G-wire: A Livewire Segmentation Algorithm Based on a Generalized Graph Formulation

Hyung W. Kang

Department of Mathematics and Computer Science
University of Missouri - St. Louis
One University Blvd. St. Louis, MO 63121, USA
Phone: 314-516-5841, Fax: 314-516-5400
kang@cs.umsl.edu

Abstract

A novel livewire algorithm for image segmentation is proposed. Based on a generalized graph formulation, our algorithm incorporates the internal energy of the boundary curve while preserving the principle of optimality, resulting in better segmentation results in noisy images.

Keywords*: Semi-automatic image segmentation, Livewire, Snakes, Dynamic programming, Generalized graph formulation*

1 Introduction

Image segmentation techniques can be generally classified into manual, semi-automatic, and fully automatic ones. Manual segmentation, while good at object recognition, takes excessive amount of time and effort for precise boundary capture. Fully automated methods, while much more efficient, often fail to recognize the object of interest, resulting in inaccurate segmentation. On the other hand, semi-automatic techniques provide both high efficiency and accuracy by allowing users to do the object recognition and letting computers capture the fine details. Thus, semi-automatic image segmentation techniques are of practical use for various applications including medical image analysis, digital image composition, key extraction, etc.

While there are a variety of techniques of this category including region growing [1], snakes [17], livewire [21, 12], graph-cuts [5, 6], etc., our main focus in this paper is on the boundary-based techniques such as livewire and snakes. Snake, also called active contour, is an energy-minimizing curve which deforms to the target boundary from the initial curve specified by the user [17]. The energy is defined by a combination of internal force such as the curvature at the point and external forces such as its image gradients. One problem of snake is that since the deformation stage is purely automatic, the final shape of the boundary is hard to predict or control. If the resulting boundary is not acceptable, the whole process must be repeated with a new initial curve, or the computed boundary must be post-edited either manually or by adding new constraints interactively. livewire segmentation technique, based on minimum-cost path search from a single seed point on a graph, provides much tighter control to users since the desired path can be interactively 'selected' from multiple candidate paths [21, 22, 12, 13, 16]. Also, path digression is handled more effectively by allowing interactive placement of a new seed point from which to start the new boundary segment.

Livewire, however, has a limitation of its own. The cost function used in livewire graph search algorithm is only based on *external* features in the input image, such as gradient magnitudes. That is, there is no *internal curve energy* term used in existing livewire algorithms such as curvatures of the points on the boundary curve. Thus, unlike snakes, livewire does not produce a smooth boundary curve in which the internal energy is minimized, making livewire technique sensitive to noises. One might think of simply incorporating the internal energy such as curvature term into the cost function of the existing livewire framework, but as will be explained in the next section, it would violate the *principle of optimality* in the underlying shortest-path algorithm, and thus the optimality of the resulting path would no longer be preserved.

In this paper, we propose a new livewire algorithm called G-wire, which is capable of handling the internal energy as well as the external energy of the boundary curve. Based on a generalized multi-dimensional graph formulation, our algorithm allows both internal and external features to be incorporated into the cost function and handled in a unified manner. Given the new cost function and the generalized graph, G-wire successfully works without violating the principle of optimality, thus preserves the optimality of the computed path. As will be shown in the experimental results, our algorithm produces good segmentation results even in noisy images while retaining all the merits of the original livewire algorithm. Also, it provides fast enough feedback since we localize the search domain to expedite the segmentation process. The remainder of this paper is organized as follows. In Section 2, based on the detailed comparison of livewire and snake, we derive our new cost function for the graph search. In Section 3, we present the new graph formulation and the algorithm, as well as the search domain localization strategy. Sections 4 shows some experimental results, and Section 5 concludes this paper.

2 Livewire vs. Snake

After the first debut of snake [17], there have been a number of its variants developed so far, including finite element snakes, B-spline snakes, Fourier snakes, level-set snakes, etc [9, 26, 10, 19, 8, 4, 18, 7, 23]. In general, a snake models a contour as a time-varying curve $\mathbf{v}(s) = (x(s), y(s))$ where s represents the arc length. Once a snake curve is initialized by the user, the curve is deformed to find the desired boundary in an attempt to minimize the following energy functional.

$$E_{snake} = \int_0^1 E_{int}(\mathbf{v}(s)) + E_{ext}(\mathbf{v}(s))\, ds \tag{1}$$

where E_{int} represents the internal curve energy due to bending or discontinuities, and E_{ext} is the external energy mainly based on image forces. Although the image forces can be drawn by various events such as lines, edges, terminations, etc., we will focus on edge features like gradient magnitudes, which usually constitute the most dominant external force. The internal spline energy is written as

$$E_{int} = (\alpha|\mathbf{v}_s(s)|^2 + \beta|\mathbf{v}_{ss}(s)|^2) \tag{2}$$

where the subscripts indicate derivatives with respect to s. The first-order term increases where there is a gap in the curve, thus controlling the continuity of the curve. The second-order term gets large where the curve bends abruptly, which is essentially equivalent to the curvature of the curve. Accordingly, the values of α and β control the extent to which the contour is allowed to stretch or bend at a point. Especially, large β value lowers the curvature values, resulting in a smooth snake curve even in a noisy input image.

While the snake always produces a single solution which may not match the target boundary, livewire allows a user to select the most suitable curve segment from multiple choices, providing better controllability and predictability. Finding the target boundary is much easier with livewire since it works as if the curve is automatically snapped to the target boundary as the user moves the cursor around it. The livewire algorithm is essentially a graph-based single-source shortest path algorithm [11]. Whenever a *seed pixel* is interactively selected on the image, livewire algorithm computes the path map that records the minimum-cost paths from the seed pixel to all other pixels, considering the image as a directed graph with *nodes* (pixels) and *arcs* (links connecting neighboring pixels).

To assign an appropriate cost for each arc in the graph, a cost function should be defined first. Although livewire also has a number of its variants which share the basic underlying algorithm [21, 22, 12, 13, 16], most of their cost functions are solely based on external forces, that is, the edge features in the input image. Thus, the cost for an arc between two neighboring pixels $\mathbf{v}_k$ and $\mathbf{v}_{k+1}$ can be written as:

$$c(\mathbf{v}_k, \mathbf{v}_{k+1}) = c_{ext}(\mathbf{v}_k, \mathbf{v}_{k+1}) = (|\nabla I(\mathbf{v}_k)| + |\nabla I(\mathbf{v}_{k+1})|)/2 \tag{3}$$

where $|\nabla I(\mathbf{v})|$ denotes the gradient magnitude of image I at pixel $\mathbf{v}$. However, the lack of internal energy in existing livewire algorithms could lead to rough and inaccurate boundary segmentation especially when the input image is noisy.

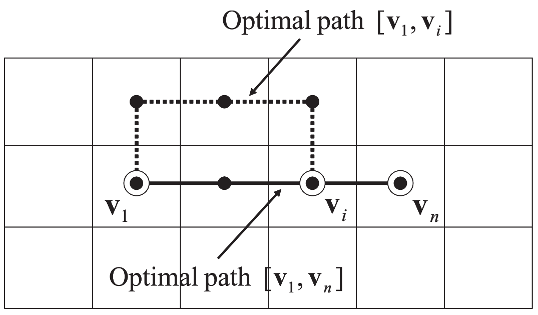
In order to incorporate internal energy (such as in Eq. 2) into the cost function, we take advantage of an interesting fact that both livewire and snake algorithms can be formulated as dynamic programming. The livewire algorithm, as a dynamic programming, attempts to minimize the function $f(\mathbf{v}_n)$ which represents the cost of a path from a seed pixel $\mathbf{v}_1$ to any pixel $\mathbf{v}_n$ though a sequence of neighboring pixels $\mathbf{v}_2, \mathbf{v}_3, ..., \mathbf{v}_{n-1}$. The recurrence relation for the dynamic programming is as follows:

$$f(\mathbf{v}_{k+1}) = \min_{\mathbf{v}_k}\{f(\mathbf{v}_k) + c(\mathbf{v}_k, \mathbf{v}_{k+1})\} \tag{4}$$

Here, *the principle of optimality* states that if an optimal path from $\mathbf{v}_1$ to $\mathbf{v}_n$ passes through a pixel $\mathbf{v}_i$, then its subpath from $\mathbf{v}_1$ to $\mathbf{v}_i$ should be itself an optimal path from $\mathbf{v}_1$ to $\mathbf{v}_i$ [3]. That is, the decision-maker should always choose the optimal solution at each stage. Obviously, Eq. 4 does not violate this principle since all the future decisions at a current state k are simply based on an external function c_{ext} which is independent of how we got to the current state.

Now, let us incorporate the following discrete versions of the two internal energy terms:

$$|\mathbf{v}_s(s)|^2 \approx |\mathbf{v}_{i+1} - \mathbf{v}_i|^2 \ \ and\ \ |\mathbf{v}_{ss}(s)|^2 \approx |\mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}|^2 \tag{5}$$



Optimal path $[\mathbf{v}_1, \mathbf{v}_i]$
$\mathbf{v}_1$
$\mathbf{v}_i$
$\mathbf{v}_n$
Optimal path $[\mathbf{v}_1, \mathbf{v}_n]$

Figure 1. The violation of the optimality principle

to obtain the new cost function

$$
\begin{aligned}
c(\mathbf{v}_{k-1}, \mathbf{v}_k, \mathbf{v}_{k+1}) \;=\;& \alpha|\mathbf{v}_{k+1} - \mathbf{v}_k|^2 \\
& +\beta|\mathbf{v}_{k+1} - 2\mathbf{v}_k + \mathbf{v}_{k-1}|^2 \\
& +\gamma c_{ext}(\mathbf{v}_k, \mathbf{v}_{k+1}) \qquad (6)
\end{aligned}
$$

where α, β, and γ are weight parameters. Now if we apply this new cost function to Eq. 4, the principle of optimality no longer holds since the future decisions at a state k depends not only on the external functions, but also on the decisions made at the previous state $k-1$. This implies that the decision at a certain stage may not be optimal and has to be corrected later. That is, an optimal path from $\mathbf{v}_1$ to $\mathbf{v}_i$ may not be inlcuded in the eventual optimal path from $\mathbf{v}_1$ to $\mathbf{v}_n$ that passes through $\mathbf{v}_i$ (Fig. 1). For this reason, simply putting the internal energy term into the existing livewire framework does not guarantee the optimality of the computed path.

To resolve this problem, we need to re-define a state as a combination of two neighboring pixels $\mathbf{v}_{k+1}$ and $\mathbf{v}_k$ since the internal energy spans two consecutive arcs. Then the new recurrence relation can be written as:

$$
f(\mathbf{v}_{k+1}, \mathbf{v}_k) = \min_{\mathbf{v}_{k-1}} \{f(\mathbf{v}_k, \mathbf{v}_{k-1}) + c(\mathbf{v}_{k-1}, \mathbf{v}_k, \mathbf{v}_{k+1})\} \qquad (7)
$$

That is, our objective at each stage is now to find a pixel $\mathbf{v}_{k-1}$ that minimizes the cost of a path that ends with an arc $(\mathbf{v}_k, \mathbf{v}_{k+1})$. This formulation is similar to that of the dynamic programming version of snakes [2]. Note that now the principle of optimality is preserved in that an optimal path ending with an arc ($\mathbf{v}_{i-1}$, $\mathbf{v}_i$) is always included in the eventual optimal path that passes through this arc ($\mathbf{v}_{i-1}$, $\mathbf{v}_i$). The generalization of this idea leads to the following recurrence relation with an energy term that spans m consecutive arcs (such as multiscale curvature):

$$
\begin{aligned}
f(\mathbf{v}_{k+1}, ..., \mathbf{v}_{k-m+2}) \;=\;& \min_{\mathbf{v}_{k-m+1}} \{f(\mathbf{v}_k, ..., \mathbf{v}_{k-m+1}) \\
& +c(\mathbf{v}_{k-m+1}, ..., \mathbf{v}_{k+1})\} \qquad (8)
\end{aligned}
$$

Based on this framework, we propose in the next section a new livewire algorithm together with a generalized graph formulation.

3 The G-wire algorithm

3.1 The graph formulation

As described above, our algorithm is based on a high-order dynamic programming, which is often used for solving various problems in computer vision and pattern recognition such as contour matching, feature extraction, motion tacking, stereo matching, speech recognition, etc [2, 14, 20, 15, 24, 25]. To our knowledge, however, existing livewire algorithms are all based on a first-order dynamic programming and thus they model the image as a 2-dimensional graph where a single pixel $\mathbf{v} = (x, y)$ serves as a node. Since the new recurrence relation (Eq. 7) depends on two consecutive pixels at the same time, it requires a 4-dimensional graph where two pixels constitute a single node. Similarly, a $2m$-dimensional graph would be needed for the general case with a cost function that spans m arcs (Eq. 8). However, considering the fact that a single pixel in an image only has a constant number of neighbors (at most 8), we can lower the dimensionality of the graph down to $2 + (m - 1)$. Note that livewire, unlike snake, only allows a graph arc to be formed between two immediately neighboring pixels in the image. Thus, a 3-dimensional graph would be sufficient for implementing Eq. 7. Although the proposed idea itself is general to cover an arbitrary n-dimensional case, we will mainly focus on 3-dimensional case to clearly show the practicality of our method.

Given an input image with N pixels, the corresponding graph will be constructed with $N \times 8$ nodes, based on the 8-neighbor system. Each node is represented by 3 numbers (x, y, z), where x and y denote the location in the image and z denotes the source neighboring pixel from which it is connected. That is, (x, y) denotes position and z denotes direction. As shown in Fig. 2, a single pixel $\mathbf{p}$ can have 8 distinct nodes in the graph (denoted as p^0,...,p^7) indicating that these nodes will be treated independently even though they all lead to the same pixel $\mathbf{p}$. A graph arc is created between any two nodes p and q which satisfy all of the followings:

1. Their (x, y) coordinates are in the neighboring positions in the image.
2. The z value of one node points to the (x, y) position of the other.
3. Their z values do not point in exactly opposite directions.

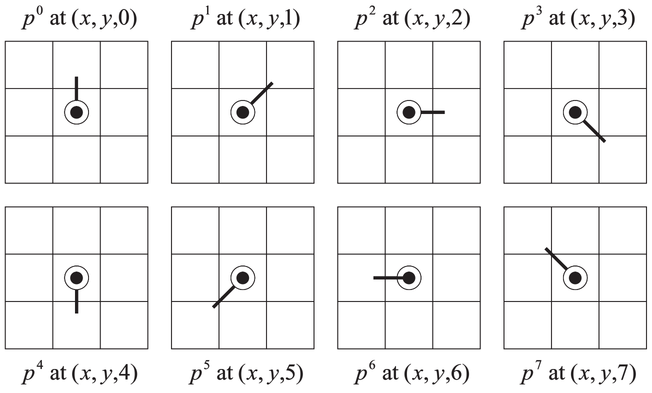


Figure 2. The 8 possible graph nodes for a single pixel p at (x, y)

Algorithm 1 ConstructPathMap(v_s, D)

```
INPUT : A starting node $v_s$ and the domain $D$
OUTPUT : An optimal path from $v_s$ to each node $v$ in $D$
DATA STRUCTURES : $A$(active node list), $E$(expanded node list)
 1: Insert $v_s$ in $A$
 2: while $E$ is not empty do
 3:   Remove the minimum-cost node $v$ from $A$, and insert it in $E$
 4:   for each $v'$ at $(x, y, z)$ ($\notin E$) where $(x, y)$ is in the neighborhood of $v$ and $z$ points to $v$ do
 5:     Let $\mathbf{v}_0$ be the pixel pointed to by $z$ coordinate of $v$
 6:     Let $\mathbf{v}_1$ be the pixel containing $v$ and let $\mathbf{v}_2$ be the pixel containing $v'$
 7:     $cc_{tmp}(v') := cc(v) + c(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2)$
 8:     if $cc_{tmp}(v') < cc(v')$ then
 9:       $cc(v') := cc_{tmp}(v')$ and now $v'$ is connected to $v$
10:       if $v' \notin A$ then insert $v'$ in $A$
11:     end if
12:   end for
13: end while
```

For example, an arc is generated between a destination node p at $(x, y, 2)$ and a source node q at $(x+1, y, z)$ for any values of z except for 6, which leads to $(8 - 1) = 7$ such cases. Since each pixel can be either a source or a destination, there are $7 + 7 = 14$ possible arcs between two neighboring pixels.

3.2 The algorithm

G-wire algorithm presented here is a modified shortest-path algorithm that runs on a 3D graph, and it can be naturally extended to an arbitrarily higher-dimensional case. Initially, all the nodes in the graph are assigned the maximum cost possible. Once a seed pixel is specified by user, a minimum-cost path from the seed pixel to each pixel in the image is computed. Starting from the seed pixel, the path map is expanded to the neighboring pixels, which is essentially the iteration of a single node expansion cycle. At each cycle, we pick the node v whose cumulative cost (denoted $cc(v)$) is minimum in the *active node list* containing the nodes visited at least once, and compute the cost from v to each node v' in its neighborhood, using the cost function defined in Eq. 6. Note that v' is considered only if it points to v by its z-coordinate. Whenever the new cost for v' is lower than its previous cost, the minimum-cost path for v' is updated to include this new arc (v, v'). If this process is done for all 8 neighbors of v, then v is inserted into the *expanded node list* and we move on to the next minimum-cost node. In this way, we can construct the minimum-cost path from the seed pixel to every pixel in the image. Note that, however, in our case there are 8 distinct paths computed for a single pixel. This path map construction process is summarized in Algorithm 1.

Given the constructed path map, as the user interactively moves the cursor point (also called *free pixel*, obtained at an event time t_k) around the target boundary, the minimum-cost path reaching the free pixel is instantly displayed. This gives an impression that the livewire curve automatically snaps to the target boundary, making it easy to for the user to select the most desirable path of all. While each pixel stores 8 distinct paths that lead to it, only the minimum-cost path out of them is displayed when the mouse cursor reaches it. Note, however, this path segment may not be displayed later as part of another optimal path even if it also contains the same pixel, which is a big improvement from previous algorithms. Also, note that there are 8 distinct nodes starting from the seed pixel and thus any one of these nodes might be selected as part of an optimal path. Whenever the curve digresses from the target boundary during this operation, the new seed pixel is interactively specified and the new path map construction resumes from this pixel, while freezing the previous boundary segment as approved. The complete boundary curve will be obtained as a sequence of such frozen segments. The proposed G-wire algorithm is summarized in Algorithm 2.

Another important issue in our algorithm is the continuity at the seed points. In the previous livewire algorithms, two successive boundary segments sharing a seed pixel were treated completely independently, which could result in non-smooth connection at the seed points. In our framework, however, the inclusion of internal energy that spans multiple arcs naturally leads us to the inspection of the cost at the seed points connecting two path segments. That is, successive segments can be connected smoothly in an attempt to minimize the cost function at each seed point. For this purpose, we need to look at how the previous segment ended, which is recorded in each of 8 nodes at the new seed pixel (which coincides with the previous ending point). Whenever the new segment starts, each of these 8 nodes at the seed pixel should be initialized with proper costs obtained from the previous boundary construction, and the path construction resumes with taking all of these 8 nodes into consideration. Now the next optimal path segment will be obtained such that it minimizes the internal energy for connecting to the previous segment.

Algorithm 2 G-wire

```
INPUT : A sequence of seed pixels $\mathbf{s}_0$, ..., $\mathbf{s}_{n-1}$ and the input image $I$
OUTPUT : A set of boundary segments forming an optimal path from $\mathbf{s}_0$ to $\mathbf{s}_{n-1}$ passing through $\mathbf{s}_1$, ..., $\mathbf{s}_{n-2}$
 1: $k := 0$, $\mathbf{v}(t_0) := \mathbf{s}_0$, $cc(v^p(t_0)) := 0$ for all $p \in \{0, ..., 7\}$
 2: while $\mathbf{v}(t_k) \neq \mathbf{s}_{n-1}$ do
 3:     if $k = 0$ then
 4:         $v^p(t_0)$:=the minimum-cost node at pixel $\mathbf{v}(t_0)$
 5:         ConstructPathMap($v^p(t_0)$, $I$)
 6:     else
 7:         $v^q(t_k)$:=the minimum-cost node at pixel $\mathbf{v}(t_k)$
 8:         Display a boundary segment from $v^p(t_0)$ to $v^q(t_k)$
 9:     end if
10:     if $\mathbf{v}(t_k)$ is a new seed then
11:         Freeze the previous boundary segment from $v^p(t_0)$ to $v^q(t_k)$
12:         $\mathbf{v}(t_0) := \mathbf{v}(t_k)$ and $k := 0$
13:     else
14:         $k := k + 1$ and get $\mathbf{v}(t_k)$
15:     end if
16: end while
```

Based on our algorithm, the two successive path segments $[\mathbf{s}_{l-1}, \mathbf{s}_l]$ and $[\mathbf{s}_l, \mathbf{s}_{l+1}]$ sharing a single seed pixel $\mathbf{s}_l$ is identical to the minimum-cost path $[\mathbf{s}_{l-1}, \mathbf{s}_{l+1}]$ that passes through $\mathbf{s}_l$. Also, it can be concluded that given a sequence of seed pixels $\mathbf{s}_0$, $\mathbf{s}_1$, ..., $\mathbf{s}_{n-1}$, our algorithm finds the optimal path between $\mathbf{s}_0$ and $\mathbf{s}_{n-1}$ that passes through all the intermediate the seed points, considering them as *hard constraints* which are often dealt with in snake algorithm.

3.3 Path map localization and incremental update

At each seed point selection, the path map is computed for all the pixels in the image. The time complexity for the path map construction is $O(N \times M)$ where N is the number of pixels in the image and M is the number of nodes for each pixel. When the size of the image is large, or the dimensionality of the graph is high, it could significantly slow down the feedback at the seed point selection. To resolve this problem, we employ the path map localization strategy from our previous work [16].

First, we locate a window $w(t_0)$ centered at the seed pixel, and the path map is constructed only inside the window. Then the window moves along with the free pixel, and the path map is incrementally expanded into the newly added region in the current window $w(t_k)$ at any time k (Fig. 3(a)). The path map expansion at time k begins with the minimum-cost node on the border of the previous window intersecting the current window (denoted $v_s(t_k)$). For

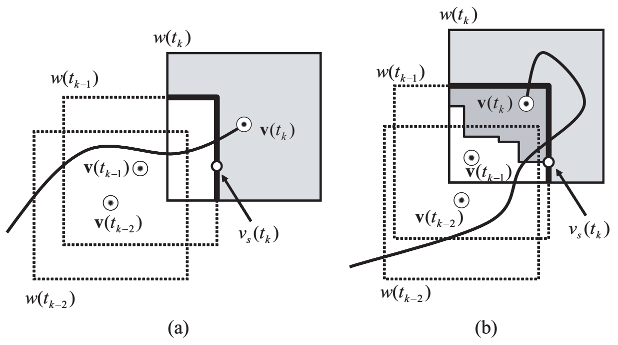


Figure 3. Incremental path map update

example, if there are m pixels on this intersecting border, then there are total of $m \times 8$ nodes to consider and the one with the minimum cost is selected as the new starting point. Basically, the path map is expanded into the nodes of the unexplored region in the current window. As shown in Fig. 3(b), however, some nodes in the already explored region in the current window might also have to be updated. Among the nodes in this already explored region, only the ones with the bigger cost than that of the minimum-cost node $v_s(t_k)$ have chances of getting updated. Thus, in the update domain at time k (denoted $D(t_k)$), we include all the nodes in the unexplored region and the ones with bigger cost than that of the minimum-cost node $v_s(t_k)$.

Algorithm 3 Incremental G-wire

```
INPUT : A sequence of seed pixels $\mathbf{s}_0$, ..., $\mathbf{s}_{n-1}$
OUTPUT : A set of boundary segments forming an optimal path from $\mathbf{s}_0$ to $\mathbf{s}_{n-1}$ passing through $\mathbf{s}_1$, ..., $\mathbf{s}_{n-2}$
 1: $k := 0$, $\mathbf{v}(t_0) := \mathbf{s}_0$, $cc(v^p(t_0)) := 0$ for all $p \in \{0, ..., 7\}$
 2: while $\mathbf{v}(t_k) \neq \mathbf{s}_{n-1}$ do
 3:     if $k = 0$ then
 4:         $v^p(t_0)$:=the minimum-cost node at pixel $\mathbf{v}(t_0)$
 5:         $v_s(t_0)$:=$v^p(t_0)$, $D(t_0) := w(t_0)$
 6:     else
 7:         $v_s(t_k)$ := the minimum-cost node in the border of $w(t_{k-1})$ in $w(t_k)$
 8:         $D(t_k)$ := the non-overlapping region in $w(t_k)$ $\cup$ the region containing the nodes of the previous windows in $w(t_k)$ whose costs are higher than that of $v_s(t_k)$
 9:     end if
10:     ConstructPathMap($v_s(t_k)$, $D(t_k)$)
11:     $v^q(t_k)$:=the minimum-cost node at pixel $\mathbf{v}(t_k)$
12:     Display a boundary segment from $v^p(t_0)$ to $v^q(t_k)$
13:     if $\mathbf{v}(t_k)$ is a new seed then
14:         Freeze the previous boundary segment from $v^p(t_0)$ to $v^q(t_k)$
15:         $\mathbf{v}(t_0) := \mathbf{v}(t_k)$ and $k := 0$
16:     else
17:         $k := k + 1$ and get $\mathbf{v}(t_k)$
18:     end if
19: end while
```

Algorithm 3 shows our modified algorithm based on the path map localization and incremental update paradigm. As proven by [16], this incremental algorithm still preserves the optimality of the computed path with respect to the given cost function, and in the cumulative region formed by the window sequence. Since the path map construction is always restricted to the current window, our incremental algorithm gives interactive feed back speed strictly bounded by the size of the window. That is, the time complexity for the feedback is $O(L \times M)$ if L is the number of pixels in a window and M is the number of nodes for each pixel.

In addition to the better time efficiency, this algorithm also leads to better accuracy in that it keeps the curve from digressing to nearby edges outside the window sequence, thus reducing the number of seed points required. The previous livewire techniques always look for a globally optimal path between two points, which often does not coincide with the target path. Our technique, however, provides a better chance to extract the target path without digressions by finding the locally optimal path in the domain formed by the user-guided window sequence (see Fig. 4). Also, the window is interactively resizable, providing more flexibility in segmenting boundaries with varying complexities.

4 Experimental results

Our G-wire algorithm has been successfully tested on hundreds of images with various sizes and complexities. As mentioned earlier, the biggest advantage of our algorithm is that it gives smooth and accurate segmentation results even in noisy images. Figure 5 shows the test results on the same input image, obtained by the standard livewire algorithm and G-wire. We spent the same number of seed points with both methods, to show G-wire produces smoother and more accurate segmentations on such noisy images.

In Figure 6, the effect of internal energy is clearly shown. By varying the values of the coefficients α and β for the internal energy terms, we can get different segmentation results from the same image. With larger internal energy coefficients, it gets easier to prevent the curve from digressing to nearby objects. Also, Fig. 7 shows that with our algorithm, the smoothness of the boundary curve is preserved at the seed points connecting two successive boundary segments. Note that since G-wire is a generalization of the livewire framework, it is capable of doing everything that can be achieved by the standard livewire algorithm. For example, when the target object has a clean but highly complex boundary, we can simply turn off α and β coefficients to get accurate segmentation results (Fig 8).

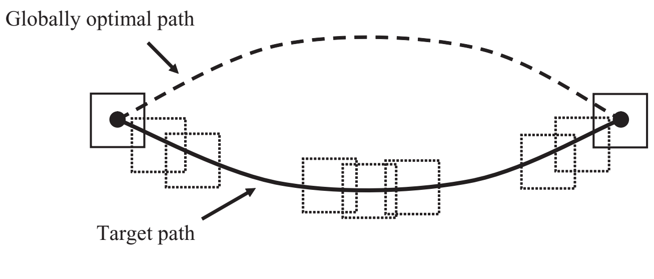


Figure 4. Finding a locally optimal path

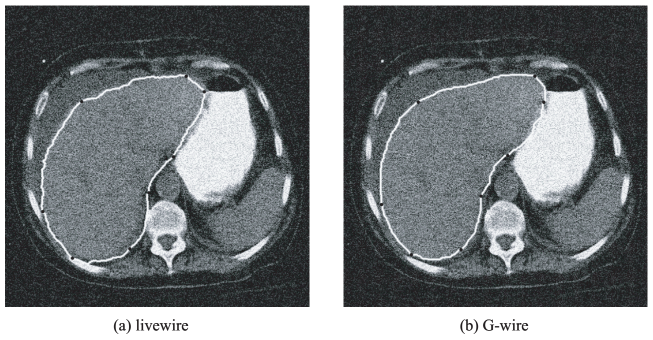


Figure 5. Livewire vs. G-wire

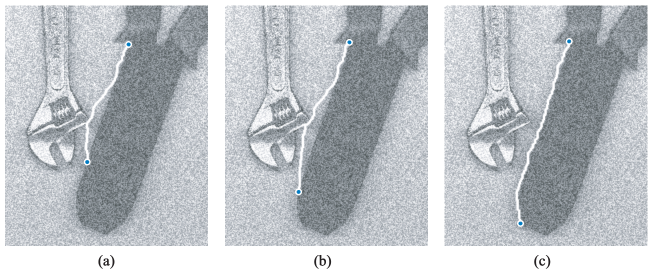


Figure 6. Effect of internal energy: (a) $\alpha = 0, \beta = 0, \gamma = 1$ **(b)** $\alpha = 0.1, \beta = 0.1, \gamma = 0.8$ **(c)** $\alpha = 0.1, \beta = 0.2, \gamma = 0.7$

For the path map localization, we used a fixed size of the window (80×80), and as a result our incremental G-wire algorithm performed at an interactive speed regardless of the image size, complexity of the target boundary, or the level of noise. All the experiments have been conducted on Intel $\mathrm{Pentium}^{®}$ PC (P4 2.90 GHz processor with 1 GB memory).

5 Conclusions

We have presented a generalized livewire segmentation algorithm based on a multidimensional graph formulation. Unlike existing livewire algorithms, our algorithm is capable of incorporating both the internal energy and the external energy of the boundary curve while preserving the optimality of the computed path on the graph, and thus leads to better segmentation results especially in noisy images. While we mainly focused on a 3-dimensional graph

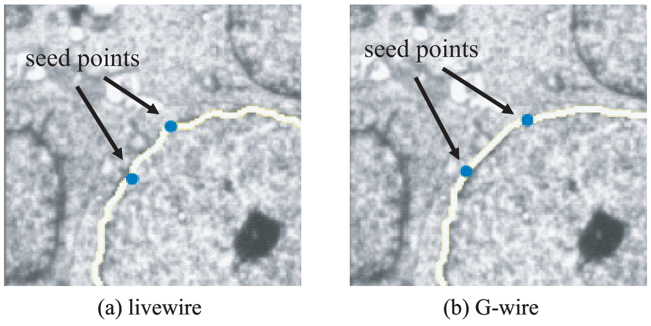


Figure 7. Smooth connection at the seed points

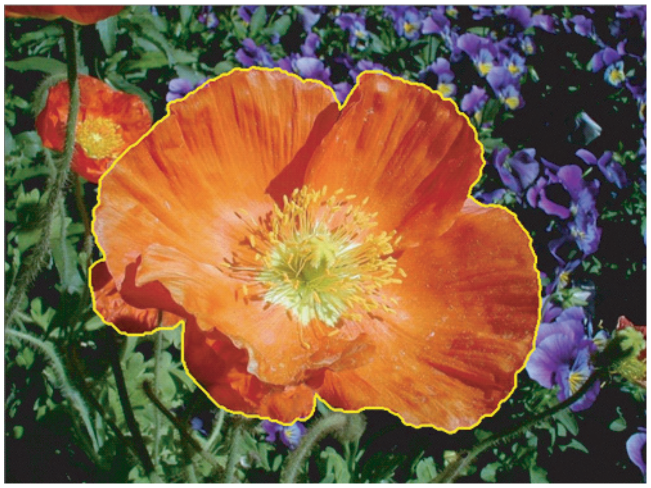


Figure 8. Turning off internal energy: $\alpha = 0, \beta = 0, \gamma = 1$

in this paper, it can be naturally extended to an even higher-dimensional case, such as segmentation of a boundary curve with multiscale internal energy. Also, based on the path map localization and incremental update strategy, our algorithm ensures the interactive speed of the segmentation regardless of the image size or the dimensionality of the graph.

6 Acknowledgment

This research is supported by the MIC (Ministry of Information and Communication), Korea, under ITRC (Information Technology Research Center) support program supervised by IITA (Institute of Information Technology Assessment).

References

[1] Adams, R., Bischof, L., 1994. Seeded Region Growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 6, pp. 641-647.

[2] Amini, A., Weymouth, T., Jain, R., 1990. Using Dynamic Programming for Solving Variational Problems in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 9, pp. 855-866.

[3] Bellman, R., Dreyfus, S., 1962. *Applied Dynamic Programming*. Princeton, NJ: Princeton University Press, 1962.

[4] Blake, A., Isard, M., 1998. *Active Contours*. Springer-Verlag, 1998.

[5] Boykov, Y., Jolly, M., 2001. Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D images. *Proc. International Conference on Computer Vision*, Vol. I, pp. 105-112.

[6] Boykov, Y., Kolmogorov, V., 2004. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 26, No. 9, pp. 1124-1137.

[7] Brigger, P., Hoeg, J., Unser, M., 2000. B-Spline Snakes: A Flexible Tool for Parametric Contour Detection. *IEEE Trnasactions on Image Processing*, Vol. 9, No. 9, pp. 1484-1496.

[8] Caselles, V., Kimmel, R., Sapiro, G., 1995. Geodesic active contours. *Proc. 5th International Conference on Computer Vision*, pp. 694-699.

[9] Cohen, L., 1991. On Active Contour Models and Balloons. *CVGIP: Image Understanding*, Vol. 53, No. 2, pp. 211-218.

[10] Cohen, L., Cohen, I., 1993. Finite element methods for active contour models and balloons for 2D and 3D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 11, pp. 1131-1147.

[11] Dijkstra, E., 1959. A Note on Two Problems in Connexion with Graphs. In *Numerical Mathematik*, Vol.1, pp. 269-270.

[12] Falcão, A., Udupa, J., Samarasekera, S., Sharma, S., Hirsch, B., Lotufo., R., 1998. User-Steered Image Segmentation Paradigms: Live Wire and Live Lane. *Graphical Models and Image Processing*, Vol. 60, No. 5, pp. 233-260.

[13] Falcão, A., Udupa, J., Miyazawa, F., 2000. An Ultra-Fast User-Steered Image Segmentation Paradigm: Live Wire on the Fly. *IEEE Transactions on Medical Imaging*, Vol. 19, No. 1, pp. 55-62.

[14] Geiger, D., Gupta, A., Costa, Luiz., Vlontzos, J., 1995. Dynamic Programming for Detecting, Tracking, and Matching Deformable Contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 3, pp. 294-302.

[15] Jelinek, F., 1998. *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, MA.

[16] Kang, H., Shin, S., 2002. Enhanced Lane: Interactive Image Segmentation by Incremental Path Map Construction. *Graphical Models*, Vol. 64, No. 5, pp. 282-303.

[17] Kass, M., Witkin, A., Terzopoulos, D., 1988. Snakes: Active Contour Models. *International Journal of Computer Vision*, Vol. 1, No. 4, pp. 321-331.

[18] Liang, J., McInerney, T., Terzopoulos, D., 1999. United Snakes. In *Proc. IEEE International Conference on Computer Vision (ICCV '99)*, pp. 933-940.

[19] McInerney, T., Terzopoulos, D., 1995. Topologically Adaptive Snakes. In *Proceedings of Fifth International Conference on Computer Vision*, pp. 840-845.

[20] Merlet, N., Zerubia, J., 1996. New Prospects in Line Detection by Dynamic Programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 4, pp. 426-431.

[21] Mortensen, E., Barrett, W., 1995. Intelligent Scissors for Image Composition. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pp. 191-198.

[22] Mortensen, E., Barrett, W., 1998. Interactive Segmentation with Intelligent Scissors. *Graphical Models and Image Processing*, Vol. 60, No. 5, pp. 349-384.

[23] Paragios, N., Deriche, R., 2000. Geodesic active contours and level sets for the detection and tracking of moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 3, pp. 266-280.

[24] Pardas, M., Sayrol, E., 2001. Motion estimation based tracking of active contours. *Pattern Recognition Letters*, Vol. 22, pp. 1447-1456.

[25] Scharstein, D., Szeliski, R., 2002. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, Vol. 47, No. 1-3, pp. 7-42.

[26] Staib., L., Duncan., J., 1992. Boundary finding with parametrically deformable models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 11, pp. 1061-1075.