. Based on this fact, we formulate sub-optimal algorithms for the track detection problem and we show their detective ability by testing them on simulated data.

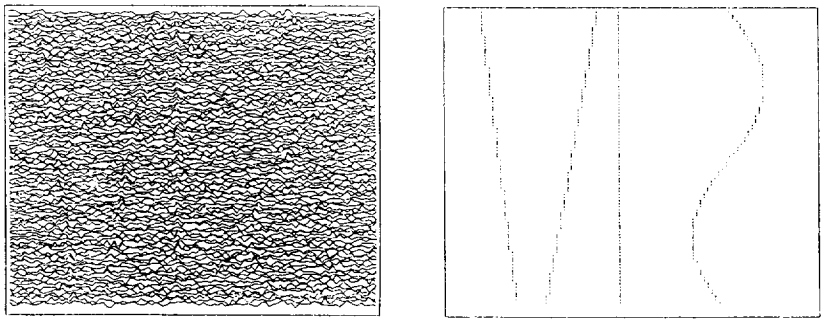
Besides detective ability we are also interested in time efficiency as well. The sub-optimal algorithms, in practice, provide better time efficiency than the globally optimal ones. Our sub-optimal formulation offers itself nicely to parallelism. Thus, parallel implementations are also shown for the sub-optimal and optimal detection algorithms under both the CREW and CRCW computational PRAM models [1,9,15,17,19].

The remainder of the paper is organized as follows: Section 2 describes the track detection problem, the formulation of the problem and the proposed solution approaches. Section 3 presents the appropriate graph model and provides an introduction to graph theoretic tools used throughout the paper. Section 4 gives the graph transformation of the problem and the cost function used. The description of two optimal algorithms for single and multiple track detection is cited in Section 5 and Section 6, respectively. Sub-optimal solutions are given in Section 7. Section 8 shows that our graph theoretic approach to the track detection problem is ideally suited for parallel computation, and provides parallel implementations of the proposed optimal and sub-optimal detection algorithms. Section 9 gives a clear indication of the detective ability of the proposed algorithms by testing them on the same set of simulated data. It also shows the conditions under which the optimal and sub-optimal algorithms may detect identically. Finally, Section 10 concludes the paper.

2. Track detection problem, formulation and solutions

In broader terms the problem we consider is that of image-processing/feature-extraction. Such a problem has applications in the area of automated target detection and tracking; an area quite significant and evident in systems such as radar, sonar, radio-astronomy, etc. It is well-known that in such systems, operators are in danger of being overwhelmed by display data. This is indeed a real problem considering the size and complexity of these systems. Thus, there is obviously a need for track detection algorithms that support not only automatic but also a fast and accurate detection of tracks to assist operators in their task. They usually see noisy images, while what they actually need is their noiseless representation, that is, they prefer the image of Fig. 1(b) instead of that of Fig. 1(a).

We have referred to the track detection problem



(a) (b)

Fig. 1. (a) A noisy display image. (b) Its noiseless representation.

as one of trying to locate particular line patterns embedded in a very noisy background. The tracks represented by this patterns, usually appear as roughly vertical lines although they can also appear as carve or discontinuous lines. Moreover, considering those types of lines, a display image might contain parallel or crossing line patterns (see Fig. 5 and Fig. 6). In general, the goal of a track detection algorithm is to extract line patterns of interest from display images that tend to be noisy and of low contrast. Conventional image processing techniques are not appropriate because they have been developed for a different type of images that are characterized by relatively high contrast and possess a two-dimensional shape and object information. Graph theoretic techniques might be more suitable in this context.

In order to identify tracks in display images using graph theoretic concepts, the original image must be mapped onto a weighted graph. Such a graph is composed of a set of vertices connected to each other by edges where the vertices and edges have values associated with them. An obvious mapping is to map each pixel onto a vertex of the graph in a one-to-one relationship with the pixel intensity assigned to the vertex weight. If the edge weights are defined as a function of the weights of the vertices that they join, then the edge weight is a measure of similarity between the two vertices, and hence between the two corresponding pixel intensities. For representational purposes, Fig. 2 shows the mapping of a display (data) image onto a graph; actually a K-trellis graph (see Section 3) [13,14].

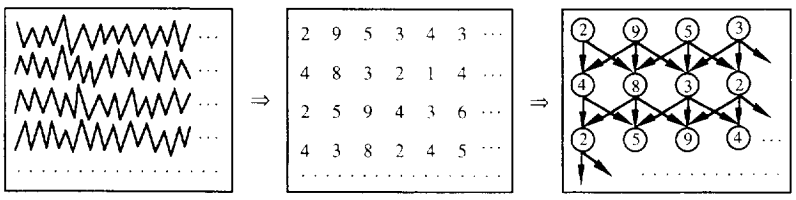


Fig. 2. Transformation of a display image (artificial) onto a K-trellis graph.

Having transformed an image onto a graph, we can view a track as a path through the graph which has particular values for its vertices and edges. Specifically, we can define such a path to be one that minimizes or maximizes a specified cost function. Thus, we can easily transfer the track detection problem to the problem of finding the shortest path through a graph, provided that appropriate values have been assigned to the edges of the graph (provided that a "good" cost function is available). The problem of finding a shortest path trough a graph can be handled in many ways; a popular one is that of dynamic programming. We point out that the idea behind a dynamic programming solution is that small problems are solved and combined to form a larger solution, ideal in our case, considering the structure and the graph representational form of our problem. However, our goal is not just to find a graph representation, but to find one that fits the characteristics of the problem and in a way simulates its structure. It is obvious that, a vertex could be linked to any other vertex, but in our case (see the type of tracks) we observe that it is only necessary to be linked to its nearest specific neighbors. These neighbors can not represent part of the previous or same history line, and as a matter of that they can represent only a specific part of the next history line; they should represent only the closures neighbors of the next history line. This is a valid claim considering the nature of the tracks in the problem we deal with. Formulating our problem in this way leads to a graph representational form of type trellis. Specifically, an appropriate trellis variation that we call K-trellis (see Fig. 3) and which we define in the next section.

Based on the above formulation, we are in a position to say that even for a display image with k tracks on it, the problem of detecting these k tracks in the image can be converted into the problem of finding k shortest paths (we call them optimal survivor paths) through a K-trellis. This approach gives an optimal solution with respect to a cost function used.

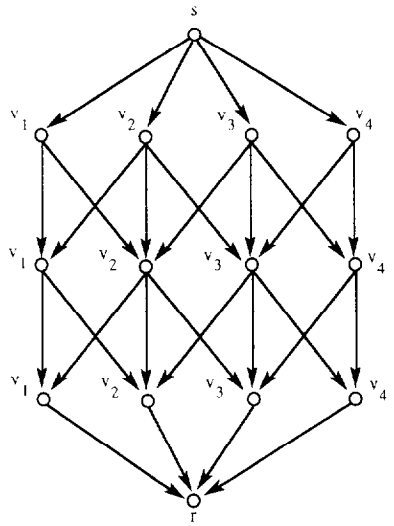


Fig. 3. A K-trellis graph with $T = 3$, $M = 4$ and $K = 3$.

The optimal approach has a major weakness, that is, its inability to detect a pair of crossing or discontinuous lines in the image. A way to overcome these weaknesses is to consider sub-optimal solutions. In this case, the question we should answer is how well the lines selected by a sub-optimal algorithm fit the trajectory models (type of lines in the image) that are given. The algorithms we propose determine local optimal paths through a K-trellis graph, and therefore, the solution given is obviously sub-optimal. Experimental results, show how "near" to the optimal solution (i.e., to the optimal path given by the optimal algorithm), the sub-optimal solution is.

Furthermore, in real systems we are interested in having algorithms that support not only automatic, but also a very fast detection of line patterns. For example, if N is a function of the size of the original graph and its connectivity, then, an $O(N^2)$ complexity algorithm is nowhere near the real-time throughput rate and therefore such an algorithm is practically unable to process more than a limited set of data. A natural way to increase the speed of the detection process is to use an ensemble of proces-

sors. Our graph theoretic approach to the track detection problem is ideally suited for parallel computation.

To give a feeling of the difference in time efficiency of the previously described approaches, i.e., optimal, sub-optimal and parallel, let us summarize the asymptotic time and processor complexity of the algorithms materializing these approaches. More detailed analysis can be found in later sections. Let $T \cdot M$ be the number of vertices of a K-trellis graph, where $K = 3$. The overall asymptotic time complexity of the optimal algorithm which detect one track is $O(K \cdot T \cdot M)$, while it becomes $O(K \cdot T \cdot M^2)$ when it detects more than one tracks. The overall asymptotic time complexity of the sub-optimal algorithms is $O(K \cdot T \cdot M^2)$. In real-time processing the time complexity is reduced to $O(K \cdot L \cdot M^2)$, where L is small enough (in our case $L = 6$). The overall time and processor complexities of the parallel optimal algorithm are $O(T \cdot \log K + \log M)$ and M^2 respectively, $3 \le K \le M$, when it is executed on a CREW-PRAM. As far as the model CRCW-PRAM is concerned, this algorithm can be executed either in time $O(T + M)$ with $\max\{K^2 \cdot M, M^2\}$ processors or in time $O(T)$ with M^3 processors. Finally, the parallel sub-optimal algorithms can be executed in time $O(L \cdot \log K + \log M)$ with $T/L \cdot M^2$ processors on a CREW-PRAM model. When a CRCW-PRAM computational model is available, these algorithms require $O(L + M)$ time and $\max\{T/L \cdot K^2 \cdot M, T/L \cdot M^2\}$ processors or $O(L)$ time and $T/L \cdot M^3$ processors.

3. Graph modeling

In this section, we introduce some graph theoretic concepts and we establish the notation and terminology we shall use throughout this paper, and apply these concepts in the track direction problem.

3.1. Trellis and K-trellis graphs

A *directed graph* $G = (V, E)$ is a structure consisting of a finite set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and a finite set of edges $E = \{(v_i, v_j) \mid v_i, v_j \in V \text{ and } v_i \ne v_j\}$, where each edge is an ordered pair.

We define a *trellis* as a directed graph $G = (V, E)$ with vertices and directed edges that satisfies the following conditions:

(i) The vertex set V is partitioned into T (mutually disjoint) subsets $V_1, V_2, \ldots, V_T$, such that $|V_i| = |V_j| = M$, $1 \le i, j \le T$.
(ii) Edges connect vertices only of consecutive subsets V_t and V_{t+1}, i.e., if $(v_i, v_j) \in E$, then $v_i \in V_t$ and $v_j \in V_{t+1}$, $1 \le t < T$.

The magnitude T we shall call *depth* of the trellis. A *K-trellis* is a trellis graph with two additional properties:

(i) It has two more vertices $s \in V_0$ and $r \in V_{T+1}$, such that $(s, v_i) \in E$, for every $v_i \in V_1$ and $(v_j, r) \in E$, for every $v_i \in V_T$, $1 \le i, j \le M$.
(ii) The vertex v_i of the set V_t is connected (where possible) with $K = 2g + 1$ vertices $\{v_{i-g}, \ldots, v_i, \ldots, v_{i+g}\}$ of the set V_{t+1}, where $1 \le i \le M$, $1 \le t < T$ and $g = 1, 2, \ldots, (M-1)/2$.

The depth of a K-trellis graph will be equal to $T + 2$. Fig. 3 presents a K-trellis with $T = 4$, $M = 4$ and $K = 3$. Let $G = (V, E)$ be a K-trellis graph with $|V| = T \times M$. We shall denote by $V_{t,t'}$ the union of $t' - t + 1$ consecutive vertex subsets $V_t, V_{t+1}, \ldots, V_{t'}$ of the graph, i.e.,

$$V_{t,t'} = V_t \cup V_{t+1} \cup \ldots \cup V_{t'},$$

$$0 \le t < t' \le T.$$

Given a subset $V_{t,t'} \subseteq V$ of vertices, we define the subgraph *induced* by $V_{t,t'}$ to be $G(V_{t,t'}) = (V_{t,t'}, E(V_{t,t'}))$, where $E(V_{t,t'}) = \{(v_i, v_j) \mid (v_i, v_j) \in E \text{ and } v_i, v_j \in V_{t,t'}\}$. Based on this definition, if $G = (V, E)$ is a K-trellis graph with $|V| = T \cdot M$, then $G(V_{1,T})$ is a trellis graph.

A *walk* in a K-trellis is an alternating sequence of vertices and edges, i.e., $P = [v_1, (v_1, v_2), v_2, \ldots, (v_{k-1}, v_k), v_k]$. The *length* $L(P)$ of a walk is the number of edges in it. A *path* is a walk in which all vertices are distinct. For simplicity, we shall denote the path P by $P = \{v_1, v_2, \ldots, v_k\}$ and we shall refer to v_1 and v_k as *first* and *last* vertices of P, respectively.

3.2. Weights and metrics

In each vertex $v_i \in V_t$ of a K-trellis graph we associate two numbers, which we call *vertex weight* and *vertex metric* and we denote by:

$b_t(v_i)$: vertex weight,

$Q_t(v_i)$: vertex metric,

where $1 \le t \le T$. Sometimes we shall denote the vertex weight of $b_t(v_i)$ by $b_t(i)$ or $b(i)$. In a similar way, in each edge $(v_i, v_j) \in E$, $v_i \in V_t$ and $v_j \in V_{t+1}$, we associate two numbers, which we call *edge weight* and *edge metric* and we denote by

$w_t(v_i, v_j)$: edge weight,

$D_t(v_i, v_j)$: edge metric,

where $1 \le t \le T$, $1 \le i, j \le M$. As with vertices, in some cases we shall denote the edge weight $w_t(v_i, v_j)$ by $w_t(i, j)$ or $w(i, j)$ or w_{ij}.

Given a path P in a K-trellis, we define the *path metric* $F(P)$ to be the sum of the vertex weights in the path P, i.e.,

$$F(P) = \sum_{v_i \in P} b(v_i),$$

where $v_i \in V_i$ and $1 \le i \le T$.

3.3. Edge and path cost

In addition to the above defined edge weights and metrics, in each edge $(v_i, v_j) \in E$ of a K-trellis, $v_i \in V_t$ and $v_j \in V_{t+1}$, we associate a third number, which we call *edge cost* and denote by $c_t(v_i, v_j)$ or $c_t(i, j)$ or $c(i, j)$, $1 \le t \le T$ and $1 \le i, j \le M$. In this paper the cost of each edge is given by

$$c(v_i, v_j) = Q(v_i) - D(v_i, v_j),$$

$$\forall v_i \in V_t \text{ and } \forall (v_i, v_j) \in E. \tag{1}$$

$t = 1, 2, \ldots, T$, where Q and D will be computed in Section 9. We state here that, in all cases the vertex metrics of vertices $s \in V_0$ and $r \in V_{T+1}$, as well as the edge metrics of edges $(s, v_i) \in E$ and $(v_j, r) \in E$ are 0, $\forall v_i \in V_1$ and $\forall v_j \in V_T$, i.e.,

$$Q(s) = Q(r) = 0,$$

$$D(s, v_i) = D(v_j, r) = 0 \quad \forall v_i \in V_1 \text{ and } \forall v_j \in V_T.$$

Let $P = \{v_1, v_2, \ldots, v_k\}$ be a path in a K-trellis graph. The *cost* $c(P)$ of a path P through the K-trellis is defined as

$$c(P) = \sum_{(i,j) \in P} c(v_i, v_j),$$

where $c(v_i, v_j)$ is the cost of edge $(i, j) \in E$. The *shortest path* from the vertex v_i to vertex v_j is a path $P = \{v_i, v_{i+1}, \ldots, v_j\}$ with minimum cost.

4. Problem transformation

Each pixel of the display image is mapped onto a vertex of the K-trellis (except vertices s and r) in a one-to-one relationship with the pixel intensity assigned to the vertex weight $b_t(v_i)$, for every $v_i \in E$. It is obvious that the problem of track detection is reduced to the problem of finding a set of vertices in the K-trellis which minimizes some cost function. It is important to point out that the quality of the detection process significantly depends on the cost function used.

Taking the above into consideration, we define a line in an image to be any vertex sequence $\{v_1, v_2,$

$\ldots, v_t\}$ which (i) forms a path $P = \{v_1, v_2, \ldots, v_t\}$, and (ii) minimizes the following cost function

$$c(P) = \sum_{i=1}^{t} c(v_i, v_j),$$

where $v_i \in V_i$ and $2 \le t \le T$.

We formulate the track detection problem as a graph theoretic, shortest path, problem with a well-know dynamic programming solution.

5. Single line detection – Dynamic programming

As stated earlier, a K-trellis diagram will be the graphical representation of a display image. The aim is to find a path $P = \{s, v_1, v_2, \ldots, v_T, r\}$ through the K-trellis (from $s \in V_0$ to vertex $r \in V_{T+1}$), which minimizes the cost function defined in Section 4, i.e.,

$$c(P) = c(s, v_1) + \sum_{i-1}^{T} c(v_i, v_j) + c(v_T, r),$$

where $v_i \in V_i$ and $1 \le i \le T$.

We define $P_{t,i}$ to be the shortest path from vertex $s \in V_0$ to vertex $v_i \in V_t$, with cost $c(P_{t,i})$, where $t = 0, 1, \ldots, T+1$. Starting at time $t = 1$ by

$$c(P_{1,i}) = b_1(v_i)$$

we have

$$c(P_{t,j}) = \min_{j-g \le i \le j+g}\{(P_{t-1,i}) + c(v_i, v_j)\}.$$

Thus, a new extended path to vertex $v_j \in V_t$ has cost, that is, the sum of the shortest path from s to vertex $v_i \in V_{t-1}$, plus the cost of the edge $(v_i, v_j) \in E$.

Recall that the cost $c(v_i, v_j)$ of the edge $(v_i, v_j) \in E$ (see Eq. (1)), is given by

$$c(v_i, v_j) = Q(i) - D(v_i, v_j),$$

$$\forall v_i \in V_t \text{ and } \forall (i, j) \in E.$$

$t = 1, 2, \ldots, T$, where Q and D will be computed in Section 9, and

$$Q(s) = Q(r) = 0,$$

$$D(s, v_i) = D(v_j, r) = 0 \quad \forall v_i \in V_1 \text{ and } \forall v_j \in V_T.$$

Thus,

$$c(P) = \sum_{i=1}^{T} c(v_i, v_j) = \sum_{i=1}^{T-1} c(v_i, v_j) + \{Q_T(v_j)\}$$

$$= Q_1(v_j) + \sum_{i=2}^{T} \{Q(v_i) - D(v_i, v_j)\}.$$

From the last equation we have

$$c(P_{t,j}) = Q_t(v_j) + \min_{j-g \le i \le j+g} \{c(P_{t-1,i}) - D(v_i, v_j)\}$$

which is the key equation for updating the shortest path from $t-1$ to t, $1 \le t \le T$. For each j, the smallest index i for which the minimum is attained, is given by the argmin function

$$y_t(v_j) = \operatorname{argmin}_{j-g \le i \le j+g}\{c(P_{t-1,i}) - D(v_i, v_j)\}.$$

The method described above provides a complete globally optimum track detection in the sense that the path from vertex $s \in V_0$ to vertex $r \in V_{T+1}$ through a K-trellis graph determined is an optimum solution, $T \ge 2$. The algorithm which implements the above method is listed below.

Algorithm DP

1. **for** $j = 1$ to M **do** $C(1, j) = Q(1, j)$; $A(1, j)$ = arbitrary **end**;

2. **for** $t = 2$ to T **do**
  for $j = 1$ to M **do**
    $C(t, j) = Q(t, j) + \min_{j-g \le i \le j+g}\{C(t-1, i) - D(i, j)\}$;
    $A(t, j) = \operatorname{argmin}_{j-g \le i \le j+g}\{C(t-1, i) - D(i, j)\}$;
  end;
end;

3. $P(T) = \operatorname{argmin}_{1 \le j \le M}\{C(T, j)\}$;
for $t = T-1$ downto 1 **do**
  $P(t) = A(t+1, P(t+1))$ **end**;

end.

It is easy to see that Step 1 of the algorithm is executed in time $O(M)$. Step 2 has time complexity $O(K \cdot T \cdot M)$, where $1 \le K \le M$. Clearly Step 3 takes time $O(T)$. Thus, the overall asymptotic time complexity of the algorithm is $O(K \cdot T \cdot M)$.

6. Optimal survivor paths

Let $G(V_{1,\,T})$ be a trellis graph with $|V| = T \cdot M$. Two paths $P_i = \{u_1, u_2, \ldots, u_T\}$ and $P_j = \{u'_1, u'_2, \ldots, u'_T\}$ are said to be *mutually exclusive* or *unmerged* if $u_t \neq u'_t$ for all $t = 1, 2, \ldots, T$; otherwise, they are said to be *merged*. Similarly, two lines in a display image are said to be mutually exclusive or *unmerged* if they can be represented by two mutually exclusive paths in a trellis graph.

Bellman's optimality principle for Dynamic Programming [3] states that if a shortest path from vertex $u_i \in V_1$ to vertex $u_j \in V_T$, passes through vertex $w \in V_t$, $1 < t < T$, then the subpaths from u_i to w and from w to u_j, must also be shortest paths. Based on this principle, we can also state that if P_i is the shortest path from vertex $v_i = u_1 \in V_1$ to vertex $r \in V_{T+1}$, i.e.,

$$P_i = \{u_1, u_2, \ldots, u_{t-1}, w, u_{t+1}, \ldots, u_T, r\}$$

and is P'_j the shortest path from vertex $u'_1 \in V_1$ to vertex $w \in V_t$, i.e.,

$$P'_j = \{u'_1, u'_2, \ldots, w\}$$

then the path

$$P_j = \{u'_1, u'_2, \ldots, w, u_{t+1}, \ldots, u_T, r\}$$

is the shortest path from vertex $u'_1 \in V_1$ to vertex $u \in V_{T+1}$, where $u_i \neq u'_i$, $1 \le i \le t-1$ and $1 < t < T$. It follows that, if P_i and P_j are two merged optimal paths in graph $G(V_{1,\,T})$, i.e.,

$$P_i \cap P_j \neq \varnothing$$

then, paths P_i and P_j have the same last vertex.

We suppose that a display image, represented by a trellis graph $G(V_{1,\,T})$, contains more than one un-merged lines. It is then obvious that, if the optimal paths $P_1, P_2, \ldots, P_q$, in the graph $G(V_{1,\,T})$ are such that $P_1 \cap P_2 \cap \ldots \cap P_q \neq \varnothing$, then the path with minimum cost, say P_i, is more likely to lie along a line in the image. We call a path with the previous property *optimal survivor path*, i.e.,

$$c(P_i) = \min\{c(P_j) \mid j = 1, 2, \ldots, q\}.$$

Based on the above definition, we can claim that two optimal survivor paths P_1 and P_2, i.e., $P_1 \cap P_2 \neq \varnothing$, with minimum cost among all optimal survivor paths in $G(V_{1,\,T})$, are more likely to lie along two unmerged lines in the image.

Given a display image with k lines on it, the problem of detecting the k lines in the image can be converted into the problem of finding k optimal survivor paths $P_1, P_2, \ldots, P_k$, through a trellis which minimizes the total cost

$$J = \sum_{i=1}^{k} c(P_i).$$

We are interested in formulating an algorithm for finding all possible survivor paths in a trellis graph. We can do this by using the optimal DP algorithm. Below we describe an algorithm, which we call OSP (Optimal Survivor Paths), for computation of all possible optimal survivor paths of length $T-1$ in a trellis graph $G(V_{1,\,T})$.

Algorithm OSP

1. **Find** the set S of the M optimal paths $P_1, P_2, \ldots, P_M$, through a trellis $G(V_{1,\,T})$ using optimal algorithm DP, such that:
   (i) path P_i has length $T-1$, and
   (ii) path P_i has first vertex $v_i \in V_1$, $1 \leqslant i \leqslant M$.
2. **Partition** set S into q subsets $S_1, S_2, \ldots, S_q$, $1 \le q \le M$, such that:
   (i) $S = S_1 \cup S_2 \cup \ldots \cup S_q$, $1 \le q \le M$, and
   (ii) set S_i contains all optimal paths which have the same last vertex, $1 \le i \le$

q.

3. **Compute** the optimal survivor path from each subsets $S_1, S_2, \ldots, S_q$, $1 \le q \le M$.
4. **Select** the k best optimal survivor path $P_1, P_2, \ldots, P_k$, $1 \le k \le q$.

end.

Let us now compute the time complexity of algorithm OSP (Optimal Survival Paths). It is easy to see that Step 1 can be executed in time $O(K \cdot T \cdot M^2)$ (see optimal solution of algorithm DP), where $1 \le K \le M$. Obviously, the computation of the sets $S_1, S_2, \ldots, S_q$, $1 \le q \le M$, has time complexity $O(M^2)$. In a sequential machine, the operation of finding the minimum of n numbers required time proportional to n. Therefore, Step 3 of the algorithm requires $|S_1| + |S_2| + \ldots + |S_q| = |S|$ comparison steps, $1 \le q \le M$. Thus, the overall asymptotic time complexity of the algorithm is $O(K \cdot T \cdot M^2)$.

7. Multiple line detection – Sub-optimal solution

Apart from the fact that algorithm OSP takes a considerable amount of computing time since T becomes very large in some applications, it also has a major weakness, that is, its inability to detect a pair of crossing or discontinuous lines in the image. The reason is obvious, looking at the definition of the optimal survivor paths and taking into consideration that algorithm OSP computes the optimal survivor paths of the whole trellis graph, i.e., paths of length T. Our objective here is to study methods for the track detection problem, which overcome this weakness.

7.1. Sub-optimal approach

A way to overcome the above mentioned weaknesses is to consider sub-optimal solutions. By the term "sub-optimal solution" we simply mean that, instead of finding all optimal survivor paths of length T in the whole trellis graph, we find all optimal survivor paths of length L in graphs $G(V_{1,L})$, $G(V_{L,2L})$, $\ldots$, $G(V_{T-L,T})$. In this case, the question we should answer is how well the lines selected by a sub-optimal algorithm fit the trajectory models (type of lines in the image) that are given.

Let $G(V_{1,T})$ be a trellis graph, and P an optimal path in graph $G(V_{i,j})$ which fits a given line in an image represented by the trellis graph $G(V_{1,T})$. Let P_1 and P_2 be two optimal paths in graphs $G(V_{i,t})$ and $G(V_{t+1,j})$, respectively. Suppose that the vertices of the optimal paths P_1 and P_2 lie along P. It is likely that the positions of the vertices of the optimal path P in $G(V_{i,j})$ will be close to those in the set $P_1 \cup P_2$. Therefore, we can compute a sub-optimal path $\{u_i, \ldots, u_j\}$ in graph $G(V_{i,j})$, $i < j$, by computing first an optimal path $\{u_i, \ldots, w_1\}$ in $G(V_{i,t})$, and then an optimal path $\{w_2, \ldots, u_j\}$ in graph $G(V_{t+1,j})$, where $i < t < t+1$ and $(w_1, w_2) \in E$. Consequently, we can compute a sub-optimal path from $s \in V_0$ to $r \in V_{T+1}$ through a K-trellis by computing the optimal paths $\{s, \ldots, u_i\}, \{u_{i+1}, \ldots, u_j\}, \{u_{j+1}, \ldots, u_k\}, \ldots, \{u_r, \ldots, r\}$ in graphs $G(V_{0,i})$, $G(V_{i+1,j})$, $G(V_{j+1,k})$, $\ldots$, $G(V_{r,T+1})$, respectively.

7.2. Sub-optimal detection algorithms

This approach leads us to formulate sub-optimal algorithms which have the potential ability to detect a pair of crossing lines even in the region of the crossing. Additionally, it is obvious that such an algorithm is also able to detect discontinued lines.

Let us now focus on a sub-optimal solution of the track detection problem in order to detect crossing or discontinuous lines in the image. Given a trellis graph $G(V_{1,T})$ with $T \cdot M$ vertices, we partition its vertex set $V_{1,T}$ into T/L subsets $V_{1,L}, V_{L,2L}, \ldots, V_{T-L,T}$, each with $(L+1) \cdot M$ vertices (except the first vertex set $V_{1,L}$, and possible the last one). We assume that $T/L = \lceil T/L \rceil$.

Using algorithm OSP, we can find all possible optimal survivor paths $P_1, P_2, \ldots, P_q$ of each

graph $G(V_{1,L}), G(V_{L,2L}), \ldots, G(V_{T-L,T})$, i.e., all paths of each graph with the property

$P_i \cap P_j \neq \varnothing$,

where $1 \leq q \leq M$, $1 \leq i, j \leq M$ and $i \neq j$.

There are two possible ways to extract lines from the image:

(1) Using a fixed number of extracting lines.
(2) Using a threshold to extract lines.

These approaches lead us to formulate two different track detection sub-optimal algorithms.

Algorithm SubOSP-k

For each graph $G(V_{1,L}), G(V_{L,2L}), \ldots, G(V_{T-L,T})$, do

1. Compute all possible optimal survivor paths.
2. Extract the k best optimal survivor paths.

end.

Algorithm SubOSP-h

For each graph $G(V_{1,L}), G(V_{L,2L}), \ldots, G(V_{T-L,T})$, do

1. Compute all possible optimal survivor paths.
2. Extract all optimal survivor paths, whose cost metric does not exceed a threshold determined experimentally.

end.

Both algorithms determine local optimal paths through a trellis graph, and therefore, the solution given is obviously sub-optimal. Experimental results, presented in Section 9, show how "near" to the optimal solution, i.e., to the optimal path, given by algorithm OSP, the sub-optimal solution given by the described algorithms SubOSP-k and SubOSP-h is.

We should comment in advance that algorithm SubOSP-k is preferable when the number of tracks is known, as well as when the tracks are represented by continuous lines. On the other hand, algorithm SubOSP-h is preferable when the number of lines is unknown, or when the lines are represented by discontinuous lines. The cost paid for the SubOSP-h algorithm's ability is (i) the a priori determination of a threshold, and (ii) the unclear processed image.

8. Parallel implementation

In most applications we are interested in designing algorithms that support not only automatic, but also a very fast detection of line patterns in display images (i.e., lofargrams). A natural way to increase the speed of the detection process is to use an ensemble of processors [4,15]. Obviously, our graph theoretic approach to the track detection problem is ideally suited for parallel computation. Therefore, we turn our attention to the parallel implementation of the optimal and sub-optimal detection algorithms proposed in the previous sections.

8.1. Algorithm OSP

We fist describe a parallel implementation of the optimal algorithm OSP, which hereinafter referred to as Par-OSP.

Algorithm OSP consists of four steps. In Step 1 it computes M optimal paths $P_1, P_2, \ldots, P_M$, through a trellis. This computation can be done by using algorithm DP to find M optimal path P_i from vertex $v_i \in V_1$ to vertex $r \in V_{T+1}$, $i = 1, 2, \ldots, M$. Let us consider algorithm DP. Obviously, Step 1 can be executed in parallel. As far as Step 2 is concerned, we observe that the computation of $C(t, j)$ depends on the value of $C(t-1, j)$, $1 \leq j \leq M$. On the other hand, for each $t = 2, 3, \ldots, T$, the computation of the M values $C(t, 1), C(t, 2), \ldots, C(t, M)$ can be executed independently, and therefore, in parallel. Obviously, the last step of algorithm DP cannot be executed faster, since different iterations of the for-loop cannot be performed in parallel. Let us now

implement Step 2 and Step 3 of the algorithm OPS. We define an $M \times M$ matrix L as follows:

$$L[i, j] = \begin{cases} c(P_j) & \text{if } v_i \text{ is the last vertex of } P_j \\ \alpha & \text{otherwise.} \end{cases}$$

$j = 1, 2, \ldots, M$. (Actually, vertex v_i is the last but one vertex of P_j; the last is the vertex r.) Obviously, path P_m, $1 \le m \le M$, is an optimal survivor path if and only if

(i) $m = \text{argmin}\{L[i, 1], L[i, 2], \ldots, L[i, M]\}$, and
(ii) $L[i, m] \neq \alpha$, $1 \le i \le M$.

Let P_{OSP} be the set of all Optimal Survivor Paths computed in Step 3 of algorithm OSP. This basic operation (find the smallest element in the array $L[i, 1 \ldots M]$, $1 \le i \le M$) can be performed in parallel [6,17]. Finally, in Step 4 the algorithm OSP selects k best optimal survivor paths of the set P_{OSP}. This, obviously, is equivalent to the problem of selecting the k-th smallest out of $|P_{OSP}|$ elements. For this operation, we invoke a parallel algorithm proposed by Cole [6].

In order to evaluate the overall complexity of algorithm Par-OSP, let us compute the complexity of each step separately. As a model of parallel computation, we use both the concurrent-read, exclusive-write (CREW-PRAM) and the concurrent-read, concurrent-write (CRCW-PRAM) parallel RAMs [1,9,19].

Step 1: It is easy to verify that algorithm DP, which is used by OPS in Step 1, can be implemented on a CREW-PRAM in time $O(T \cdot \log K)$ with $K \cdot M$ processors, where $1 \le K \le M$. Moreover, if we use an CRCW-PRAM it is also easy to verify that it runs in time $O(T)$ with $O(K^2 \cdot M)$ processors, when a constant time algorithm for finding the minimum on a CRCW-PRAM is used.

Steps 2 and 3: The operations performed in Steps 2 and 3 can be executed in $O(\log M)$ time with M^2 processors on a CREW-PRAM model. The same operations required $O(1)$ time and $O(M^3)$ processors or $O(M)$ time and $O(M^2)$ processors, when an CRCW-PRAM computational model is used.

Step 4: As we have mentioned, Step 4 selects the 1st, 2nd, ..., k-th best out of q optimal survivor paths. As we have mentioned, we can use the algorithm of Cole [6] which selects the k-th smallest out of n elements on a EREW-PRAM in time $O(\log n \log^* n)$ by using $n/(\log n \log^* n)$ processors, where $\log^* n$ is the least i such that the i-th iterate of the logarithm function, i.e., $\log^{(i)} n$, is less that or equal to 2. Therefore, this step can be executed in time $O(\log q \log^* q)$ with $k \cdot q/(\log q \log^* q)$ processors on both, ERCW-PRAM and CRCW-PRAM models, where $1 \le q \le M$.

Thus, the overall time and processor complexities of algorithm Par-OSP are $O(T \cdot \log K + \log M)$ and M^2 respectively, $3 \le K \le M$, when it is executed on a CREW-PRAM. As far as the model CRCW-PRAM is concerned, algorithm Par-OSP can be executed either in time $O(T + M)$ with $\max\{K^2 \cdot M, M^2\}$ processors or in time $O(T)$ with M^3 processors.

8.2. *Parallel multiple line detection – Sub-optimal solution*

Despite algorithm Par-OSP behaves linearly with T (the depth of the K-trellis minus 1), it can take a considerable amount of computing time since T becomes very large in some applications. In these cases, as we have seen in Section 7, instead of a complete globally optimum solution, a quick sub-optimal solution is preferable.

Let us now focus on a parallel sub-optimal solution to the problem of finding line patterns in a display image in order to reduce the computational time needed for this operation. We consider parallel implementations of the sub-optimal algorithms Sub-OSP-k and SubOSP-h, described in Section 7. Hereafter, these implementations will be referred to as Par-SubOSP-k and Par-SubOSP-h, respectively.

Having a parallel implementation of algorithm

OSP, i.e., parallel algorithm Par-OSP, the parallel implementations of algorithms SubOSP-k and SubOSP-h are straightforward. We recall that, given a trellis graph $G(V_{1,T})$ with $T \cdot M$ vertices, algorithm SubOSP-k extract the k best optimal survivor paths from T/L graphs $G(V_{1,L})$, $G(V_{L,2L})$, ..., $G(V_{T-L,T})$, each with $(L+1) \cdot M$ vertices, while algorithm SubOSP-h extracts all optimal survivor paths whose cost metric does not exceed a threshold determined experimentally. Therefore, take into consideration the time and processor complexities of the algorithm Par-OSP, we conclude that algorithms Par-SubOSP-k and Par-SubOSP-h can be executed in time $O(L \cdot \log K + \log M)$ with $T/L \cdot M^2$ on a CREW-PRAM model. When a CRCW-PRAM computational model is available, these algorithms require $O(L+M)$ time and $\max\{T/L \cdot K^2 \cdot M,\ T/L \cdot M^2\}$ processors or $O(L)$ time and $T/L \cdot M^3$ processors.

Without a doubt, the previous sub-optimal parallel algorithms have the benefit to be very fast – they can operate in almost constant time since L can be considered small enough – but, they have a disadvantage, that is, their unknown detective ability; this is because the optimal algorithm OSP has unknown detective ability. Therefore, we are interested in testing algorithms OSP, SubOSP-k and SubOSP-h on simulated data in order to determine, first, how well algorithm OSP detects line patterns in a display image, next, any significant difference in the performance of the optimal algorithm OSP and sub-optimal algorithms SubOSP-k and SubOSP-h, and, finally, the conditions (e.g., range of values for L and K) under which sub-optimal algorithms detect at least as well as an optimal algorithm.

9. Testing the algorithms on simulated data

The performance of the optimal and sub-optimal algorithms, i.e., OSP, SubOSP-k and SubOSP-h, is evaluated by an application to simulated data. The advantage of simulated data over real data is that the tracks in the image are precisely known, thus we are able to assess the algorithms' performance.

The main points of this section are (i) to show the detective ability of the proposed algorithms by testing them under various conditions, and (ii) to find the possible conditions under which the optimal algorithm OSP and the sub-optimal SubOSP-k may detect identically.

9.1. Simulated data

Display images are generated by cross-correlating the output of two sensors that each receive a signal, $r_1(t)$ and $r_2(t)$ respectively, given by

$$r_1(t) = s(t) + n_1(t),$$

$$r_2(t) = s(t-d) + n_2(t),$$

where $s(t)$, $n_1(t)$ and $n_2(t)$ are uncorrelated Gaussian random processes which are observed for a total time of T seconds and d is the time delay [7,8]. The peaks of the correlograms form particular line patterns in the image.

Four images generated in this way are shown in Fig. 5(a) and Fig. 5(b) and Fig. 6(a) and Fig. 6(b). These images have been specially chosen because they combine all types of line patterns (straight, curve, unmerged, crossed, continuous and discontinuous lines). We have chosen to present display images with SNR $= -4$ and SNR $= -5$, since these images are adequate in showing the directive ability of the proposed algorithms.

9.2. Computation of vertex and edge metrics

Let us now compute the cost of each edge in a given K-trellis. As mentioned in Section 3, in this paper the cost of edges is given as a function of the metrics Q end D. Let us consider the computation of

Q and D. The vertex metric $Q_t(v_i)$ for the vertex $v_i \in V_t$, $1 \le i \le M$, $1 \le t \le T$, is computed by

$$Q_t(i) = \left(\max\{B_t\} - b_t(i)\right)^2, \tag{2}$$

where $B_t = \{b_t(v_1), b_t(v_2), \ldots, b_t(v_M)\}$.

The edge metric $D_t(v_i, v_j)$ for the edge $(v_i, v_j) \in E$, $v_i \in V_t$ and $v_j \in V_{t+1}$, $1 \le i \le M$ and $1 \le t \le T$, is computed as follows:

$$\begin{aligned} D_t(i, i-g) &= w_g b_{t+1}(i-g) + w_{g0} b_{t+2}(i-g) \\ &\quad + w_{g1} b_{t+2}(i-g-1), \\ D_t(i, i) &= w_0 b_{t+1}(i) + w_{00} b_{t+2}(i), \\ D_t(i, i+j) &= w_g b_{t+1}(i+g) + w_{g0} b_{t+2}(i+g) \\ &\quad + w_{g1} b_{t+2}(i+g+1), \end{aligned} \tag{3}$$

where

$$\begin{aligned} w_g &= w_t(i, i-g) = w_t(i, i+g), \\ w_0 &= w_t(i, i), \\ w_{g1} &= w_{t+1}(i-g, i-g-1) \\ &= w_{t+1}(i+g, i+g+1), \\ w_{g0} &= w_{t+1}(i-g, i-g) = w_{t+1}(i+g, i+g), \\ w_{00} &= w_{t+1}(i, i). \end{aligned}$$

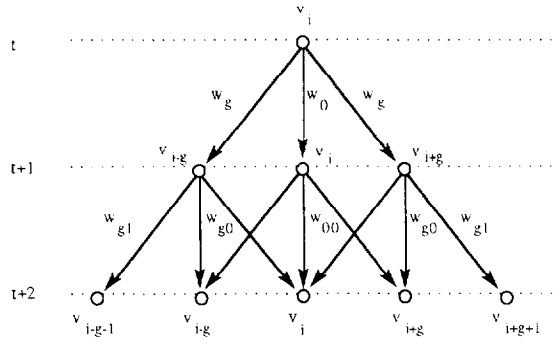


Fig. 4. Edge weights w_0, w_g, w_{00}, w_{g0}, w_{g1} used in the computation of the edge metrics $D_t(i, i-g)$, $D_t(i, i)$ and $D_t(i, i+g)$.

Moreover, the edge weights w_0, w_g, w_{00}, w_{g0}, w_{g1} satisfy the following conditions:

$$w_{g0} + w_{g1} = w_g,$$

$$w_{g0}/w_{g1} = w_0/w_g,$$

$$w_0 = w_{00},$$

where $j = i-g, \ldots, i, \ldots, i+g$ and $t = 1, 2, \ldots, T$ (see Fig. 4). Thus, the cost of each edge $(i, j) \in E$ in the K-trellis is given by the formula

$$c(i, j) = Q(i) - D(i, j), \quad \forall (i, j) \in E,$$

where $Q(i)$ and $D(i, j)$ are given by Eq. (2) and Eq. (3), respectively.

9.3. Test conditions

We are interested in testing the sub-optimal algorithms' performance as a function of signal-to-noise ratio (SNR) and cost function. For this purpose, display images were generated with SNR values of 0, -1, $\ldots$, -7 dB. Furthermore, one image has been generated with only noise. As mentioned earlier, the cost $c(i, j)$ of each edge $(i, j) \in E$ is given as a function of vertex and edge weights. For the needs of the test, five different sets of edge weights have been generated as follows.

Let v_i be a vertex of the set V_t, $t = 1, 2, \ldots, T-1$. By definition the vertex v_i is connected with vertices $\{v_{i-g}, \ldots, v_i, \ldots, v_{i+g}\}$ of the set V_{t+1}. We initially set

$$w_t(i, i) = 1$$

and, we compute the weights $w_{i,\, i-g}$ and $w_{i,\, i+g}$ using the formula

$$w_t(i, i-g) = w_t(i, i+g) = \frac{x}{2^{(g-1)}},$$

where x is a real number in the interval (0, 1) and $g = 1, 2, 3, \ldots, M-1$. The five sets of edge weights, called $W_1, W_2, \ldots, W_5$, are generated using

values of $x = 0.8, 0.6, 0.5, 0.4, 0.2$ respectively, i.e.,

$W_1 = \{1, 0.8, 0.40, \ldots\}$

$W_2 = \{1, 0.6, 0.30, \ldots\}$

$W_3 = \{1, 0.5, 0.25, \ldots\}$

$W_4 = \{1, 0.4, 0.20, \ldots\}$

$W_5 = \{1, 0.2, 0.10, \ldots\}$

According to these weights, five sets of edge costs, referred to as $S_1, S_2, \ldots, S_5$, are generated using Eq. (2) and Eq. (3).

9.4. Experiments

As mentioned above, eight images have been generated with SNR = 0, −1, ..., −7 dB and one image with noise only. Both sub-optimal algorithms were tested in each image with $K = 3$ and $L = 6$, where L is the length of the paths computed by each algorithm. In all cases, tests were done for each edge set $S_1, S_2, \ldots, S_5$.

Apart from testing the performance of the sub-optimal algorithms SubOSP-k and SubOSP-h, we are also interested in testing the difference in the performance between the optimal algorithm OSP and the sub-optimal SubOSP-k. For this purpose the generated display images have only one line presented as a continuous curved line. The path metric $F(P)$, defined in Section 3, has been chosen as criterion to evaluate the algorithms' performance. For each value of SNR, 100 display images were used, each generated with a different seed. The path metrics for each sample are given by

$$a_i^j = \left\{F\left(P_i^j\right) | \text{SNR} = i\right\},$$

$$a_n^j = \left\{F\left(P_n^j\right) | \text{noise only}\right\},$$

where P_i^j is the path taken by the tested algorithm in the j-th sample in the image and P_n^j is the path taken in the image with noise only, $i = 0, -1, -2, \ldots, -7$ and $j = 1, 2, \ldots, 100$. Since algorithm SubOSP-k always computes paths of length $L = 6$ (it computes T/L paths of length $L = 6$), we take $F(P_i^j)$ to be the sum of the path metric of each path.

The values of the path metrics are taken for each algorithm, i.e., OSP and SubOSP-k, and for each combination of SNR = 0, −1, −2, ..., −7 and S_i, $i = 1, 2, \ldots, 5$. The mean and standard deviation of each normal curve (100 sample values of each case) taken by the algorithm OSP are given in Table 1, and by the algorithm SubOSP-k are given in Table 2.

Table 1
Optimal algorithm OSP

SNR	S_1		S_2		S_3		S_4		S_5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
0	864.3	8.5	864.3	8.5	864.3	8.5	864.3	8.5	864.3	8.5
−3	216.2	4.9	216.2	4.9	216.2	4.9	216.2	4.9	216.2	4.9
−4	136.5	4.5	136.5	4.5	136.5	4.5	136.5	4.5	136.5	4.5
−5	89.0	4.0	88.3	4.0	88.0	4.0	87.6	4.0	87.1	4.2
−6	60.1	3.2	59.5	3.2	59.1	3.2	58.6	3.3	57.7	3.3
−7	50.9	2.5	50.4	2.5	49.9	2.5	49.4	2.5	48.3	2.6
Noise	44.2	1.4	43.9	1.5	43.6	1.5	43.1	1.6	42.1	1.7

Table 2
Sub-optimal algorithm SubOSP-k; $k = 1$, $L = 6$

SNR	S_1		S_2		S_3		S_4		S_5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
0	864.3	8.5	864.3	8.5	864.3	8.5	864.3	8.5	864.3	8.5
−3	216.2	4.9	216.2	4.9	216.2	4.9	216.2	4.9	216.2	4.9
−4	136.5	4.5	136.5	4.5	136.5	4.5	136.5	4.5	136.5	4.5
−5	88.5	4.2	87.9	4.2	87.7	4.2	87.4	4.2	86.9	4.2
−6	58.2	4.2	57.7	4.4	57.6	4.4	57.7	4.3	56.6	4.1
−7	47.8	3.2	47.8	3.2	47.3	3.1	47.0	3.2	46.3	3.2
Noise	40.8	1.7	40.5	1.8	40.2	1.8	40.0	1.7	36.1	1.8

9.5. Results

The graphical results show that there is no significant difference in the performance of the optimal algorithm OSP and the sub-optimal algorithm SubOSP-k, when SNR (in dB) belongs to the interval

$[-4, 0]$ and $L = 6$ (see Fig. 5(c)). In other words, both algorithms have the same detective ability with $K = 3$, edge set S_i, $i = 1, 2, \ldots, 5$ and SNR = 0, -1, -2, -3, -4, when local optimal paths of length 6 are used to construct sub-optimal paths. Moreover, in order to verify the graphical results an approximation of the standard normal distribution was used. As with the graphical results, these results showed no difference between the algorithms under the same conditions, i.e., with $K = 3$, $L = 6$, edge set S_i, $i = 1, 2, \ldots, 5$ and SNR = 0, -1, -2, -3, -4. The ability of the sub-optimal algorithms SubOSP-k and SubOSP-h to detect multiple lines is shown in Fig. 5(c) and Fig. 5(d) and Fig. 6(c) and Fig. 6(d), respectively.

Algorithm SubOSP-h computes paths of length 6 and accepts these paths whose metric does not exceed a threshold. The algorithm extracts the three middle vertices of each accepted path. In real applications the threshold can be determined experimentally in a variety of ways. In this paper, a preprocessing of data was performed in order to determine the minimum and maximum value of the path metric in the sub-images. The results showed in Fig. 6(c) and Fig. 6(d) have been taken with values of a threshold in the interval [min, max]. Further, we

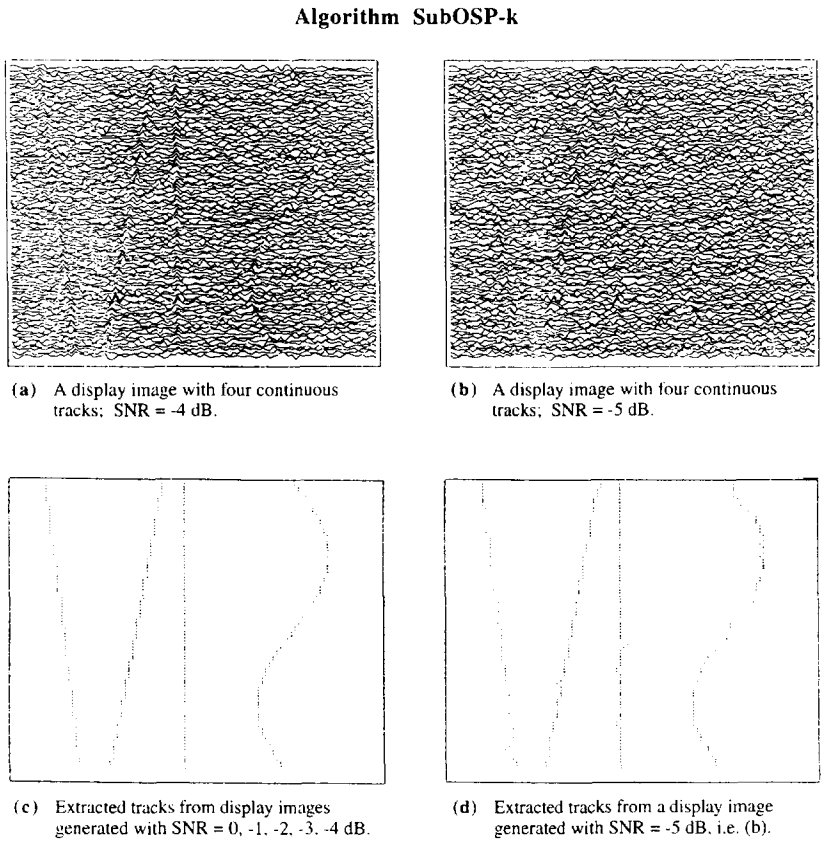
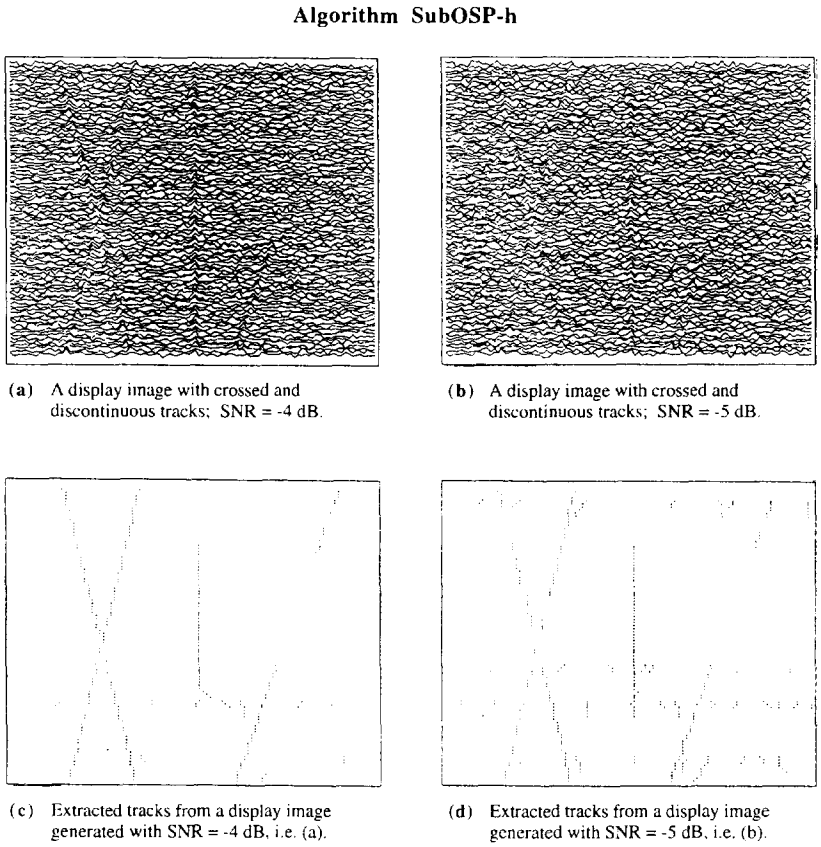


Fig. 5. (a) and (b) Display images containing four continuous tracks. (c) and (d) Graphical results (extracted tracks) taken by algorithm SubOSP-k, with $k = 4$. Note that the results presented in (c) are the same for all display images generated with SNR in the range $[-4, 0]$.



(a) A display image with crossed and discontinuous tracks; SNR = -4 dB.

(b) A display image with crossed and discontinuous tracks; SNR = -5 dB.

(c) Extracted tracks from a display image generated with SNR = -4 dB, i.e. (a).

(d) Extracted tracks from a display image generated with SNR = -5 dB, i.e. (b).

Fig. 6. (a) and (b) Display images containing four crossed and discontinuous tracks. (c) and (d) Graphical results (extracted tracks) taken by algorithm SubOSP-h, with $h = 18$ and $h = 9$ respectively.

should point out that some of the tracks extracted from these images by algorithm SubOSP-h actually represent false detection. These tracks can potentially be eliminated through further post-processing. Finally, we point out that the experimental results verify the theoretical results reported by Ianniello [7,8].

10. Conclusions

This paper has given a formulation of the track detection problem in terms of a directed weighted graph, and has proposed algorithms for optimal and sub-optimal solutions based on graph theoretic techniques. The experimental results have shown the ability of the proposed sub-optimal algorithms to detect multiple, crossing and discontinuous lines in a display image. Moreover, the results indicate that the sub-optimal algorithms can perform at least as well as a globally optimal algorithm when simulated data are used. However, work with real data is required before any definite statements can be made as to their applicability to target detection.

In closing, it should be mentioned that the sub-op-

timal approach to the track detection problem leads us to formulate sub-optimal parallel algorithms whose computational times are negligible compared with global optimal algorithms or even with sub-optimal ones. As shown, both sub-optimal algorithms proposed in this paper have such features that can easily be implemented in a parallel process environment.

Acknowledgements

The first author wishes to thank Dr. J. Ianniello of SACLANT Undersea Research Center for his assistance with various parts of this work and Dr. R. Strait of Naval Underwater System Center for enlightening conversations on this subject.

References

[1] P. Beame and J. Hastad, Optimal bounds for decision problems on the CRCW PRAM, *J. Assoc. Comput. Mach.* 36 (1989) 643–670.

[2] D.P. Atherton, E. Gul, A. Kountzeris and M.M. Kharbouch, Tracking multiple targets using parallel processing, *IEE Proceedings* 137 (1990) 225–234.

[3] R.E. Bellman and S.E. Dreyfus, *Applied Dynamic Programming* (Princeton University Press, Princeton, 1962).

[4] W.G. Bliss and L.L. Scharf, Algorithms and architectures for dynamic programming on Markov chains, *IEEE Trans. ASSP* 37 (1989) 900–912.

[5] D. Castanon, Efficient algorithms for finding the K best paths through a trellis, *IEEE Trans. AES* 26 (1990) 405–410.

[6] R. Cole and U. Vishkin, Deterministic coin tossing and accelerating cascades: Micro and macro techniques for designing parallel algorithms, *Processing of the 18th Annual ACM Symposium on Theory of Computing*, Berkeley, California (1986) 206–219.

[7] J.P. Ianniello, Time delay estimation via cross-correlation in the presence of large estimation error, *IEEE Trans. ASSP* 30 (1982) 998–1003.

[8] J.P. Ianniello, Threshold effects in time delay estimation, *ACSSC* 21 (1988) 88–92.

[9] L. Kucera, Parallel computation and conflicts in memory access, *Inf. Process. Lett.* 14 (1982) 93–96.

[10] T.A. Lanfear and A.G. Constantinidis, Graph partitioning with applications to signal processing, TR-1110, Imperial College, Signal Processing Section, 1986.

[11] H.F. Li, D. Pao and R. Jayakumar, Improvements and systolic implementation of the hough transformation for straight line detection, *Patttern Recognition* 22 (1989) 697–706.

[12] C.L. Morefield, Application of 0–1 integer programming to multitarget tracking problems, *IEEE Trans. AC* 22 (1977) 302–312.

[13] S.D. Nikolopoulos and G. Samaras, Sub-optimal approach to track detection for real-time systems, *21st Euromicro Conference on Design of Hardware and Software Systems, IEEE / CS*, Como, Italy (1995) 230–240.

[14] S.D. Nikolopoulos and A. Pitsillides, Towards network survivability by finding the K-best paths through a trellis graph, *ICT'96: International Conference on Telecommunications*, Istanbul, Turkey (1996) 817–821.

[15] C. Savage and J. Ja'Ja, Fast, efficient parallel algorithms for some graph problems, *SIAM J. Comput.* 10 (1981) 682–691.

[16] L.L. Scharf and H. Elliott, Aspects of dynamic programming in signal and image processing, *IEEE Trans. AC* 26 (1981) 1018–1029.

[17] Y. Shiloach and U. Vishkin, Finding the maximum, merging, and sorting in a parallel computation model, *J. Algorithms* 2 (1981) 88–102.

[18] R.L. Streit and R.F. Barret, Frequency line tracking using hidden Markov models, *IEEE Trans. ASSP* 38 (1990) 584–598.

[19] U. Vishkin, Implementation of simultaneous memory address access in model that forbid it, *J. Algorithms* 4 (1983) 45–50.

[20] A. Washburn, On not losing track, *IEEE Trans. AC* 35 (1990) 852–855.

[21] J.K. Wolf, A.M. Viterbi and G.S. Dixon, Finding the best set of K paths through a trellis with applications to multitarget tracking, *IEEE Trans. AES* 25 (1989) 287–296.

[22] L.J. Wu and T.E. Curtis, Practical graph partitioning algorithms for SONAR, Underwater Acoustic Data Processing (1989) 637–643 (1989).

Stavros D. Nikolopoulos received his B.Sc. degree (with Honours) in Mathematics from the University of Ioannina, Greece, in 1982, his M.Sc. degree in Computer Science from the University of Dundee, Scotland, in 1985, and his Ph.D. degree in Computer Science from the University of Ioannina in 1991. He has worked as researcher at the SACLANT Undersea Research Centre, Signal Processing Group, Italy (Research Assistant) and has taught at the Hellenic Airforce Academy, Greece (Visiting Professor). He is currently a Faculty member (Lecturer) in the Computer Science Department at the University of Cyprus, Cyprus, and an Assistant Professor in the Computer Science Department at the University of Ioannina, Greece. His current research interests focus on graph theory, graph and distributed algorithms, combinatorial mathematics, parallel and distributed computation, and distributed transaction processing.

George Samaras received his Ph.D. in Computer Science from Rensselaer Polytechnic Institute, USA. He is currently an Assistant Professor in the Computer Science Department of the University of Cyprus, Cyprus. He was previously at IBM Research Triangle Park, USA. He served as the lead architect of IBM's distributed commit architecture (LU6.2 Sync Point). His research interests include transaction processing, databases, mobile computing, object-oriented technology and real-time systems. He also served on several of IBM's internal international standard committees.