A New Method for Multidimensional Optimization and Its Application in Image and Video Processing

Dan Schonfeld, *Senior Member, IEEE*, and Nidhal Bouaynaya, *Student Member, IEEE*

***Abstract*—We derive a new method for multidimensional dynamic programming using the inclusion–exclusion principle. We subsequently propose an extension of the Viterbi algorithm to semi-causal, multidimensional functions. This approach is based on extension of the 1-D trellis structure of the Viterbi algorithm to a tree structure in higher dimensions. We apply the dynamic tree programming algorithm to active surface extraction in video sequences. Simulation results show the efficiency and robustness of the proposed approach.**

***Index Terms*—Active contours, dynamic programming (DP), Viterbi algorithm.**

I. Introduction

BELLMAN's dynamic programming (DP) algorithm is a fundamental technique for solving optimization problems [1]. The dynamic programming solution is obtained by using an iterative procedure that determines the optimal control vector for any admissible state at any stage. This optimization procedure follows from Bellman's principle of optimality: the "tail" of the optimal space is optimal for the corresponding "tail" subproblem [1]. The computational requirements of this technique become excessive when it is applied to high-dimensional problems. This is called the "curse of dimensionality" by Bellman. A much more efficient solution to DP is provided by the Viterbi algorithm, sometimes referred to as the forward algorithm [2]. The Viterbi algorithm can be used to optimize nearest-neighbor, causal 1-D systems by using a forward process. In this process, intermediary costs and optimal state mappings are computed at each node in the forward path. This approach, however, does not scale to multidimensional systems since causality breaks down in higher dimensions. In this letter, we will extend the Viterbi algorithm to provide a forward process to optimize semi-causal, multidimensional functions. We will then apply the proposed solution for computation of active surfaces in video sequences.

II. n-D Dynamic Programming

Based on the principle of optimality, we derive a new representation of the n-D DP algorithm using the inclusion–exclusion principle for multidimensional systems and prove, by induction, that it converges to the optimal solution.

Manuscript received November 20, 2005; revised February 17, 2006 The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Vince D. Calhoun.

The authors are with the Department of Electrical and Computer Engineering, University of Illinois at Chicago. Chicago, IL 60607-7053 USA (e-mail: dans@uic.edu; nbouay1@uic.edu).

Digital Object Identifier 10.1109/LSP.2006.873142

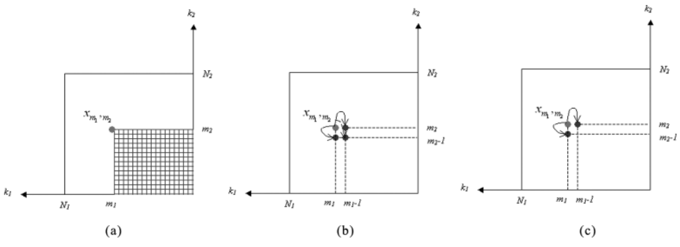

Fig. 1. 2-D systems. (a) Semi-causal. (b) Nearest-neighbor semi-causal. (c) Nearest-neighbor strictly semi-causal.

Definition 1: A discrete n-D system is semi-causal if

$$x_{k_1+1,\cdots,k_n+1} = f_{k_1+1,\cdots,k_n+1}\left(\{x_{i_1,\cdots,i_n}, u_{i_1,\cdots,i_n}\}\right)$$

where $i_1 \le k_1+1, i_2 \le k_2+1, \cdots, i_n \le k_n+1$, $k_i = 0,1,\cdots,N_i$, for all $i = 1,\cdots,n$. $x_{i_1,i_2,\cdots,i_n}$ is the n-D state at point $(i_1,i_2,\cdots,i_n)$, and $u_{i_1,i_2,\cdots,i_n}$ is the control or decision to be selected from a given set. Fig. 1(a) illustrates the semi-causality condition in a 2-D space. The control $u_{k_1,k_2,\cdots,k_n}$ is related to the state $x_{k_1,k_2,\cdots,k_n}$ by the policy $\mu_{k_1,k_2,\cdots,k_n}$, i.e., $u_{k_1,k_2,\cdots,k_n} = \mu_{k_1,k_2,\cdots,k_n}(x_{k_1,k_2,\cdots,k_n})$. Let π be the policy vector $\pi = (\mu_{0,0,\cdots,0}, \mu_{1,0,\cdots,0}, \cdots, \mu_{N_1,N_2,\cdots,N_n})$. The cost function starting at $x_{0,0,\cdots,0}$ is assumed to be additive, i.e.,

$$J(x_{0,\cdots,0}) = \sum_{k_1=0}^{N_1}\cdots\sum_{k_n=0}^{N_n} g_{k_1,\cdots,k_n}\left(x_{k_1,\cdots,k_n}, u_{k_1,\cdots,k_n}\right).$$

The optimization problem is performed over the policy vector π, i.e., $J^*(x_{0,0,\cdots,0}) = \min_\pi J_\pi(x_{0,0,\cdots,0})$. The optimal policy satisfies $J_{\pi^*}(x_{0,0,\cdots,0}) = J^*(x_{0,0,\cdots,0})$, and π^* is independent of $x_{0,0,\cdots,0}$.

Theorem 1 (n-D Dynamic Programming): Start with $J(x_{N_1,N_2\cdots,N_n}) = g_{N_1,N_2\cdots,N_n}(x_{N_1,N_2\cdots,N_n})$. Then, go backward using

$$J^*\left(x_{k_1,k_2,\cdots,k_n}\right) = \min_{u_{k_1,k_2,\cdots,k_n}}\left\{ g\left(x_{k_1,k_2,\cdots,k_n}, u_{k_1,k_2,\cdots,k_n}\right) + \sum_{i_1=k_1}^{N_1}\cdots\sum_{\substack{i_n=k_n \\ (i_1,\cdots,i_n)\neq(k_1,\cdots,k_n)}}^{N_n} g\left(x_{i_1,\cdots,i_n}\right)\right\}.$$

Moreover, using the inclusion–exclusion principle, we obtain

$$\begin{aligned}J^*(x_{k_1,k_2,\cdots,k_n}) = &\min_{u_{k_1,k_2,\cdots,k_n}} \{g(x_{k_1,k_2,\cdots,k_n}, u_{k_1,k_2,\cdots,k_n}) \\ &+ \sum_{i=1}^{n} J\left(x_{k_1,\cdots,k_{i-1},k_i+1,k_{i+1},\cdots,k_n}\right) \\ &- \sum_{\substack{(i_1,i_2)=(1,1)\\ i_1\neq i_2\\ (i_1,i_2)=(i_2,i_1)}}^{(n,n)} J\left(x_{k_1,\cdots,k_{i_1}+1,\cdots,k_{i_2}+1,\cdots,k_n}\right) + \cdots\cdots \\ &+ (-1)^{m-1} \sum_{\substack{(i_1,\cdots,i_m)=(1,\cdots,1)\\ i_1\neq\cdots\neq i_m\\ (i_1,\cdots,i_m)=P((i_1,\cdots,i_m))}}^{(n,\cdots,n)} J\left(x_{k_{i_1}+1,\cdots,k_{i_m}+1,\cdots,k_n}\right) \\ &+ \cdots\cdots + (-1)^{n-1} J\left(x_{k_1+1,k_2+1,\cdots,k_n+1}\right)\} \qquad (1)\end{aligned}$$

where $P((i_1, i_2, \cdots, i_m))$ denotes a permutation of $(i_1, i_2, \cdots, i_m)$. The energy is computed, for the different states, following this non-unique order: compute the optimal energy in (1) for the states $x_{N_1-i_1,k_2,\cdots,k_n}, x_{k_1,N_2-i_2,\cdots,k_n}, \cdots, x_{k_1,\cdots,N_n-i_n}$, for all values of $k_1, k_2, \cdots, k_n$. Then, increment $i_1, i_2, \cdots, i_n$, where $i_m = 0, \cdots, N_m - 1$ for $m = 1, \cdots, n$ until the optimal energy is computed for all the states. $J(x_{0,0,\cdots,0})$, generated at the last step, is equal to the optimal cost $J^*(x_{0,0,\cdots,0})$. Also, the policy $\pi^* = (\mu^*_{0,0,\cdots,0}, \cdots, \mu^*_{N_1,\cdots,N_n})$ is optimal.

Corollary 1 (2-D Dynamic Programming): Start with $J(x_{N_1,N_2}) = g_{N_1,N_2}(x_{N_1,N_2})$. Then, go backward using

$$\begin{aligned}J^*(x_{k_1,k_2}) = &\min_{\mu_{k_1,k_2}} \{g_{k_1,k_2}(x_{k_1,k_2}, \mu_{k_1,k_2}(x_{k_1,k_2})) \\ &+J(x_{k_1+1,k_2}) + J(x_{k_1,k_2+1}) - J(x_{k_1+1,k_2+1})\}\end{aligned}$$

computed in the following non-unique order: $(N_1 - i, k_2)$ and $(k_1, N_2 - j)$, where $k_2 = N_2 - i, \cdots, 1$ and $k_1 = N_1 - j, \cdots, 1$ for $i = 0, \cdots, N_1 - 1$ and $j = 0, \cdots, N_2 - 1$.

Proof: The proof of the n-D case follows by induction from the 2-D case. Let $\hat{\pi}_{k_1,k_2}$ denote a tail policy from point x_{k_1,k_2} onward, excluding the point x_{k_1,k_2}, i.e., $\hat{\pi}_{k_1,k_2} = \{\mu_{k_1+1,k_2}, \mu_{k_1,k_2+1}, \mu_{k_1+1,k_2+1}, \cdots, \mu_{N_1,N_2}\}$. Let $W(x_{k_1,k_2}) = J(x_{k_1,k_2}) - g(x_{k_1,i_2}, \mu_{k_1,k_2}(x_{k_1,k_2}))$. Initially, $J(x_{N_1,N_2}) = J^*(x_{N_1,N_2})$ is known. Let us assume that $J(x_{k_1,k_2})$ is optimal for the tail problem from point (k_1, k_2) onward, excluding the point (k_1, k_2). Therefore, $W(x_{k_1,k_2})$ is optimal. We have

$$\begin{aligned}J^*(x_{k_1,k_2}) &= \min_{\mu_{k_1,k_2}} \left\{g_{k_1,k_2}(x_{k_1,k_2}, \mu_{k_1,k_2}(x_{k_1,k_2})) + \min_{\hat{\pi}_{k_1,k_2}} W(x_{k_1,k_2})\right\} \\ &= \min_{\mu_{k_1,k_2}} \{g_{k_1,k_2}(x_{k_1,k_2}, \mu_{k_1,k_2}) + W(x_{k_1,k_2})\} \\ &= \min_{\mu_{k_1,k_2}} \{g_{k_1,k_2}(x_{k_1,k_2}, \mu_{k_1,k_2}(x_{k_1,k_2})) \\ &\qquad +J(x_{k_1+1,k_2}) + J(x_{k_1,k_2+1}) - J(x_{k_1+1,k_2+2})\} \\ &= J(x_{k_1,k_2}).\end{aligned}$$

This completes the proof by induction. ■

We will employ this method for computation of the cost function in the dynamic tree programming (DTP) algorithm proposed in the next section.

III. 2-D Dynamic Tree Programming

We now present an extension of the classical Viterbi algorithm to multidimensional systems by providing a forward process DP algorithm for multidimensional cost functions.

For clarity, we will focus our presentation on nearest-neighbor, strictly semi-causal 2-D functions. However, variations of the proposed algorithm can be used for other forms of semi-causality and higher dimensional spaces.

The cost function is given by

$$J(x_{0,0}) = \sum_{k_1=0}^{N_1} \sum_{k_2=0}^{N_2} g_{k_1,k_2}(u_{k_1,k_2}).$$

Definition 2: A discrete 2-D system is *nearest-neighbor semi-causal* if

$$x_{k_1+1,k_2+1} = f_{k_1+1,k_2+1}(u_{k_1,k_2+1}, u_{k_1+1,k_2}, u_{k_1+1,k_2+1}).$$

Definition 3: A discrete 2-D system is *nearest-neighbor strictly semi-causal* if

$$x_{k_1+1,k_2+1} = f_{k_1+1,k_2+1}(u_{k_1,k_2+1}, u_{k_1+1,k_2}).$$

Both concepts are illustrated in Fig. 1.

We now present the DTP algorithm for nearest-neighbor, strictly semi-causal 2-D functions.

a) Forward Process: We assume that $u_{k_1,k_2} = \mu_{k_1,k_2}(x_{k_1,k_2})$ is one of a finite number s of values, $k_1 = 1, \cdots, N_1$ and $k_2 = 1, \cdots, N_2$. Let $N_{\min} = \min(N_1, N_2)$ and $N_{\max} = \max(N_1, N_2)$.

Initialization: Set $J(x_{0,0})$ to a known initial value.

Recursion: For iteration $m = 1$, determine the optimal control $u^*_{1,1}$ such that $J(x_{1,1}) + g(u_{1,2}) + g(u_{2,1})$ is minimized for a given pair $(u_{1,2}, u_{2,1})$. Record the value of $u^*_{1,1}$ for the corresponding pair. Compute $J^*(u_{1,2}) = J(u^*_{1,1}) + g(u_{1,2})$ and $J^*(u_{2,1}) = J(u^*_{1,1}) + g(u_{2,1})$. Repeat the process for all pairs $(u_{1,2}, u_{2,1})$.

For iterations $m = 2, \cdots, N_{\min} + N_{\max} - 2$, do the following.

1) Fix a future $(g + \epsilon)$-tuple $\{u_{k_1,m-k_1+2} : k_1 = K_0, \cdots, K_1\}$, where

$$(\epsilon, K_0, K_1) = \begin{cases} (1, 1, m+1), & m \leq N_{\min} - 1 \\ (0, 1, N_{\min}), & N_{\min} \leq m < N_{\max} \\ (-1, N_{\max} + N_{\min} - m, N_{\min}), & N_{\max} \leq m. \end{cases}$$

The $(g + \epsilon)$-tuple lies along a diagonal layer of the 2-D structure [see Fig. 2(a)].

2) Determine the optimal g-tuple from the fixed future $(g + \epsilon)$-tuple as follows:

$$\begin{aligned}&\{u^*_{k,m-k+1} : k = K^g_0, \cdots, K^g_1\} \\ &= \underset{\substack{x_{k,m-k+1}\\ k=K^g_0,\cdots,K^g_1}}{\operatorname{argmin}} \left\{\sum_{k=K^g_0}^{K^g_1} \sum_{k_1=K_0}^{K_1} J(x_{k,m-k+1}) + g(u_{k_1,m-k_1+2})\right\}\end{aligned}$$

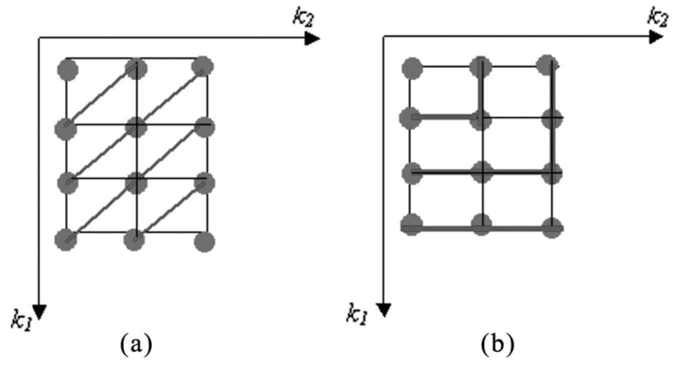

Fig. 2. Layers of the DTP algorithm. (a) Diagonal lines for nearest-neighbor, strictly semi-causal functions. (b) Quadrant walls for nearest-neighbor, semi-causal functions.

where

$$(K_0^g, K_1^g) = \begin{cases} (K_0, K_1 - 1), & m \leq N_{\min} - 1 \\ (K_0 - 1, K_1 - 1), & N_{\min} \leq m < N_{\max} \\ (K_0 + 1, K_1), & N_{\max} \leq m. \end{cases}$$

3) Record the optimal g-tuple, $\{u^*_{k,m-k+1} : k = K_0^g, \cdots, K_1^g\}$, for the fixed $(g+\epsilon)$-tuple.
4) Compute the cumulative cost at each control state in the $(g+\epsilon)$-tuple from the cumulative costs of the optimal g-tuple as follows:

$$J^*((x_{k_1,k_2})) = g_{k_1,k_2}(\mu(x_{k_1,k_2})) + J\left(u^*_{k_1-1,k_2}\right) + J\left(u^*_{k_1,k_2-1}\right) - J\left(u^*_{k_1-1,k_2-1}\right).$$

5) Repeat steps 1)–4) for all $s^{g+\epsilon}$ $(g+\epsilon)$-tuples.
6) Let $\mathrm{m} = \mathrm{m} + 1$, and go back to step 1).

b) Backtracking: The optimal control at the last node $u^*_{N_1,N_2}$ corresponds to the minimal cumulative cost $J^*((x_{N_1,N_2}))$. The optimal controls in the previous diagonal layer are uniquely determined by following the optimal mapping from the node $u^*_{N_1,N_2}$ (see Fig. 3). This process is repeated by following the optimal mappings from the optimal $(g+\epsilon)$-tuple to the g-tuple,$\{u^*_{k,m-k+1} : k = K_0, \cdots, K_1 - \epsilon\}$, for iterations $m = N_{\min} + N_{\max} - 1, \cdots, 2$. Observe that the optimal controls on each diagonal layer are determined simultaneously.

The main obstacle in the design of the DTP algorithm is the cost propagation in the forward process. The cost propagated to its neighboring nodes depends on the selection at multiple nodes. We therefore determine the optimal mapping among all permutations of nodes between adjacent dependent node classes in successive iterations, which form the layered structure. The DTP algorithm for a 2-D nearest-neighbor strictly semi-causal system with $N_1 = 4$, $N_2 = 3$, and $s = 2$ is illustrated in Fig. 3.

The 2-D DTP algorithm for nearest-neighbor, semi-causal functions follows the same forward process by considering states along the quadrant walls shown in Fig. 2(b). We can extend the DTP algorithm to higher dimensions (n-D with $n \geq 3$) by considering states along hyperplanes for nearest-neighbor, strict semi-causality and hyperquadrants for nearest-neighbor, semi-causality. It is possible to relax the nearest-neighbor assumption, which results in more complex and less efficient algorithms.

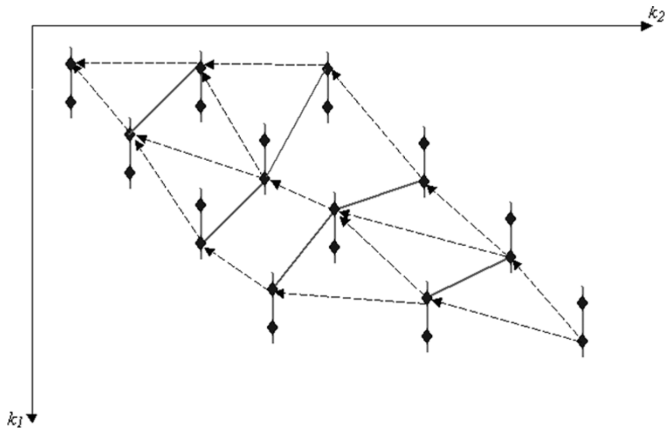

Fig. 3. Illustration of the 2-D DTP algorithm for a nearest-neighbor strictly semi-causal system, where $N_1 = 4, N_2 = 3$, and $s = 2$. The vertical segments are the states. The two distinct control values for each state appear as diamonds on each segment. The arrows point to the optimal g-tuple from a given future $(g+\epsilon)$-tuple. At the last state, the control value corresponding to the minimum cumulative energy is selected, and the minimum cost path is traced back.

TABLE I
COMPUTATIONAL COMPLEXITY

2-D DTP	$2s^{2N-1}$	1-D Viterbi	s^2N
Exhaustive search	s^{N^2}	Exhaustive search	s^N
Saving ratio	$\frac{1}{2}s^{N^2-2N+1}$	Saving ratio	$\frac{1}{N}s^{N-2}$

Complexity of the 2-D DTP: Consider a 2-D nearest-neighbor, strictly semi-causal system with states $x_{k_1,k_2}, k_1 = 1, \cdots, N_1, k_2 = 1, \cdots, N_2$ and controls u_{k_1,k_2} to be selected from a set of s distinct values. For simplicity, we assume that $N_1 = N_2 = N$. Consider now a 1-D system with N states and s controls. Table I shows the number of operations required to compute the 2-D DTP, exhaustive search in 2-D, the 1-D Viterbi, and an exhaustive search in 1-D. The saving ratio of an algorithm is defined as the ratio of the computational complexity of exhaustive search compared to the algorithm. For instance, if $N = s = 10$, the saving ratio of the 2-D DTP is equal to 5×10^{80}, while the saving ratio of the 1-D Viterbi is equal to 10^7.

IV. Active Surfaces

Active contours (snakes) are often used to extract an object's boundary. This method, however, produces a jittery and unstable boundary in video sequences. We propose a framework for active surface extraction from video sequences. An active surface is a generalization of the notion of an active contour to higher dimensions. It can be viewed as a collection of active contours in successive frames such that the active contours are constrained by spatial and temporal energy terms.

A. 2-D Active Surface Model

In order to reduce the computational complexity, we consider a 1-D active contour model in each frame [3]. At frame t, only observations along the normal lines of the contour are detected. Let $\phi = 1, \cdots, M$ be the index of normal lines and $\lambda_{\phi,t} = -s, \cdots, s$ be the index of pixels along the normal line ϕ at time t. Let $I(\lambda_{\phi,t})$ denote the image intensity value at the pixel $\lambda_{\phi,t}$. The 2-D contour is then represented by the set of its

M normal lines. We consider a sequence of T video frames, each containing M normal lines. Hence, we have reduced the 3-D model of the active surface to a 2-D model.

B. Spatio–Temporal Energy Function

The spatial external energy pushes the contour toward image features such as edges

$$E_{\rm ext}^{\rm s}(\lambda_{\phi,t}) = -\left|I(\lambda_{\phi,t}+1) - I(\lambda_{\phi,t})\right|^2.$$

The spatial internal energy imposes smoothness of the contour by penalizing rough contour points

$$E_{\rm int}^{\rm s}(\lambda_{\phi,t}) = \left|\lambda_{\phi,t} - \lambda_{\phi-1,t}\right|^2.$$

The temporal external energy attracts the contour to features in neighboring frames, which provides stability when the current frame has weak features due to noise or blurring

$$E_{\rm ext}^{\rm t}(\lambda_{\phi,t}) = -\left|I(\lambda_{\phi,t})I(\lambda_{\phi,t-1}) + I(\lambda_{\phi,t}+1)I(\lambda_{\phi,t-1}+1)\right|^2.$$

The temporal internal energy imposes smoothness constraints on the active surface

$$E_{\rm int}^{\rm t}(\lambda_{\phi,t}) = \left|\lambda_{\phi,t} - \lambda_{\phi,t-1}\right|^2.$$

We define the spatio–temporal energy function $E_{\rm spatiotemporal}(\lambda_{\phi,t})$, given by

$$\begin{aligned} E_{\rm spatiotemporal}(\lambda_{\phi,t}) = {} & \alpha_{\rm ext}^{s} E_{\rm ext}^{\rm s}(\lambda_{\phi,t}) + \alpha_{\rm int}^{s} E_{\rm int}^{\rm s}(\lambda_{\phi,t}) \\ & + \alpha_{\rm ext}^{t} E_{\rm ext}^{\rm t}(\lambda_{\phi,t}) + \alpha_{\rm int}^{t} E_{\rm int}^{\rm t}(\lambda_{\phi,t}) \end{aligned}$$

where $\alpha_{\rm ext}^{s}$, $\alpha_{\rm int}^{s}$, $\alpha_{\rm ext}^{t}$, and $\alpha_{\rm int}^{t}$ are weight parameters introduced to balance the different energy terms.

The objective function of the active surface is given by

$$E(\lambda_{\phi,t}) = \sum_{t=1}^{T}\sum_{\phi=1}^{M} E_{\rm spatiotemporal}(\lambda_{\phi,t}).$$

The optimal active surface minimizes the total energy $E(\lambda_{\phi,t})$.

C. Dynamic Tree Programming

The cumulative energy $E_C(\lambda_{\phi,t})$ is given by

$$E_C(\lambda_{\phi,t}) = \sum_{n=1}^{t}\sum_{\varphi=1}^{\phi} E_{\rm spatiotemporal}(\lambda_{\varphi,n}).$$

We provide simulation results for the active surface model using the proposed DTP algorithm. Fig. 4 shows that our approach to active surface extraction yields a much more stable and less jittery target boundary in video sequences.

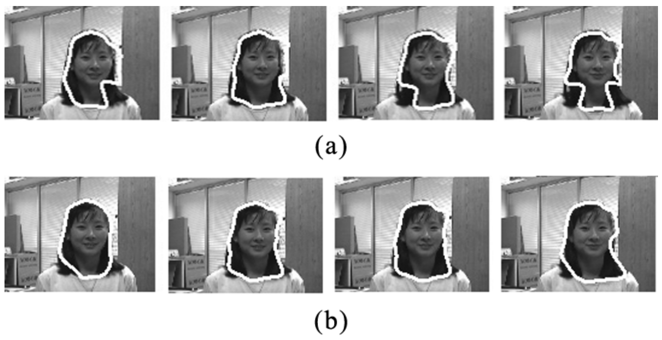

Fig. 4. Boundary extraction. (a) Active contours without temporal constraints. (b) Active surfaces (video is courtesy of S. Birchfield of Stanford University).

V. Conclusions

In this letter, we extended the well-known Viterbi algorithm to multidimensional systems. We first presented a new method for the representation of DP for multidimensional systems using the inclusion–exclusion principle. We then proposed a forward process to optimize semi-causal, multidimensional functions. This approach is based on extension of the 1-D trellis structure of the Viterbi algorithm to a tree structure in higher dimensions. We extended the notion of active contours to active surfaces in video sequences. We then applied the DTP algorithm to obtain the minimal energy active surface in video sequences. The proposed DTP algorithm can be used in many different applications, including multidimensional image segmentation, multicast network routing, multidimensional convolutional decoding, etc. The approach presented in this letter can be used to extend the 1-D variable-state Viterbi algorithm presented in [4] to arbitrary semi-casual, neighborhood dependence multidimensional Markov models.

Acknowledgment

The authors would like to thank Dr. D. Forney for his comments.

References

[1] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.
[2] A. J. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE Trans. Inf. Theory*, vol. IT-13, no. 4, pp. 260–269, Apr. 1967.
[3] Y. Chen, Y. Rui, and T. Huang, “Mode-based multi-hypothesis head tracking using parametric contours,” in *Proc. Automatic Face Gesture Recognition Conf.*, May 2002, pp. 112–117.
[4] J. Li, A. Najmi, and M. Gray, “Image classification by two-dimensional hidden Markov model,” *IEEE Trans. Signal Process.*, vol. 48, no. 2, pp. 517–533, Feb. 2000.