Graph cut based image segmentation with connectivity priors

Sara Vicente* Vladimir Kolmogorov
University College London
{s.vicente,vnk}@adastral.ucl.ac.uk

Carsten Rother
Microsoft Research Cambridge
carrot@microsoft.com

Abstract

Graph cut is a popular technique for interactive image segmentation. However, it has certain shortcomings. In particular, graph cut has problems with segmenting thin elongated objects due to the "shrinking bias". To overcome this problem, we propose to impose an additional connectivity prior, which is a very natural assumption about objects. We formulate several versions of the connectivity constraint and show that the corresponding optimization problems are all NP-hard.

For some of these versions we propose two optimization algorithms: (i) a practical heuristic technique which we call DijkstraGC*, and (ii) a slow method based on problem decomposition which provides a lower bound on the problem. We use the second technique to verify that for some practical examples DijkstraGC is able to find the global minimum.*

1. Introduction

The task of interactive image segmentation has attracted a significant attention in recent years [10, 3, 18, 6, 24, 21]. The ultimate goal is to extract an object with as few user interactions as possible. It is widely accepted that some prior on segmentations is needed for achieving this goal. Different priors have a preference towards different types of shapes, as we discuss next.

Graph cut A very popular approach, which we also use in this paper, is based on graph cut [7, 3, 18]. It minimizes an energy function consisting of a data term (computed using color likelihoods of foreground and background) and a spatial coherency term. The latter term is the length of the boundary modulated with the contrast in the image, therefore minimizing the energy with this term has a bias towards shorter boundaries. (This behavior is sometimes referred to as the "shrinking bias".) In particular, it is hard for the graph cut approach to segment thin elongated structures. Consider Fig. 1. First the user constrains some pixels to be fore- and background using brushes (a). The segmentation by graph cut (b) cuts off some of the legs of the insect. If we reduce the influence of the coherency term then the legs get segmented but the overall quality of the the segmentation is decreased (c). This shows the trade-off between data terms and regularization, and it indicates that some form of coherency is crucial.

Alternative segmentation models One approach to overcome the shrinking bias is to add flux of some vector field to the model [10, 25, 12, 15]. It has been shown to be effective for segmenting thin objects such as blood vessels in grayscale images [25]. The vector field was taken as the image gradient, which corresponds to the assumption that the object is bright and the background is dark. However, extending this approach to arbitrary color images, which is the scenario considered in this paper, may be challenging. To our knowledge it was not addressed so far. The difficulty here is choosing the vector at each point and the sign of this vector. Imperfect vector field might lower the segmentation quality. The issue of choosing the sign can be overcome in the level set framework [12], but at the expense of losing global optimality.

One possible method to integrate flux into segmentation is to optimize the ratio of flux over boundary length [10, 16]. Thus, we are looking for the boundary with the highest average contrast. Arguably, this model has no bias towards any particular shape [10, 16]. However, the issue of choosing a good vector field for color images remains.

Other interesting approaches include the method in [21] which imposes a prior on the curvature of the boundary, spectral techniques [22] and the random walker algorithm [6]; results in [24] indicate that this method is slightly more robust towards the shrinking bias.

Our approach In this paper we propose a very different way to solve the task of segmenting challenging objects with very thin, elongated parts. We build the coherency prior in form of an **explicit connectivity prior** into the model. Assume that the user has already segmented a part of the object using graph cut [18] as in Fig. 1(b). In our interactive framework the user has to click only those pixels which must be connected to the main object. As Fig. 1(d) shows a few clicks are sufficient to obtain a satisfying result (e). We believe that this is a new and very powerful user interface for segmenting challenging objects.

We consider several versions of the connectivity constraint. Unfortunately, the corresponding optimization

*Sara Vicente is supported by Microsoft Research Cambridge through its PhD Scholarship Programme.

978-1-4244-2243-2/08/$25.00 ©2008 IEEE

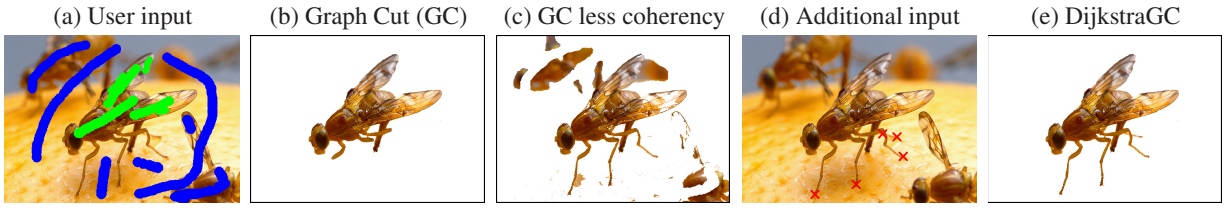


Figure 1. **Image segmentation using graph cut with standard (b) and reduced coherency (c) based on input (a). Our new DijkstraGC method (e) with additional user input (d).**

problems are all NP-hard, as we show. To enable the interface shown in Fig. 1 we propose a heuristic algorithm which we call *DijkstraGC*. On an abstract level it merges the Dijkstra algorithm and graph cut. Note that Dijkstra-like methods have already been used for extracting thin objects such as blood vessels [5], although without an explicit segmentation. (A fast marching technique was used in [5], which can be viewed as a continuous analogue of the Dijkstra algorithm for discrete graphs.) The key feature of our method that distinguishes it from [5] is the addition of the graph cut component. This allows to explicitly use the MAP-MRF formulation which proved to be very successful [3, 18].

We show that on some practical examples DijkstraGC is able to find the global minimum. In order to verify this, we developed a second (slow) technique based on *dual decomposition*, which provides a lower bound on the problem.

Related work Connectivity is automatically enforced in the classical "snakes" approach [11], since the segmentation is represented by a simple closed contour. Han et al. [9] proposed a topology preserving level set method which allows to specify more general topologies. A disadvantage of both techniques is that the objective is optimized via gradient descent, which can easily get stuck in a local minimum. Recently, Zeng et al. [29] followed a similar approach with a discrete graph-based formulation. After posing the problem the authors of [29] proved an NP-hardness result and proposed to modify the maxflow algorithm in [4] so that the topology of the segmentation is preserved. However, despite our best effort we were unable to compare it to our approach for the task of segmenting thin objects.[1] (Note, results in [29] are shown for very different types of objects.)

2. Problem formulation

We use an energy function of the form which is standard for graph cut based image segmentation approaches [3, 18]:

$$E(\boldsymbol{x}) = \sum_{p \in \mathcal{V}} E_p(x_p) + \sum_{(p,q) \in \mathcal{E}} E_{pq}(x_p, x_q) \qquad (1)$$

Here $(\mathcal{V}, \mathcal{E})$ is an undirected graph whose nodes correspond to pixels. $x_p \in \{0, 1\}$ is the segmentation label of pixel p, where 0 and 1 correspond to the background and the foreground, respectively. We assume that the pairwise terms E_{pq} are submodular, i.e. $E_{pq}(0,0) + E_{pq}(1,1) \leq E_{pq}(0,1) + E_{pq}(1,0)$.

As stated in the introduction, our goal is to minimize function $E(\boldsymbol{x})$ under certain connectivity constraints on the segmentation $\boldsymbol{x}$. Three possible constraints are formulated below. In all of them we assume that we are given an undirected graph $(\mathcal{V}, \mathcal{F})$ defining the "connectivity" relations between nodes in $\mathcal{V}$. This graph can be different from the graph $(\mathcal{V}, \mathcal{E})$ defining the structure of function $E(\boldsymbol{x})$ in eq. (1). (In our experiments we usually take $(\mathcal{V}, \mathcal{E})$ to be an 8-connected 2D grid graph and $(\mathcal{V}, \mathcal{F})$ to be 4-connected.)

Perhaps, the most natural connectivity constraint is the following:

C0 *The set* $[\boldsymbol{x}]$ *corresponding to segmentation* $\boldsymbol{x}$ *must form a single connected component in the graph* $(\mathcal{V}, \mathcal{F})$.

(We denoted $[\boldsymbol{x}]$ to be the set of nodes with label 1, i.e. $[\boldsymbol{x}] = \{p \in \mathcal{V} \mid x_p = 1\}$.) This constraint seems to be very useful for solving problems discussed in the introduction. However, minimizing function (1) under the constraint **C0** appears to be a very challenging task. This problem can be shown to be NP-hard even if function (1) has only unary terms (see below).

In this paper we will focus on different constraints **C1** and **C2**. We will assume that the user specified two nodes $s, t \in \mathcal{V}$. Constraint **C1** is then formulated as follows:

C1 *Nodes* s, t *must be connected in the segmentation set* $[\boldsymbol{x}]$*, i.e. there must exist a path in the graph* $(\mathcal{V}, \mathcal{F})$ *from* s *to* t *such that all nodes* p *in the path belong to the segmentation:* $x_p = 1$.

We believe that **C1** is very useful for interactive image segmentation. It suggests a natural user interface (Fig. 1). In this interface node s is assumed to lie in the largest connected component of the current segmentation. By clicking at pixel t the user would get a segmentation which connects t to the main object. We handle multiple clicks in an incremental fashion.

Unfortunately, minimizing (1) under **C1** is an NP-hard problem as well (see below). However, it appears that it is easier to design good heuristic algorithms for **C1** than for **C0**. In particular, if function $E(\boldsymbol{x})$ has only unary terms

[1] We downloaded the source code (ver. 0.9) but did not succeed in applying it to our examples: sometimes user-provided hard constraints were not satisfied, or the segmented thin structure was clearly incorrect. Reimplementing the algorithm in [29] did not look straightforward - we found that many details were missing.

then the problem with **C1** can be reduced to a shortest path computation with a single source and a single sink and thus can be solved in polynomial time (see section 3).

Enforcing constraint **C1** may result in a segmentation which has a "width" of one pixel in certain places, which may be undesirable (see Fig. 6). One way to fix this problem is to allow the user to specify a parameter δ which controls the minimum "width" of the segmentation. Formally, assume that for each node $p \in \mathcal{V}$ we have a subset $\mathcal{Q}_p \subseteq \mathcal{V}$. (This subset would depend on δ; for example, for a grid graph $\mathcal{Q}_p$ could be the set of all pixels q such that the distance from p to q does not exceed δ.) Using these subsets, we define the following connectivity constraint:

C2 *There must exist a path in the graph $(\mathcal{V}, \mathcal{F})$ from s to t such that for all nodes p in the path the subset $\mathcal{Q}_p$ belongs to $[\boldsymbol{x}]$, i.e. $x_q = 1$ for $q \in \mathcal{Q}_p$.*

Clearly, **C1** is a special case of **C2** if we choose $\mathcal{Q}_p = \{p\}$ for all nodes p.

Throughout the paper, we denote **P0**, **P1**, **P2** to be the problems of minimizing function (1) under constraints **C0**, **C1**, **C2**, respectively. The theorem below shows the difficulty of the problems; its proof is given in [26].

Theorem 1. *Problems* **P0**, **P1**, **P2** *are NP-hard.* **P0** *and* **P2** *remain NP-hard even if the set $\mathcal{E}$ is empty, i.e. function* (1) *does not have pairwise terms.*

Note, it was also shown in [29] that the following problem is NP-hard: minimize function (1) on a planar 2D grid so that the foreground is 4-connected and the background is 8-connected. It is straightforward to modify the argument in [29] to show that the problem is NP-hard if only the 4-connectedness of the foreground is imposed (in other words, **P0** is NP-hard even for planar 2D grids).

To conclude this section, we will state some simple facts about the relationship of problems **P0**-**P2** and the problem of minimizing function $E(\boldsymbol{x})$ without any constraints.

Theorem 2. *Suppose that $\boldsymbol{x}$ is a global minimum of function* (1) *without any constraints.*

(a) There exists an optimal solution $\boldsymbol{x}^*$ of **P2** *which includes $\boldsymbol{x}$, i.e. $[\boldsymbol{x}] \subseteq [\boldsymbol{x}^*]$. The same holds for the problem* **P1** *since the latter is a special case.*

(b) Suppose that $\mathcal{E} \subseteq \mathcal{F}$. Let $\mathcal{C}_1, \ldots, \mathcal{C}_k \subseteq V$ be the connected components of the set $[\boldsymbol{x}]$ in the graph $(\mathcal{V}, \mathcal{F})$. Then there exists an optimal solution $\boldsymbol{x}^*$ of **P0** *such that each component $\mathcal{C}_i$ is either entirely included in $[\boldsymbol{x}^*]$ or entirely excluded. In other words, if $\mathcal{C}_i$ and $[\boldsymbol{x}^*]$ intersect then $\mathcal{C}_i \subseteq [\boldsymbol{x}^*]$.*

A proof is given in [26]. The theorem suggests that as a first step we could run the maxflow algorithm to minimize function (1) without any constraints and then contract connected components of the obtained set $[\boldsymbol{x}]$ to single nodes. However, it leaves open the most challenging question: what to do if a minimum of function (1) does not satisfy the desired connectivity constraint.

initialize: $\mathcal{S} = \varnothing$, $PARENT(p) = NULL$ for all nodes p,
$d(s) = \min\{E(\boldsymbol{x}) \mid \mathcal{Q}_s \subseteq [\boldsymbol{x}]\}$,
$d(p) = +\infty$ for $p \in \mathcal{V} - \{s\}$

while $t \notin \mathcal{S}$ and $\mathcal{V} - \mathcal{S}$ contains nodes p with $d(p) < +\infty$

- find node $p \in \mathcal{V} - \mathcal{S}$ with the smallest distance $d(p)$
- add p to $\mathcal{S}$
- **for** all nodes $q \in \mathcal{V} - \mathcal{S}$ which are neighbors of p (i.e. $(p, q) \in \mathcal{F}$) **do**
  - using $PARENT$ pointers, get path $\mathcal{P}$ from s to q through p; compute corresponding set $\bar{\mathcal{P}} = \cup_{r \in \mathcal{P}} \mathcal{Q}_r$
  - compute a minimum $\boldsymbol{x}$ of function (1) under the constraint $\bar{\mathcal{P}} \subseteq [\boldsymbol{x}]$
  - if $d(q) > E(\boldsymbol{x})$ set $d(q) := E(\boldsymbol{x})$, $PARENT(q) := p$

Figure 2. **DijkstraGC algorithm.**

3. Algorithms

The main algorithmic contribution of this paper is a heuristic method for the problem **P2** (and thus for **P1** since the latter is a special case). This method, which we call *DijkstraGC*, is presented in section 3.1. Then in section 3.2 we propose an alternative method for a special case of problem **P1** based on the idea of *problem decomposition*. The main feature of the second technique is that it provides a lower bound on the optimal value of **P1**. We will use it for assessing the performance of DijkstraGC: in the experimental section it will help us to verify that for some instances DijkstraGC gives an optimal solution.

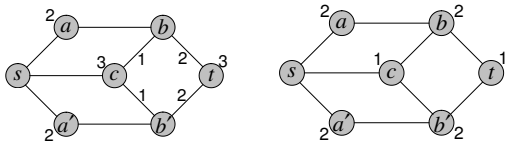
3.1. DijkstraGC: Merging Dijkstra and graph cuts

The idea of our first method is motivated by the Dijkstra algorithm [1]. Recall that the latter technique computes shortest distances $d(p)$ in a directed graph with non-negative weights from a specified "source" node s to all other nodes p.

Similar to the Dijkstra method, we will compute solutions to the problem **P2** for a fixed node s and all nodes $p \in \mathcal{V}$ (only now these solutions will not necessarily be global minima). The "distance" $d(p)$ will now indicate the cost of the computed solution for the pair of nodes $\{s, p\}$.

The algorithm is shown in Fig. 2. During the algorithm, the current solution $\boldsymbol{x}^p$ for node p with $d(p) < +\infty$ can be obtained as follows: using $PARENT$ pointers get path $\mathcal{P}$ and corresponding set $\bar{\mathcal{P}} = \cup_{r \in \mathcal{P}} \mathcal{Q}_r$, and then compute a minimum of function (1) under the constraint $\bar{\mathcal{P}} \subseteq [\boldsymbol{x}]$. Clearly, the obtained solution $\boldsymbol{x}^p$ satisfies the hard constraint **C2** for the pair of nodes $\{s, p\}$.

The set $\mathcal{S}$ contains "permanently labeled" nodes: once a node p has been added to $\mathcal{S}$, its cost $d(p)$ and the corre-



$\mathcal{Q}_c = \{c, b, b'\}$, $\mathcal{Q}_t = \{t, b, b'\}$
$\mathcal{Q}_p = \{p\}$ for all other nodes p

(a) Problem **P1** (b) Problem **P2**, no pairwise terms

Figure 3. **Suboptimality of DijkstraGC.** *Examples of problems on which DijkstraGC give suboptimal results. Graphs shown in the pictures are the connectivity graphs $(\mathcal{V}, \mathcal{F})$. Number c_p at node p gives the unary term cx_p, number c_{pq} at edge (p, q) gives the pairwise term $c_{pq}|x_q - x_p|$. Both in (a) and (b) DijkstraGC will output solution $\{s,a,b,b',t\}$ or $\{s,a',b,b',t\}$ with cost 7, while the optimal solution $\{s,c,b,b',t\}$ has cost 6.*

sponding solution will not change anymore.

Let us list some of the invariants that are maintained during DijkstraGC (they follow directly from the description):

I1 If $d(p) = +\infty$ then $p \neq s$ and $PARENT(p) = NULL$.

I2 If $d(p) < +\infty$ then $PARENT$ pointers give the unique path $\mathcal{P}$ from s to p, and $d(p) = \min\{E(\boldsymbol{x}) \mid \bar{\mathcal{P}} \subseteq [\boldsymbol{x}]\}$ where $\bar{\mathcal{P}} = \cup_{r \in \mathcal{P}} \mathcal{Q}_r$.

I3 If $PARENT(q) = p$ then $d(p) \leq d(q) < +\infty$.

I4 $d(p) < +\infty$ for nodes $p \in \mathcal{S}$.

Theorem 3. *If function $E(\boldsymbol{x})$ does not have pairwise terms and $\mathcal{Q}_p = \{p\}$ for all nodes p (i.e we have an instance of* **P1***) then the algorithm in Fig. 2 produces an optimal solution.*

A proof is given in [26]. After submission we also found another special case in which DijkstraGC gives an optimal result (see [26]).

If conditions of the theorem are relaxed then the problem may become NP-hard, as theorem 1 states. Not surprisingly, DijkstraGC may then produce a suboptimal solution. Two examples are shown in Fig. 3. Note that in these examples the "direction" of DijkstraGC matters: if we run it from s to t then we obtain a suboptimal solution, but running DijkstraGC from t to s will give an optimal segmentation.

We now turn to the question of efficient implementation. One computational component of the algorithm is finding a node $p \in \mathcal{V} - \mathcal{S}$ with the smallest value of $d(p)$ (same as in the Dijkstra algorithm). We used a binary heap structure for implementing the priority queue which stores nodes $p \in \mathcal{V} - \mathcal{S}$ with $d(p) < +\infty$. The bottleneck, however, is maxflow computations: DijkstraGC requires many calls to the maxflow algorithm for minimizing function (1) under the constraints $x_r = 1$ for nodes $r \in \bar{\mathcal{P}}$. These computations are considered in the remainder of this section.

Optimized DijkstraGC First, we will describe a technique which allows to reduce the number of calls to maxflow. Consider the step that adds node p to the set of permanently labeled nodes $\mathcal{S}$. Denote $\mathcal{P}$ to be the path from s to p given by $PARENT$ pointers, and let $\bar{\mathcal{P}} = \cup_{r \in \mathcal{P}} \mathcal{Q}_r$. Let us fix nodes in $\bar{\mathcal{P}}$ to 1 and compute a minimum $\boldsymbol{x}$ of function (1) under these constraints. The segmentation set $[\boldsymbol{x}]$ will contain $\bar{\mathcal{P}}$, but it may include many other nodes as well. Then it might be possible to add several nodes to $\mathcal{S}$ using this single computation. Indeed, suppose p has a neighbor $q \in \mathcal{V} - \mathcal{S}$, $(p, q) \in \mathcal{F}$, such that $\mathcal{Q}_q \subseteq [\boldsymbol{x}]$. The algorithm in Fig. 2 would set $d(q) = d(p) = E(\boldsymbol{x})$ while exploring neighbors of p. This would make the distance $d(q)$ to be the smallest among nodes in $\mathcal{V} - \mathcal{S}$, so the node q could be the next node to be added to $\mathcal{S}$. Therefore, we can add q to $\mathcal{S}$ immediately.

An algorithm which implements this idea is shown in Fig. 4. Before exploring neighbors of q, we check which nodes can be added to $\mathcal{S}$ for "free". The set of these nodes is denoted as $\mathcal{A}$; clearly, it includes p. After adding nodes in $\mathcal{A}$ to $\mathcal{S}$, we explore neighbors of $\mathcal{A}$ which are still in $\mathcal{V} - \mathcal{S}$.

Note that there is a certain freedom in implementing the DijkstraGC algorithm: it does not specify which node $p \in \mathcal{V} - \mathcal{S}$ with the minimum distance to choose if there are

initialize: $\mathcal{S} = \varnothing$, $PARENT(p) = NULL$ for all nodes p,
$d(s) = \min\{E(\boldsymbol{x}) \mid \mathcal{Q}_s \subseteq [\boldsymbol{x}]\}$,
$d(p) = +\infty$ for $p \in \mathcal{V} - \{s\}$

while $t \notin \mathcal{S}$ and $\mathcal{V} - \mathcal{S}$ contains nodes p with $d(p) < +\infty$

- find node $p \in \mathcal{V} - \mathcal{S}$ with the smallest distance $d(p)$
- using $PARENT$ pointers, get path $\mathcal{P}$ from s to p; compute corresponding set $\bar{\mathcal{P}} = \cup_{r \in \mathcal{P}} \mathcal{Q}_r$
- compute a minimum $\boldsymbol{x}$ of function (1) under the constraint $\bar{\mathcal{P}} \subseteq [\boldsymbol{x}]$
- add p to $\mathcal{S}$, set $\mathcal{A} = \{p\}$, mark p as "unprocessed"
- **while** $\mathcal{A}$ has unprocessed nodes
  - pick unprocessed node $p' \in \mathcal{A}$
  - **for** all edges $(p', q) \in \mathcal{F}$ with $q \in \mathcal{V} - \mathcal{S}$ **do**
    - $\diamond$ if $\mathcal{Q}_q \subseteq [\boldsymbol{x}]$ set $d(q) := E(\boldsymbol{x})$, $PARENT(q) := p'$, add q to $\mathcal{S}$ and to $\mathcal{A}$ as an unprocessed node
  - mark p' as "processed"
- **for** all nodes $q \in \mathcal{V} - \mathcal{S}$ which are neighbors of $\mathcal{A}$ (i.e. $(p', q) \in \mathcal{F}$ for some node $p' \in \mathcal{A}$) **do**
  - pick node $p' \in \mathcal{A}$ with $(p', q) \in \mathcal{F}$
  - using $PARENT$ pointers, get path $\mathcal{P}$ from s to q through p'; compute corresponding set $\bar{\mathcal{P}} = \cup_{r \in \mathcal{P}} \mathcal{Q}_r$
  - compute a minimum $\boldsymbol{x}$ of function (1) under the constraint $\bar{\mathcal{P}} \subseteq [\boldsymbol{x}]$
  - if $d(q) > E(\boldsymbol{x})$ set $d(q) := E(\boldsymbol{x})$, $PARENT(q) := p'$

Figure 4. **Optimized version of the DijkstraGC algorithm.**

several such nodes. It is not difficult to see that under a certain selection rule DijkstraGC becomes equivalent to the algorithm in Fig. 4.

Flow and search tree recycling We used the maxflow algorithm in [4], and reused flows and search trees as described in [13].

In DijkstraGC we often need to fix/unfix nodes in different parts of the graph in a rather chaotic order. We believe that this significantly reduces the effectiveness of flow and search tree recycling. Two ideas could potentially be used to overcome this drawback. The first one is based on the observation that different "branches" are often independent in a certain sense.This could allow to reorder maxflow computations. To get the same type of result as DijkstraGC we would need to redo computations if we detect an inconsistency, as in the Bellman-Ford label-correcting algorithm. The second idea is to maintain multiple graphs for performing computations in different parts of the image, so that changes in each graph would be more "local". It could also be feasible to store a small subset of the nodes for each graph, increasing it "on demand". Reduced memory requirements could then allow to use a larger number of graphs. Exploring these ideas is left as a future work.

3.2. Problem decomposition approach

In this section we propose a different technique for a special case of problem **P1**; we will use it for assessing the performance of DijkstraGC.

Overview On the high level, the idea is to decompose the original problem into several "easier" subproblems, for which we can compute efficiently a global minimum (or obtain a good lower bound). Combining the lower bounds for individual subproblems will then provide a lower bound for the original problem. The decomposition and the corresponding lower bound will depend on a parameter vector θ; we will then try to find a vector θ that maximizes the bound.

This approach is well-known in combinatorial optimization; sometimes it is referred to as "dual decomposition" [2]. In vision the decomposition approach is probably best known in the context of the MAP-MRF inference task. It was introduced by Wainwright et al. [27] who decomposed the problem into a convex combination of trees and proposed message passing techniques for optimizing vector θ. These techniques do not necessarily find the best lower bound (see [14] or review article [28]). Schlesinger and Giginyak [19, 20] and Komodakis et al. [17] proposed to use subgradient techniques [23, 2] for MRF optimization, which guarantee to converge to a vector θ yielding the best possible lower bound.

Solving P1 via problem decomposition We now apply this approach to **P1**. To get tractable subproblems, we impose the following simplifying assumptions. First, we assume that the graph $(\mathcal{V}, \mathcal{F})$ is planar, and $\mathcal{E} = \mathcal{F}$. Second, we assume that pixels on the image boundary are constrained to be background, i.e. their label is 0. We argue that these assumptions represent an important practical subclass of the image segmentation task, and thus can be used for assessing the performance of DijkstraGC for real problems. Note that the second assumption encodes the prior knowledge that the object lies entirely inside the image, which is very often the case in practice.

We denote $C(\boldsymbol{x})$ to be the hard constraint term which is 0 if the segmentation $\boldsymbol{x}$ satisfies the connectivity constraint **C1** and the background boundary condition described above, and otherwise $C(\boldsymbol{x})$ is $+\infty$. Some of these hard constraints will also be included in function $E(\boldsymbol{x})$ as unary terms, namely the background boundary constraints and foreground constraints $x_s = x_t = 1$, which follow from **C1**. Our parameter vector θ will have two parts: $\theta = (\theta^1, \theta^2)$ where vectors θ^1 and θ^2 correspond to nodes and edges of the graph $(\mathcal{V}, \mathcal{E})$, respectively ($\theta^1 \in \mathbb{R}^{\mathcal{V}}$, $\theta^2 \in \mathbb{R}^{\mathcal{E}}$). Given labeling $\boldsymbol{x}$, let $\phi(\boldsymbol{x}) \in \{0,1\}^{\mathcal{E}}$ be the vector of indicator variables showing discontinuities of $\boldsymbol{x}$, i.e. $\phi_{pq}(\boldsymbol{x}) = |x_q - x_p|$ for an edge $(p,q) \in \mathcal{E}$. We will use the following decomposition:

$$E(\boldsymbol{x}) + C(\boldsymbol{x}) = E^0(\boldsymbol{x} \mid \theta) + E^1(\boldsymbol{x} \mid \theta) + E^2(\boldsymbol{x} \mid \theta) \quad (2)$$

where

$$E^0(\boldsymbol{x} \mid \theta) = E(\boldsymbol{x}) - \langle \boldsymbol{x}, \theta^1 \rangle - \langle \phi(\boldsymbol{x}), \theta^2 \rangle \quad (2a)$$

$$E^1(\boldsymbol{x} \mid \theta) = C(\boldsymbol{x}) + \langle \boldsymbol{x}, \theta^1 \rangle \quad (2b)$$

$$E^2(\boldsymbol{x} \mid \theta) = C(\boldsymbol{x}) + \langle \phi(\boldsymbol{x}), \theta^2 \rangle \quad (2c)$$

Let us discuss each subproblem in more detail.

Subproblem 0 Function $E^0(\boldsymbol{x} \mid \theta)$ consists of unary and pairwise terms. We will require this function to be submodular; this is equivalent to specifying upper bounds on components θ^2_{pq}. Since there are no connectivity constraints, we can compute the global minimum $\Phi^0(\theta) = \min_{\boldsymbol{x}} E^0(\boldsymbol{x} \mid \theta)$ using a maxflow algorithm[2].

Subproblem 1 Function $E^1(\boldsymbol{x} \mid \theta)$ has only unary terms and the connectivity constraint **C1**. As discussed in the previous section, we can compute the global minimum $\Phi^1(\theta) = \min_{\boldsymbol{x}} E^1(\boldsymbol{x} \mid \theta)$ using, e.g. DijkstraGC algorithm. Note, in this case it is essentially equivalent to the Dijkstra algorithm.

Subproblem 2 We will require vector θ^2 to be non-negative. We compute a lower bound $\Phi^2(\theta)$ on $E^2(\boldsymbol{x} \mid \theta^2)$ using a very fast technique whose details are given in [26]. In short, we compute two edge disjoint paths of minimum cost in the dual graph from a set of nodes "behind" node s to a set of nodes "behind" node t. (This is motivated by the

[2]Instead of restricting function E^0 to be submodular, one could use the roof duality approach [8] to get a lower bound on $E^0(\boldsymbol{x} \mid \theta)$. For submodular functions this lower bound coincides with the global minimum, therefore the best lower bound on the original function can only become better. We have not implemented this yet.

fact that an optimal segmentation can be viewed as a simple closed contour going "around" s and t.)

Maximizing the lower bound We described a lower bound on problem **P1** which can be written as

$$\Phi(\theta) = \Phi^0(\theta) + \Phi^1(\theta) + \Phi^2(\theta) \leq E(\boldsymbol{x}) + C(\boldsymbol{x})$$

where θ belongs to a convex set $\Omega = \{(\theta^1, \theta^2 \mid 0 \leq \theta^2_{pq} \leq \theta^{2\max}_{pq}\}$. Clearly, Φ is a concave function of θ. Similar to [19, 20, 17], we used a projected subgradient method [23, 2] for maximizing $\Phi(\theta)$. Details of our implementation and the procedure for choosing solution $\boldsymbol{x}$ are given in [26].

4. Experimental results

In the previous section we presented DijkstraGC, a new algorithm that minimizes energy (1) under certain connectivity constraints on the segmentation $\boldsymbol{x}$. In this section we first discuss the advantages of including this algorithm in an interactive system for image segmentation and second consider the optimality properties of the algorithm.

4.1. DijkstraGC for interactive segmentation

The form of the energy (1) follows the approach of previous energy minimization techniques for interactive image segmentation [3, 18]. We define $E_p(x_p)$ as a data likelihood term and $E_{pq}(x_p, x_q)$ as a contrast-dependent coherency term, which are defined as follows.

Hard constraints for background and foreground are specified in the form of brush strokes. Based on this input a probabilistic model is computed for the colors of background (G_B) and foreground (G_F) using two different Gaussian Mixture Models. $E_p(x_p)$ is then computed as $E_p(0) = -\log(\Pr(z_p|G_B))$ and $E_p(1) = -\log(\Pr(z_p|G_F))$ where z_p contains the three color channels of site p (see details in [18]). The coherency term incorporates both an Ising prior and a contrast-dependent component and is computed as

$$E_{pq}(x_p, x_q) = \frac{|x_q - x_p|}{\operatorname{dist}(p,q)} \left(\lambda_1 + \lambda_2 \exp -\beta \left\|z_p - z_q\right\|^2\right)$$

where λ_1 and λ_2 are weights for the Ising and contrast-dependent prior respectively, and $\beta = \left(2\left\langle (z_p - z_q)^2 \right\rangle\right)^{-1}$, where $\langle\cdot\rangle$ denotes expectation over an image sample (as motivated in [18]). A term of this form encourages coherence in regions of similar color and also prevents isolated pixels to appear in the segmentation (see [3, 18]). In our experiments the number of components used for G_B and G_F were 5, we fixed $\lambda_1 = 2.5$ and $\lambda_2 = 47.5$ (which sums up to 50, as in [18]). We used an 8-neighborhood system for E.

We now discuss how to integrate the DijkstraGC algorithm in an interactive system for image segmentation. After the user has provided scribbles a segmentation is computed with graph cut. As in [18] we iterate this process to further minimize the energy, where the segmentation of a previous run is used to update color models. It can happen that part of the foreground is missing or that the foreground region is disconnected. Then the user can specify with one click such a site that should be connected with the current result. DijkstraGC algorithm is used to compute the new segmentation. In this way the user only has to specify one node (from the two nodes necessary to run DijkstraGC) since the other node is assumed to be contained within the largest connected component of the graph cut segmentation.

We have tested this approach on 15 images with in total 40 connectivity problems, i.e. additional clicks for DijkstraGC. Fig. 1 and 5 show some results, where we compare graph cut, using scribbles only, with DijkstraGC, where the user set additional clicks after obtaining the graph cut result. We see that usually graph cut based algorithms tend to cut off thin elongated structures in the image. To retrieve these thin structures using brush strokes can be very difficult since they may only be $1-2$ pixel wide. To obtain a satisfying result with DijkstraGC the user only needs some additional clicks and the selection of a width parameter δ, which is a considerable reduction in the amount of user interactions needed. For the last example in Fig. 5 the number of clicks necessary to extract the segmentation was 11 since the thin structures we want to segment (the legs of the spider) intersect each other and the path that DijkstraGC computes goes throw the already segmented leg.

The running time presented in the last column of Fig. 5 includes all the clicks in the image, and it is, as to be expected, related to the number of clicks and image size. The optimized version of DijkstraGC (Fig. 4) improved the runtime over the simple version (Fig. 2) from, e.g. 28.4 to 14.8 seconds for the last image in Fig. 5.

The width parameter δ provides the possibility of specifying a minimum desired width of the connection between the two components. This parameter is not included directly in the formulation of the DijkstraGC algorithm. Instead we define for all nodes p a set $\mathcal{Q}_p$ according to δ. For $\delta = 1$, $\mathcal{Q}_p = \{p\}$; for $\delta = 2$, $\mathcal{Q}_p$ is the set of 4 nodes in a 2×2 square that includes node p and for $\delta = 3$, $\mathcal{Q}_p$ contains p and its neighbors in a 4-connected grid. Fig. 6 shows that this parameter can be important in a practical system to avoid that the connectivity constraint is satisfied by a segmentation with a one pixel width only. Please note that in general δ does not have to be the exact width of the structure we want to segment. In fig. 6 setting the width parameter to $\delta = 2$ was sufficient to recover the thin leg which has a larger width than 5 pixels.

Direction of DijkstraGC. Swapping the nodes s and t, i.e. changing the direction of DijkstraGC, may lead to two different segmentations as seen in the example of fig. 3. However we observed that the two segmentations usually only differ by a small number of pixels (on average less than

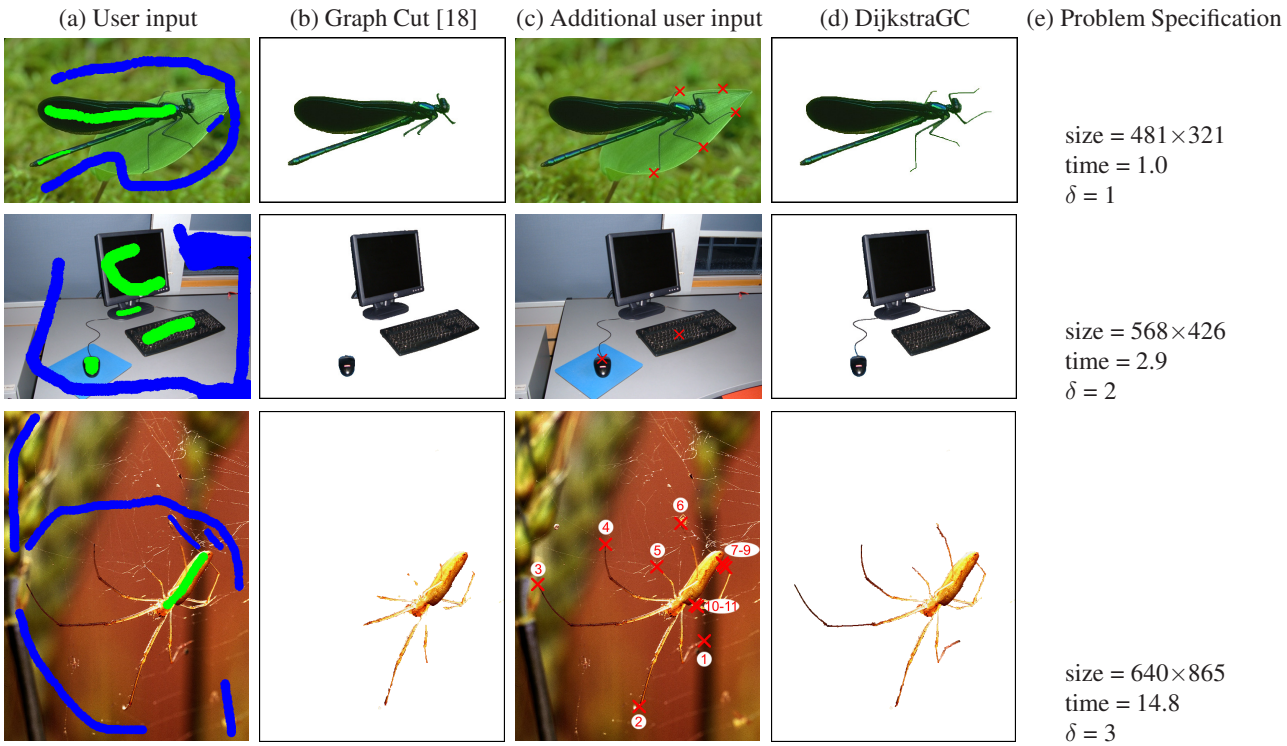


Figure 5. **Results of the DijkstraGC algorithm.** *(a) original images with user scribbles (blue background; green foreground); (b) Graph Cut results using [18]; (c) Selection of sites for connectivity, where numbers present the input order; (d) DijkstraGC results; (e) Problem specification: image size, running time for DijkstraGC (on 2.16 GHz CPU with 2GB RAM), and minimum width specified by the user.*

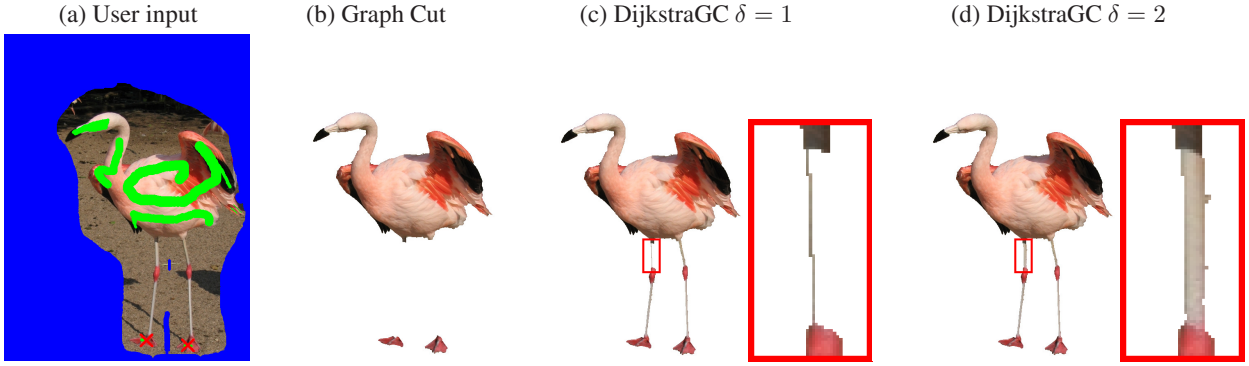


Figure 6. **Width parameter δ.** *Two different results obtained with DijkstraGC algorithm for different values of δ (minimum width).*

1% of the number of pixels in set $[\boldsymbol{x}]$) and the difference is often not visually significant.

In contrast, the difference in speed can be substantial. In our examples the running time was on average reduced by half if the "source" node s was in the smaller component (out of the two components that we want to connect). Accordingly, we chose it as the default option and used it for the results presented in Fig. 5 and 6.

4.2. Optimality of DijkstraGC

The dual decomposition algorithm, described in section 3.2, gives both a solution for a special case of **P1** and a lower bound on the optimal value of **P1**. Although this technique is not useful for a practical system, since the running time is on average 3 hours, it can be used to assess the optimality of DijkstraGC.

We considered 40 connectivity problems (i.e. user clicks) where the dual decomposition approach is applicable, i.e. all pixels at the image boundary are background. Another restriction for this approach is that we have to use a planar graph (4-connected 2D grid) for maxflow computations. For 12 out of the 40 problems the dual decomposition algorithm gave the global optimum. It is a positive result that for

all these 12 cases also DijkstraGC returned the global optimum. The first image in Fig. 5 is one of the examples for which we obtained the global optimum for all the connectivity constraints. (Note that the result is slightly different from the one presented, since for this optimality experiment we had to choose the graph to be planar, i.e. 4-connected.) For all the other problems we observed that the result provided by DijkstraGC was always better in terms of energy value than the result of the dual decomposition method.

5. Conclusions and Future Work

In this paper we proposed to overcome the "shrinking bias" of graph cut methods by imposing connectivity constraints in the segmentation. We presented a new algorithm DijkstraGC that computes a segmentation satisfying those constraints and we showed that integrating this algorithm in an interactive system for image segmentation reduces considerably the amount of user interaction necessary to segment thin structures in the image.

Although in general DijkstraGC is not guaranteed to compute the global minimum of our NP-hard optimization problem, we believe that in practice it is not an issue. This claim is supported by two facts: (i) running DijkstraGC in different directions gives almost the same result, and (ii) DijkstraGC computes the optimal solution for some particular instances (see sec. 4.2).

Currently, the speed of DijkstraGC is perhaps the main drawback for a practical interactive segmentation system. However, we believe that there is a large scope for improvement via rearrangement of the order in which nodes are visited during the algorithm, or the use of multiple graphs for maxflow computations (sec. 3.1). We intend to explore these ideas in the future.

References

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[3] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *ICCV*, 2001.

[4] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9), Sept. 2004.

[5] T. Deschamps and L. D. Cohen. Fast extraction of minimal paths in 3D images and applications to virtual endoscopy. *Medical Image Analysis*, 5(4):281–299, 2001.

[6] L. Grady. Random walks for image segmentation. *PAMI*, 28(11):1768–1783, Nov. 2006.

[7] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *J. of the Royal Statistical Society, Series B*, 51(2):271–279, 1989.

[8] P. L. Hammer, P. Hansen, and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematicl Programming*, 28:121–155, 1984.

[9] X. Han, C. Xu, and J. L. Prince. A topology preserving level set method for geometric deformable models. *PAMI*, 25(6):755–768, 2003.

[10] I. Jermyn and H. Ishikawa. Globally optimal regions and boundaries as minimum ratio weight cycles. *PAMI*, 23(10), Oct. 2001.

[11] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *IJCV*, 1(4):321–331, 1987.

[12] R. Kimmel and A. M. Bruckstein. On regularized Laplacian zero crossings and other optimal edge integrators. *IJCV*, 53(3):225–243, 2003.

[13] P. Kohli and P. H. S. Torr. Efficiently solving dynamic Markov random fields using graph cuts. In *ICCV*, 2005.

[14] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *PAMI*, 28(10), 2006.

[15] V. Kolmogorov and Y. Boykov. What metrics can be approximated by geo-cuts, or global optimization of length/area and flux. In *ICCV*, 2005.

[16] V. Kolmogorov, Y. Boykov, and C. Rother. Applications of parametric maxflow in computer vision. In *ICCV*, 2007.

[17] N. Komodakis, N. Paragios, and G. Tziritas. MRF optimization via dual decomposition: Message-passing revisited. In *ICCV*, 2005.

[18] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. *SIGGRAPH*, August 2004.

[19] M. I. Schlesinger and V. V. Giginyak. Solution to structural recognition (MAX,+)-problems by their equivalent transformations. Part 1. *Control Systems and Computers*, (1):3–15, 2007.

[20] M. I. Schlesinger and V. V. Giginyak. Solution to structural recognition (MAX,+)-problems by their equivalent transformations. Part 2. *Control Systems and Computers*, (2):3–18, 2007.

[21] T. Schoenemann and D. Cremers. Introducing curvature into globally optimimal image segmentation: Minimum ratio cycles on product graphs. In *ICCV*, Oct. 2007.

[22] J. Shi and J. Malik. Normalized cuts and image segmentation. *PAMI*, 22(8):888–905, Aug. 2000.

[23] N. Z. Shor. *Minimization methods for nondifferentiable functions*. Springer-Verlag, 1985.

[24] A. K. Sinop and L. Grady. A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm. In *ICCV*, Oct. 2007.

[25] A. Vasilevskiy and K. Siddiqi. Flux maximizing geometric flows. *PAMI*, 24(12), 2002.

[26] S. Vicente, V. Kolmogorov, and C. Rother. Graph cut based image segmentation with connectivity priors. Technical report, UCL, 2008.

[27] M. Wainwright, T. Jaakkola, and A. Willsky. MAP estimation via agreement on trees: Message-passing and linear-programming approaches. *IEEE Trans. Information Theory*, 51(11):3697–3717, 2005.

[28] T. Werner. A linear programming approach to max-sum problem: A review. *PAMI*, 29(7), 2007.

[29] Y. Zeng, D. Samaras, W. Chen, and Q. Peng. Topology cuts: A novel min-cut/max-flow algorithm for topology preserving segmentation in N-D images. Technical report, 2007.