Toward Automated Evaluation of Interactive Segmentation

Kevin McGuinness[a,*], Noel E. O'Connor[a]

[a] *Center for Digital Video Processing, CLARITY: Centre for Sensor Web Technologies, Dublin City University, Glasnevin, Dublin 9, Ireland*

Abstract

We previously described a system for evaluating interactive segmentation by means of user experiments [1]. This method, while effective, is time-consuming and labor-intensive. This paper aims to make evaluation more practicable by investigating if it is feasible to automate user interactions. To this end, we propose a general algorithm for driving the segmentation that uses the ground truth and current segmentation error to automatically simulate user interactions. We investigate four strategies for selecting which pixels will form the next interaction. The first of these is a simple, deterministic strategy; the remaining three strategies are probabilistic, and focus on more realistically approximating a real user. We evaluate four interactive segmentation algorithms using these strategies, and compare the results with our previous user experiment-based evaluation. The results show that automated evaluation is both feasible and useful.

Keywords: Image segmentation, Interactive segmentation, Evaluation, Simulation, Automation, User experiments

1. Introduction

Computer vision algorithms designed to emulate human perception are an active research area. Many of the problems tackled by these algorithms are ill-posed, in the sense that they do not have a unique solution, and accordingly

*Corresponding author. Tel.: +353 1 7006830. Fax: +353 1 7005442.

Email addresses: kevin.mcguinness@eeng.dcu.ie (Kevin McGuinness), oconnorn@eeng.dcu.ie (Noel E. O'Connor)

designing effective algorithms is very difficult. The quantity of data involved necessitates an efficient implementation, and the lack of a unique solution complicates evaluation. There are, therefore, usually a multitude of algorithms available when confronted with a particular computer vision problem, and it is often difficult to determine which is the most suitable for a given application.

In such situations it is important to develop a means to characterize and compare the performance of different algorithms, in terms of how satisfactory the solutions found by these algorithms are, and how efficient they are at finding such solutions. Methods for evaluating computer vision algorithms are not only important for application developers, who need to select the most suitable component for their system, but also for researchers. The ability to accurately gauge the performance of a computer vision algorithm gives insight into what makes it good; it allows researchers to develop more sophisticated algorithms by understanding which techniques are effective, and to justify new methods by formal comparison with the state-of-the-art.

A core problem in computer vision is image segmentation, and image segmentation algorithms are critical in many systems. The importance and utility of segmentation has made it the subject of extensive research, resulting in numerous proposed algorithms. Formal evaluation of image segmentation algorithms is therefore important, and recently such evaluation has received increased attention in the literature [2, 3, 4, 5, 6]. Much of this work has, however, been focused primarily on evaluating automatic or semi-automatic[1] segmentation algorithms; comparatively little attention has been dedicated to evaluating interactive segmentation, particularly interactive segmentation of natural images.

Our previous work [1] addressed interactive segmentation evaluation. We focused on supervised evaluation by means of user experiments, and created software, benchmarks, and a complete ground truth dataset for performing this kind of evaluation. We compared four popular interactive segmentation algorithms using this system, and showed their relative performance and characteristics. We validated our proposed measures against perceived accuracy—approximated using questionnaires—and demonstrated their effectiveness.

User experiments are arguably the most effective way to evaluate any

[1] By *semi-automatic* we refer to algorithms that require some level of user input, but do not allow iterative refinement.

interactive system, interactive segmentation being no exception. Such experiments are, however, often prohibitively time-consuming to run, especially if many variations of an algorithm or system need testing. This paper aims to develop a method for supervised evaluation of interactive segmentation algorithms that eliminates the need for user experiments.

Automating the evaluation of interactive segmentation requires replacing the human operator with an algorithmic process designed to emulate the behavior of an operator as closely as possible. To achieve this, we propose driving the segmentation by automatically generating the user interactions from the current segmentation error and the ground truth data. This paper explores four different strategies for generating these interactions. The first of these strategies is deterministic, needing to be run only once to obtain a coarse evaluation. The remaining three strategies are probabilistic: they aim to more realistically approximate a real user, and may, therefore, produce different interactions given the same inputs. We evaluate the same four algorithms from our previous work using each of the automation strategies, and compare the results with the user experiments.

Automated simulation of user interactions is not new; it has been applied successfully to systems evaluation in several other fields. Riedl and Amant [7] developed a system that autonomously carries out a limited form exploratory navigation of graphical user interfaces, and used it to evaluate the usability of these systems. Foley and Smeaton [8] used user simulation to evaluate a synchronous collaborative information retrieval system, using logged data from previous experiments to simulate interaction with their collaborative system. Jung et al. [9] recently used user simulation for automating the evaluation of spoken dialog systems.

Automation has also been used to evaluate medical image segmentation algorithms. Mao et al. [10] proposed a system for automating performance evaluation of semi-automatic segmentation algorithms on B-mode ultrasound images of carotid arteries. Their technique assumes an algorithm that accepts a single seed point as input and generates an object-background delineation based on this point. Their automation algorithm systematically selects different seed points in a matrix covering the relevant region of the carotid image, thereby allowing evaluation of how sensitive the algorithm is to seed point placement. More recently, Moschidis and Graham [11, 12] have used automated seed selection for evaluating interactive segmentation of 3D volumes in medical images, and used their technique to assess several algorithms. Unlike Mao et al.'s technique, Moschadis and Graham's method supports

multi-seed segmentation algorithms, and the authors propose two different strategies for seed selection: random seed selection, and the selection of seeds at a fixed distance to the boundary. Their technique can also be used on interactive algorithms: it measures accuracy as new seeds are added, rather than assuming a single is added prior to segmentation. Both Mao et al. and Moschidis and Graham's techniques focus on medical imaging; our primary focus here will be on interactive segmentation of natural images.

The primary contributions of this paper are as follows: first, we propose a general system for evaluating interactive segmentation algorithms that is not dependent on user experiments. Second, we investigate four strategies that may be used at each step in the automation to generate new interactions from the current segmentation error. We further examine the effectiveness of each of these strategies by correlating accuracy measured during the automated evaluation with accuracy measured during the user experiments. We show the results of the automated experiments to be very similar to those of the user experiments, demonstrating that it is indeed feasible to usefully automate interactive segmentation evaluation. Finally, we provide software and tools for performing an automated evaluation, and discuss methods of aggregating and interpreting the results.

The remainder of this paper is organized as follows. Section 2 briefly reviews the relevant aspects of our previous work: the evaluation objectives, measures, ground truth dataset, and the four segmentation algorithms evaluated. Section 3 discusses the objectives of automating the evaluation. We outline some general considerations, then develop four strategies for automating the user interactions, beginning with the simplest strategy and iteratively developing more complex ones. Section 4 discusses an experiment designed to evaluate four interactive segmentation algorithms using our new automated system, and outlines the software and tools developed for this experiment. Section 5 analyses the results of this experiment and compares them with the previously conducted user experiments. Section 6 presents our conclusions and makes some recommendations based on our findings.

2. Background

The objective of our previous work was to create a complete framework, including benchmarks, ground truth, software, and tools, for evaluating interactive segmentation algorithms by means of user experiments, and to then evaluate four popular interactive segmentation algorithms using this

infrastructure. This paper builds directly upon this framework: our goal here is to develop techniques for automating the evaluation, thus reducing the need for user experiments, and to explore the validity of such automation techniques by comparing them with user evaluation. To facilitate a direct comparison of the automation strategies with the user experiments, we evaluate the same segmentation algorithms, against the same ground truth, and using the same benchmarks that we previously developed.

The remainder of this section gives a condensed overview of the previously developed infrastructure. The software and ground truth dataset that we describe are available from our website[2]. The interested reader is referred to [1], which describes the user experiment based evaluation in detail, including the evaluation measures, the experiment setup, and the evaluated algorithms.

2.1. Algorithms

As noted in [3], the application domain of interest should be identified from the outset when performing image segmentation evaluation, to ensure a fair and consistent evaluation. We focus on evaluating interactive segmentation techniques appropriate for object extraction from photographs and natural scenes. Based on this, we chose four well-known algorithms suited to this task for evaluation:

1. **BPT**: Interactive segmentation using Binary Partition Trees [13];
2. **IGC**: Interactive Graph Cuts [14];
3. **SRG**: Seeded Region Growing [15];
4. **SIOX**: Simple Interactive Object Extraction [16].

We selected these algorithms so as to provide good coverage of the underlying algorithmic approaches used by most methods in the literature designed for object extraction from natural scenes. We selected only algorithms whose input can be modeled by pictorial input on an image grid [17], allowing transparent integration into our scribble-driven segmentation tool (Figure 1). We did not consider boundary-based algorithms (e.g. [18, 19]), as these require a different interaction model, and tend to be better suited to medical image segmentation applications.

[2] http://kspace.cdvp.dcu.ie/public/interactive-segmentation

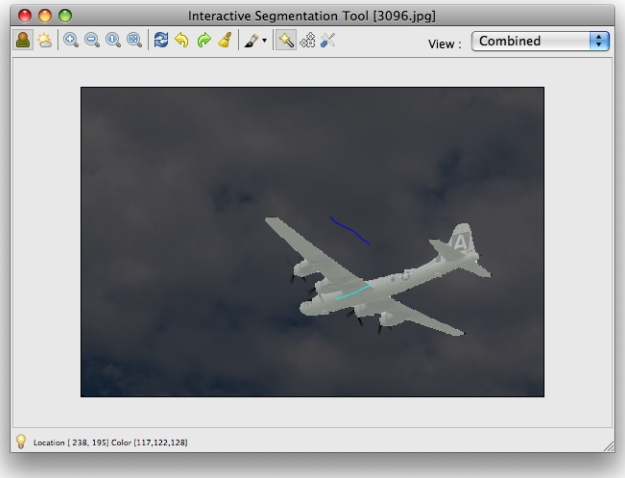

Figure 1: Screenshot of the interactive segmentation tool

2.2. Evaluation

To obtain a comprehensive evaluation of an interactive segmentation algorithm, three criteria should be considered [17, 3]:

- **Accuracy:** the degree to which the delineation of the object corresponds to the truth;
- **Efficiency:** the amount of time or effort required to perform the segmentation; and
- **Repeatability:**[3] the extent to which the same segmentation would be produced over different sessions when the user has the same intention.

To measure accuracy and efficiency, it must be possible to measure how well a segmentation matches the true object over time. For a supervised evaluation, two resources are necessary: a ground truth, which represents the required object as closely as possible, and a set of benchmarks to determine how well a particular segmentation matches the ground truth.

[3]Udupa et al. suggest the same three criteria, referring to repeatability as *precision* in their work [3]

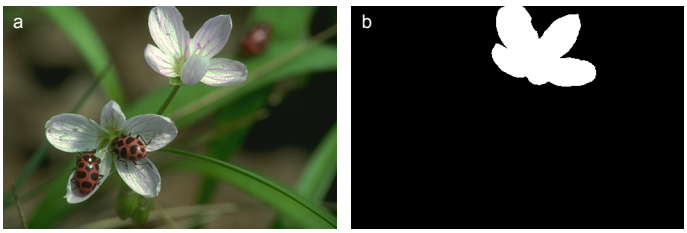

Figure 2: Example task image and the expected object. The task description is *"Extract the white flower in the upper part of the image. Do not include the stem."*

For the user experiments, we measured repeatability implicitly: by requiring multiple segmentations by different users of the same image using the same segmentation algorithm. We also consider repeatability when developing the automation strategies in Section 4.

2.2.1. Ground truth dataset

We compiled the dataset for the experiment by selecting 100 objects from 96 images in the Berkeley segmentation dataset [20]. We only selected images that contained objects that could be concisely and unambiguously be described to the experiment participants. Each object was segmented by hand using a graphics tablet to create the ground truth, then annotated with a task description, producing a database of 100 tasks for the experiment. Figure 2 shows a typical task image, task description, and corresponding ground truth object.

2.2.2. Evaluation benchmarks

We use two benchmarks to measure the accuracy of a segmentation against the ground truth object: the boundary accuracy B_A, and the object accuracy O_A. The boundary accuracy benchmark is designed to measure how well the boundary of the segmented object corresponds to the ground truth. The object accuracy benchmark measures how well the entire region matches the ground truth.

Let $\mathbf{v} \in \mathbb{Z}^2$ be any pixel inside the ground truth object, and $G_{\mathcal{O}} = \{\mathbf{v}\}$ be the set of all of these pixels. Similarly, define $M_{\mathcal{O}}$ to be the set of all pixels in the machine-segmented object. $G_{\mathcal{B}}$ and $M_{\mathcal{B}}$ denote the complements of these sets. Let $\mathcal{N}_{\mathbf{x}}$ be the standard set of eight-neighbors of any $\mathbf{x} \in \mathbb{Z}^2$. The internal border pixels for the ground-truth object are defined as the set B_G,

and for the machine-segmentation, the set B_M, as follows:

$$B_G = \{\mathbf{x} : \mathbf{x} \in G_{\mathcal{O}} \wedge \mathcal{N}_{\mathbf{x}} \cap G_{\mathcal{B}} \neq \emptyset\} \tag{1}$$

$$B_M = \{\mathbf{x} : \mathbf{x} \in M_{\mathcal{O}} \wedge \mathcal{N}_{\mathbf{x}} \cap M_{\mathcal{B}} \neq \emptyset\} \tag{2}$$

Our object accuracy measure is simply the Jaccard index used to measure the overlap between the machine segmented object and the ground truth object:

$$A_O = \frac{|G_{\mathcal{O}} \cap M_{\mathcal{O}}|}{|G_{\mathcal{O}} \cup M_{\mathcal{O}}|} \tag{3}$$

This index is sometimes written as the ratio between true positives and the sum of true positives, false positives, and false negatives: $A_O = TP/(TP + FP + FN)$, and has been used previously by several authors for comparing segmentations (e.g., [4, 5, 3, 21]).

To compute boundary accuracy, we could adopt a similar strategy by substituting the sets B_G and B_M in place of $G_{\mathcal{O}}$ and $M_{\mathcal{O}}$ in Equation (3):

$$A_B = \frac{|B_G \cap B_M|}{|B_G \cup B_M|} \tag{4}$$

Unfortunately, however, this benchmark is extremely sensitive to small errors near the boundary of the object. Since the precise boundary of an object on a raster grid is inherently ambiguous (see Ref. [1]), and since humans are generally less sensitive to small errors near the object boundary than, say, large holes in the object, the values given by Equation (4) will almost always be excessively low.

To adapt the Jaccard index so that it is more appropriate for our purposes, we introduce some tolerance to error near the border pixels. We accomplish this by transforming the sets B_M and B_G to their fuzzy analogs $\tilde{B}_G$ and $\tilde{B}_M$ as follows:

$$\tilde{B}_G(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|^2}{2\sigma^2}\right) \tag{5}$$

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{y} \in B_G} \|\mathbf{x} - \mathbf{y}\| \tag{6}$$

where the fuzzy set $\tilde{B}_M$ is similarly defined, and σ is a parameter that quantifies the degree of uncertainty in the boundary pixels. We then define the boundary accuracy using a fuzzy version of the Jaccard index:

$$A_B = \frac{\sum_{\mathbf{x}} \min(\tilde{B}_G(\mathbf{x}), \tilde{B}_M(\mathbf{x}))}{\sum_{\mathbf{x}} \max(\tilde{B}_G(\mathbf{x}), \tilde{B}_M(\mathbf{x}))} \tag{7}$$

which can be computed in linear time using a fast distance transform algorithm (for example, the algorithm by Fabbri et al. [22]) to evaluate Equation (6).

Both the object accuracy and boundary accuracy benchmarks are symmetric, and produce values in the interval $[0, 1]$, where larger values imply more accurate segmentation.

2.3. User experiments

A total of 20 volunteers participated in the user experiments. Each participant was asked to extract 25 objects with each of the four segmentation algorithms being evaluated—100 objects in total. A time limit of two minutes was imposed for each extraction.

To facilitate the experiments and host the segmentation algorithms, we developed the interactive segmentation tool (see Figure 1). The application was developed as a general purpose interactive segmentation tool, but includes an "experiment mode" for running timed user experiments. In this mode the segmentation algorithm is locked by the tool, and users are presented with a series of object extraction tasks that must be completed within the time limit. As the user marks the image with the mouse, the segmentation is updated and the accuracy measures previously discussed are computed and stored along with the elapsed time.

The result is, for each user and object, a series of accuracy measurements, giving a profile of how accuracy varies over time. We discuss the analysis of this time-series data in Section 5, and compare it with the time-series data gathered from the automated experiments.

3. Automation

When evaluating interactive segmentation by means of user experiments, the participant is required to provide seed pixels for the object and background regions by marking the image with the mouse. The segmentation algorithm builds an initial segmentation using these seed pixels as priors, and provides feedback to the participant. The participant may then iteratively refine this segmentation by marking additional pixels until either a satisfactory segmentation is obtained, or the allotted time expires. This kind of experiment, while invaluable for establishing the usability of an algorithm, is often prohibitively difficult and time-consuming, especially considering that it needs to be repeated each time a new algorithm is evaluated. The idea behind automating the evaluation is to devise an algorithmic process that can

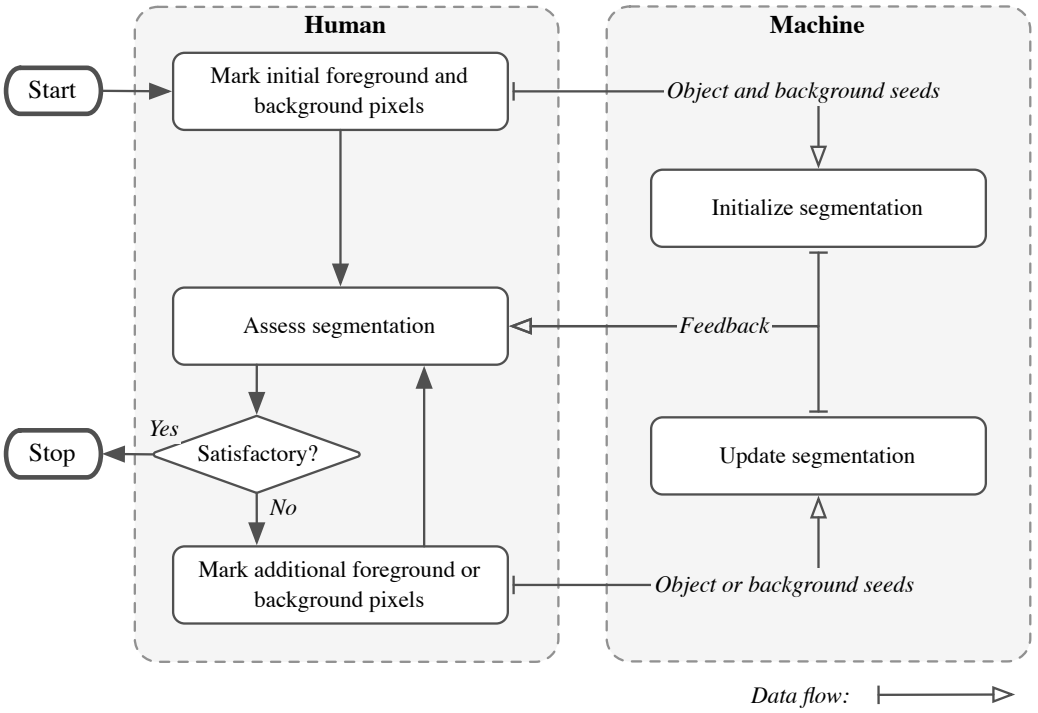

Figure 3: Flow chart of user activity when performing a segmentation task

simulate, in some reasonable way, all the actions that are usually performed by the human operator, thereby eliminating the need for user experiments.

When first considering this problem, one might initially reason that since we have already performed the user experiments, we can simply record each interaction performed by the participants and automate the procedure by replaying these interactions. Inspection reveals this reasoning to be flawed: although the first set of foreground and background seeds supplied by the user could indeed be used by an automation algorithm, all subsequent interactions are reactive. That is, at each step, the user is attempting to correct the current segmentation error. This error depends on the algorithm being evaluated, and on the interactions from previous steps.

Automating the evaluation requires us to identify each decision that is made by the user, so that it can be replaced by an automated action. Figure 3 depicts a high-level view of the flow of activity for a user during a segmentation task. From this we can identify the following user responsibilities:

1. Identify the object to be extracted from the task description;
2. Select initial foreground and background seed pixels;
3. Correct errors in the segmentation by selecting additional foreground or background seed pixels; and
4. Decide after each interaction if the segmentation is satisfactory.

Identifying the object to be extracted from the task description is, undoubtably, the most difficult of these tasks to automate. It is, however, possible to bypass this step: we are performing a supervised evaluation, so the object to be extracted is coded exactly in the ground truth. Step 2 can be accomplished by selecting, in some way, the initial seed pixels from the object and background regions in the ground truth. After an initial segmentation has been found, we can find the mislabeled pixels by comparing this initial segmentation against the ground truth. This gives us the means to automate Step 3: by selecting the additional object or background seed pixels from the set of mislabeled pixels.

The final step is to decide if the segmentation is satisfactory, and if so, terminate the process. A straightforward criterion is to declare a segmentation to be satisfactory if and only if it exactly matches the ground truth. In our experiments, however, we observed that this strategy often results in a great deal of time toward the end of the segmentation process being spent correcting insignificant errors along the boundary of the object. Since, as noted earlier, the true boundary of an object is inherently ill-defined, and since human operators do not normally notice such slight error, much less spend time correcting it, we recommend terminating the segmentation if the only remaining error pixels lie on the inner or outer boundary of the object.

We have not yet specified how to select the seed pixels in the initialization stage or correction stage. Furthermore, we have made some important assumptions in the above discussion that warrant further consideration. We address both these issues shortly. First, let us outline the general automation algorithm by drawing on the above discussion. We use the following notation for important sets of pixels in the machine segmentation and ground truth:

- $G_{\mathcal{O}}$: the set of all pixels inside the ground truth object;
- $G_{\mathcal{B}}$: the set of all pixels outside the ground truth object;
- $M_{\mathcal{O}}$: the set of all pixels inside the machine segmented object;
- $M_{\mathcal{B}}$: the set of all pixels outside the machine segmented object;

- B_G: the set of internal border pixels for the ground truth object;
- B_M: the set of internal border pixels for the ground truth background region.

Note that the set of internal border pixels for the background region equals the set of external border pixels for the object region, and similarly, the set of internal border pixels for the object region equals the set of external border pixels for the background region.

In addition to the above definitions, we denote the initialization seeds, the update seeds, and the current segmentation error as follows:

- $I_{\mathcal{O}}$ and $I_{\mathcal{B}}$ are the sets of object and background seeds used by the automation algorithm to initialize the segmentation;
- $U_{\mathcal{O}}$ and $U_{\mathcal{B}}$ are the sets of object and background seeds used to update (refine) the segmentation;
- $E_{\mathcal{O}}$ and $E_{\mathcal{B}}$ are the sets of object and background pixels that have been misclassified by the segmentation algorithm (the error).

Using this notation, our proposed general automation algorithm proceeds as follows:

1. *Initialize*: Select the initial object seed points $I_{\mathcal{O}}$, and the initial background seed points $I_{\mathcal{B}}$, such that $I_{\mathcal{O}} \subseteq G_{\mathcal{O}}$ and $I_{\mathcal{B}} \subseteq G_{\mathcal{B}}$. Mark these points as object and background and update the segmentation.
2. *Compute error*: Determine the set of misclassified object pixels $E_{\mathcal{O}}$, and the set of misclassified background pixels $E_{\mathcal{B}}$, as:

$$E_{\mathcal{O}} = M_{\mathcal{B}} \cap G_{\mathcal{O}} \tag{8}$$

$$E_{\mathcal{B}} = M_{\mathcal{O}} \cap G_{\mathcal{B}} \tag{9}$$

3. *Check for termination*: If the sets of misclassified pixels above contain only pixels from the object's internal or external border, terminate the algorithm. More formally, the algorithm terminates if:

$$E_{\mathcal{O}} \subseteq B_G \wedge E_{\mathcal{B}} \subseteq B_M$$

Note that the above holds when both $E_{\mathcal{O}}$ and $E_{\mathcal{B}}$ are the empty set.

4. *Correct*: Update the segmentation by selecting either additional object seeds $U_{\mathcal{O}} \subseteq E_{\mathcal{O}}$, or additional background seeds $U_{\mathcal{B}} \subseteq E_{\mathcal{B}}$, and return to Step 2.

There are two important, and related, implications of the above algorithm that need to be addressed. First, if the interactive segmentation algorithm being evaluated is not "well-behaved" the algorithm may never halt. By well-behaved, we mean that if we mark a pixel as object, the algorithm will always classify it as object, and if we mark it as background, the algorithm will always classify it as background. We believe that it is justified to assume such behavior, as we explain shortly. It is straightforward to modify an existing segmentation algorithm to conform to this behavior by post-processing the output.

The second implication is that since the automation algorithm only ever chooses object seeds that are inside the ground truth object, and background seeds that are outside the ground truth object, it is impossible for our automation algorithm to ever make a mistake—i.e., to incorrectly mark an object pixel as background or vice versa. Our experiments have shown that users, in general, are not so diligent.

However, we believe that both these assumptions are sensible design decisions. In our user experiments, participants universally agreed that they preferred algorithms with predictable behavior [1]. So although it is possible for an interactive segmentation algorithm to compensate for inaccurate interactions, we do not necessarily believe that they should: such compensation is a direct violation of the user's instructions, however imprecise. This kind of behavior may be helpful in a few cases, but more often than not, is an endless source of frustration to users [23]. The desire for predictable behavior justifies the requirement that algorithms must be well-behaved; well-behaved algorithms are, by definition, incapable of compensating for user errors.

What remains is to decide a suitable way of selecting the initialization seeds $I_{\mathcal{O}}$ and $I_{\mathcal{B}}$, and the update seeds $U_{\mathcal{O}}$ and $U_{\mathcal{B}}$. We now explore four strategies for their selection.

3.1. Strategy 1

We begin by investigating a very simple deterministic strategy for choosing the initialization and update seed pixels. We do so to set up a baseline approach against which we can compare more sophisticated strategies, which attempt to more closely approximate real user interactions.

The basis of this strategy is the observation that users tend to begin extracting objects by marking as foreground some pixels in the middle of the object, and marking as background some pixels well outside the object. They then proceed to refine the initial segmentation by marking pixels that lie inside large areas of misclassified pixels. To emulate this behavior, strategy 1 initializes the segmentation by selecting pixels that are near the center of the ground truth object as object seeds, and selecting pixels that are near the center of the background region in the ground truth as background seeds. Similarly, to update the segmentation, the strategy chooses those pixels that are farthest from the correctly classified pixels as the update seeds.

Let $D(\mathbf{x}, R)$ be the minimum distance from a pixel $\mathbf{x}$ to any pixel in the set R:

$$D(\mathbf{x}, R) = \min_{\mathbf{y} \in R} \|\mathbf{x} - \mathbf{y}\| \tag{10}$$

and let the $Z(Q, R)$ be the set of all points in Q that are are maximally distant to their nearest points in R:

$$Z(Q, R) = \arg\max_{\mathbf{x} \in Q} D(\mathbf{x}, R) \tag{11}$$

We choose the initial seed points $I_{\mathcal{O}}$ and $I_{\mathcal{B}}$ as:

$$I_{\mathcal{O}} = \{\mathbf{x} : \mathbf{x} \in \mathrm{Br}(\mathbf{y}, G_{\mathcal{B}}), \mathbf{y} \in Z(G_{\mathcal{O}}, G_{\mathcal{O}}^C)\} \tag{12}$$

$$I_{\mathcal{B}} = \{\mathbf{x} : \mathbf{x} \in \mathrm{Br}(\mathbf{y}, G_{\mathcal{O}}), \mathbf{y} \in Z(G_{\mathcal{B}}, G_{\mathcal{B}}^C)\} \tag{13}$$

where G^C denotes the complement of the set G, and $\mathrm{Br}(\mathbf{y}, R)$ is a brush function that returns all pixels within a fixed radius of $\mathbf{y}$:

$$\mathrm{Br}(\mathbf{y}, R) = \{\mathbf{x} : \|\mathbf{x} - \mathbf{y}\| \leq r(\mathbf{y}, R)\} \tag{14}$$

The brush radius is given by the $r(\mathbf{y}, R)$ function. It is chosen proportional to the minimum distance from the center seed points $\mathbf{y}$ to the object boundary, within the constraints of the interactive segmentation tool. Note that all such points $\mathbf{y} \in Z(G_{\mathcal{O}}, G_{\mathcal{O}}^C)$ are equidistant from the background, and similarly all points $\mathbf{y} \in Z(G_{\mathcal{B}}, G_{\mathcal{B}}^C)$ are equidistance from the object. The interactive segmentation tool has a maximum brush radius of 20 pixels, so we define our brush radius function as:

$$r(\mathbf{y}, R) = \min\left(\frac{1}{2} D(\mathbf{y}, R), 20\right) \tag{15}$$

Our reasoning here is simple: users tend to use larger brush sizes to correct larger errors, and smaller brush sizes to correct minor details; they cannot set the brush to a size larger than 20 because our tool does not support it.

To update the segmentation we follow a similar strategy, this time taking the update seeds $U_{\mathcal{O}}$ and $U_{\mathcal{B}}$ from the sets of misclassified object and background pixels $E_{\mathcal{O}}$ and $E_{\mathcal{B}}$:

$$U_{\mathcal{O}} = \{\mathbf{x} : \mathbf{x} \in \mathrm{Br}(\mathbf{y}, E_{\mathcal{O}}^C), \mathbf{y} \in Z(E_{\mathcal{O}}, E_{\mathcal{O}}^C)\} \tag{16}$$

$$U_{\mathcal{B}} = \{\mathbf{x} : \mathbf{x} \in \mathrm{Br}(\mathbf{y}, E_{\mathcal{B}}^C), \mathbf{y} \in Z(E_{\mathcal{B}}, E_{\mathcal{B}}^C)\} \tag{17}$$

In the interactive segmentation tool, the segmentation is updated after each interaction; it is impossible for users to simultaneously mark object and background pixels in a single interaction. We therefore update the segmentation using only one of the above sets: $U_{\mathcal{O}}$ or $U_{\mathcal{B}}$. We select the set of update pixels U to be the set that has larger minimum distance between its center and the external border of the set of misclassified pixels it is drawn from:

$$U = \begin{cases} U_{\mathcal{O}} & \max_{\mathbf{y} \in U_{\mathcal{O}}} D(\mathbf{y}, E_{\mathcal{O}}^C) > \\ & \qquad \max_{\mathbf{z} \in U_{\mathcal{B}}} D(\mathbf{z}, E_{\mathcal{B}}^C) \\ U_{\mathcal{B}} & \text{otherwise} \end{cases} \tag{18}$$

Figure 4 shows the first few steps of this strategy when evaluating the IGC algorithm. As can be seen, the strategy begins by placing one or more circular blobs of seed pixels in the center of the object to be extracted, and one or more blobs of seed pixels in the center of the background region. Each update then corrects errors in the segmentation by placing seed pixels in the center of the largest regions of misclassified pixels.

The strategy can be efficiently implemented using a fast 2D Euclidean distance transform algorithm. Our implementation uses the linear time algorithm proposed by Meijster et al. [24], demonstrated in [22] to be one of the fastest 2D Euclidean distance transform algorithms available. This allows us to evaluate Eq. (10) for all values of $\mathbf{x}$ in a region in linear time, and therefore initialization and each update also run in linear time.

This strategy has some advantages; it is relatively quick to compute, and since it is deterministic, it will produce the same sets of seed points given the same segmentation algorithm and ground truth each time it is run (provided, of course, that the segmentation algorithm is also deterministic). It therefore needs to be run only once for each algorithm being evaluated.

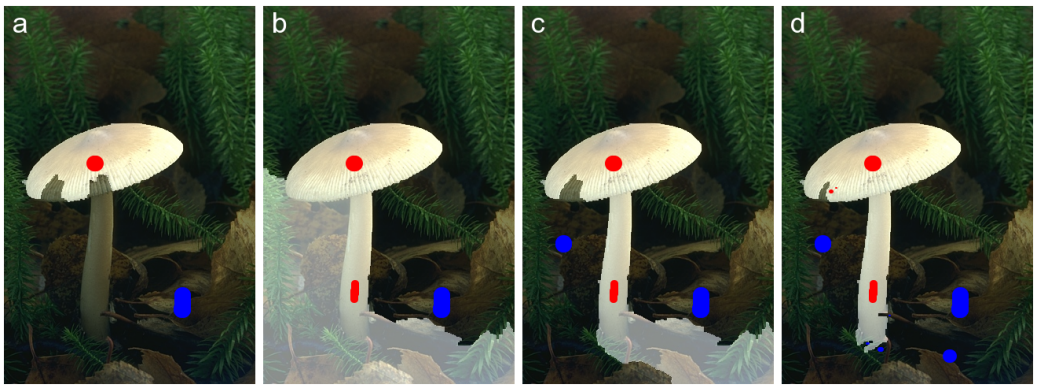

Figure 4: An illustration of the first few the steps of automation strategy 1. The segmentation algorithm being evaluated is IGC [14]. Each update adds one or more circular blobs of seed pixels to correct errors in either the object or background regions. (a) is the initialization step; (b) and (c) show the effect of the first two updates; (d) shows the image after nine updates.

There are, however, two distinct disadvantages of this approach. First, since it is deterministic, it does not evaluate repeatability. The strategy gives no indication of how robust the algorithm being evaluated is to small variations in markup: it always produces the same markup given the same algorithm and input. This behavior is in direct contrast to real users, who are unlikely to produce the same sets of markings when extracting a given object. Second, the strategy produces pixel blobs instead of lines and curves, similar to a user repeatedly clicking the mouse on the misclassified regions each time they wish to correct the segmentation. Although this is a perfectly valid way of extracting objects, observation indicates that users prefer to draw lines or curves.

We address both of these issues as we investigate more complex automation strategies. Strategy 1 is a useful baseline against which we can compare more sophisticated approaches.

3.2. Strategy 2

Recall that choosing our initialization and update seeds requires selecting a set of seed pixels S from a set of candidate pixels C. For strategy 1, we selected the set of pixels from C that were maximally distant from their nearest neighbors in C^c, then expanded our selection using a brush function. We do the same for strategy 2, except this time we select pixels from C

non-deterministically. We propose selecting pixels from C such that the probability of selecting $\mathbf{x} \in C$ is proportional to the spatial distance from $\mathbf{x}$ to C^c. This way we are more likely to select pixels that are nearer to the center of the object on initialization, and nearer to the center of groups of misclassified pixels on update.

To achieve this, we need to define a discrete probability distribution for selecting pixels from C based on their spatial distances to their nearest neighbors in C^c. Such a distribution could be defined in various ways; for simplicity, we opted to design our distribution using sum-normalized distances. This gives the following discrete probability mass function:

$$\Pr[X = \mathbf{x}] = \frac{D(\mathbf{x}, C^c)}{\sum_{\mathbf{y} \in C} D(\mathbf{y}, C^c)} \tag{19}$$

The above mass function can be used to select seeds with the desired probabilities using the inversion method [25] as follows:

1. First, compute a discrete estimation of the cumulative distribution function for Eq. (19) as:

$$\mathrm{F}(\mathbf{x}) = \sum_{\mathbf{y} \in C \leq \mathbf{x}} \Pr[X = \mathbf{y}] \tag{20}$$

   The above requires defining a partial order over C. Any such order will do; we used the row-major (lexicographical) order of the vectors in C for convenience in our implementation.
2. Generate a random number $u \in (0, 1)$ from the standard uniform distribution.
3. Find the smallest value $\mathbf{x}_i$ such that $\mathrm{F}(\mathbf{x}_i) \geq u$. Binary search on F can be used to find $\mathbf{x}_i$ in $O(\log n)$ time.

The brush function is then applied as before to expand the selection. Since we now have a non-deterministic method of selecting the initialization and update pixels, we can evaluate repeatability by using multiple runs of the method, simulating multiple users.

3.3. Strategy 3

We noted previously that users tend to draw lines and curves to mark up objects, rather than simply pointing and clicking. To make the evaluation

Figure 5: Extracting a non-convex object with straight line segments: (a) is a synthetic, non-convex object, designed so that it has two distinct maxima that cannot be joined using a straight line. (b) is the distance transform of (a). (c) is the maxima of the distance transform overlaid on the object. (d) shows a dashed line segment joining the maxima points. (e) is the ideal solution we would like to approximate.

more realistic, we would prefer if our automation strategy provided similar interactions.

Our goal here is to select a sequence of seed points $P = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_n)$ from our candidate points C, such that each seed point is a neighbor of a previously selected point:

$$\mathbf{x}_i \in P \implies \mathbf{x}_i \in C \tag{21}$$

$$\mathbf{x}_i \in P \wedge i > 1 \implies \mathbf{x}_{i-1} \in \mathcal{N}(\mathbf{x}_i) \tag{22}$$

where $\mathcal{N}(\mathbf{x}_i)$ is the set of 8-neighbors of $\mathbf{x}_i$.

There are typically many sequences P which satisfy the above predicates. Most of these, however, are not usually what we would (intuitively) consider realistic for a user to draw when marking up objects. Our experiments suggest that users tend to draw smooth, simple curves; we would ideally like to devise a strategy that emulates this behavior.

The path $P = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_n)$, has a start point $\mathbf{x}_1$, and an end point $\mathbf{x}_n$. A logical way to create such a path is to select the start and end point using the strategy outlined in Section 3.2, and find a sequence of adjacent pixels joining the start and end points. The simplest and shortest path of pixels joining $\mathbf{x}_1$ and $\mathbf{x}_n$ is a straight line rasterized on the pixel grid. Unfortunately, since the required objects are not always convex, a straight line is not guaranteed to fall entirely within the region we are marking. Figure 5 illustrates this issue. Figure 5(a) shows a synthetic non-convex object; 5(b) illustrates its distance transform. Figure 5(c) shows the maxima of the distance transform. These are also the two most likely points to be selected as endpoints by the method described in Section 3.2. Figure 5(d) shows that the line joining these two points lies outside the object. The ideal solution we would like to approximate is shown in Figure 5(e).

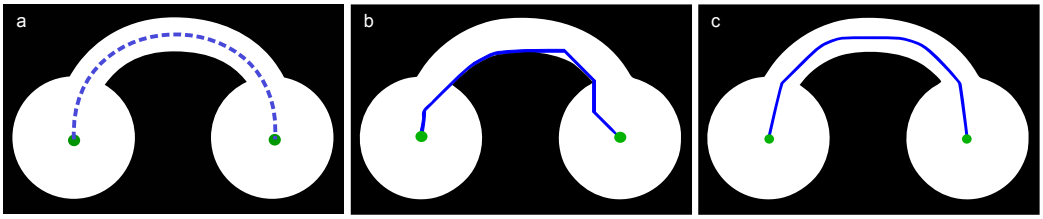

Figure 6: Paths between the maxima points of Figure 5. (a) shows the ideal solution; (b) shows the path found automation strategy 3; (c) shows the path found by automation strategy 4.

Our first approach for approximating the kind of lines and curves generated by human operators uses the shortest spatial path on the image grid between points $\mathbf{x}_1$ and $\mathbf{x}_n$ that lies completely inside the candidate region. To compute this path, we construct a graph $G = (V, E)$ from the candidate pixels C, such that each pixel $\mathbf{x} \in C$ is a vertex in the graph, and each vertex is connected to all of its eight neighbors also in C. That is:

$$G = (V, E) \tag{23}$$
$$V = \{\mathbf{x} : \mathbf{x} \in C\} \tag{24}$$
$$E = \{(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in C, \mathbf{y} \in \mathcal{N}_{\mathbf{x}} \cap C\} \tag{25}$$

Each edge in E is then given a weight equal to the spatial distance between the vertices it joins. Since we are operating on an 8-connected graph, these weights are equal to 1 for horizontal and vertical edges, and $\sqrt{2}$ for diagonal edges. Having constructed G, we can now use Dijkstra's algorithm [26] to find the shortest path between $\mathbf{x}_1$ and $\mathbf{x}_n$. Figure 6(b) shows the path found using this approach on the example in Figure 5.

The algorithm, as specified thus far, will fail if $\mathbf{x}_1$ and $\mathbf{x}_n$ lie in regions that are spatially disjoint. We must, therefore, avoid this situation and only choose our endpoints so that there always exists a path between them. The final strategy 3 algorithm is as follows:

1. Construct the candidate graph G.
2. Select an initial point $\mathbf{x}_1$ using the non-deterministic strategy in Section 3.2.
3. Determine which other vertices in G are connected to $\mathbf{x}_1$ using Dijkstra's algorithm. This also gives us the shortest path from $\mathbf{x}_1$ to every other connected vertex in G.

4. Remove $\mathbf{x}_1$ and all vertices not connected to $\mathbf{x}_1$ in G from the candidate pixels C.
5. Select a second point $\mathbf{x}_n$, again using the non-deterministic strategy in Section 3.2.
6. Set P equal to the shortest path between $\mathbf{x}_1$ and $\mathbf{x}_n$ as found in step 3.

The set of seed pixels P' is then chosen by expanding the path P using a brush function similar to that in Equation (14). In this instance, the brush radius is chosen relative to the minimum distance from any pixel on the path P to one of the non-candidate pixels C^c as follows:

$$P' = Br'(P, C^c) \tag{26}$$

$$Br'(P, R) = \{\mathbf{y} : \|\mathbf{x} - \mathbf{y}\| \leq r'(P, R), \mathbf{x} \in P\} \tag{27}$$

$$r'(P, R) = \min_{\mathbf{x} \in P} r(\mathbf{x}, R) \tag{28}$$

3.4. Strategy 4

The paths found by strategy 3 are the shortest possible. They will, therefore, often yield paths that pass very close to the boundary of the candidate region (see Figure 6(b)). We would prefer to generate paths that stay closer to the center of the region, as in Figure 6(a).

To achieve this, we propose adjusting the weights on the candidate graph G, so that paths that move toward the center of the object are preferred over the shortest possible path. Previously we set the weight for the edge between vertex $\mathbf{x}$ and $\mathbf{y}$ equal to the spatial distance between them:

$$w_{\mathbf{x}\to\mathbf{y}} = \|\mathbf{x} - \mathbf{y}\|$$

Modulating the above by the exponent of the normalized distance from $\mathbf{y}$ to the boundary introduces a preference to move toward the center of the object. Our modified distance function is:

$$w'_{\mathbf{x}\to\mathbf{y}} = \|\mathbf{x} - \mathbf{y}\| \exp\left(\frac{D(\mathbf{y}, C^c)}{\max_{\mathbf{z}\in C} D(\mathbf{z}, C^c)}\right) \tag{29}$$

which, for our previous example, yields the path in Figure 6(c). The path is again expanded using the brush function to form the set of seed pixels. Figure 7 shows a more realistic example of this strategy in action.

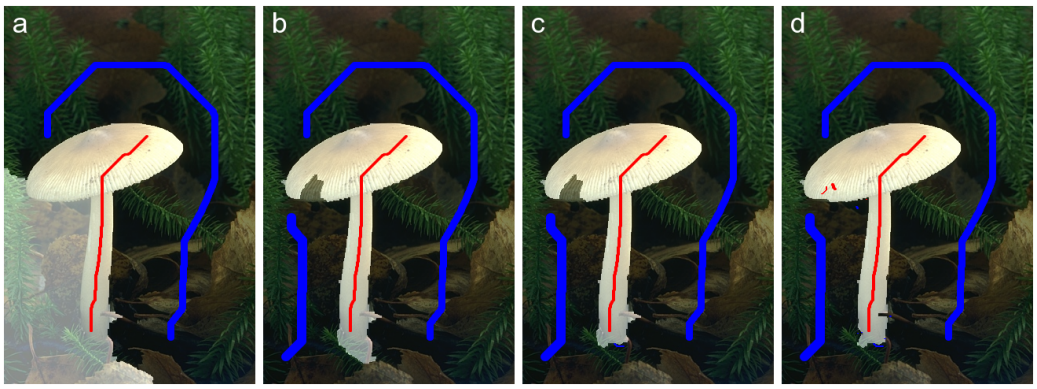

Figure 7: An illustration of the first few the steps of automation strategy 4. The segmentation algorithm being evaluated is IGC [14]. (a) is the initialization step; (b) and (c) show the effect of the first two updates; (d) shows the image after nine updates.

4. Evaluation

To run the automated evaluation we developed the automation tool, shown in Figure 8. The tool allows configuration of all aspects of the evaluation, including: the algorithm being evaluated, the automation strategy, the input and ground truth files, and the evaluation measures to use. When the evaluation is run, the tool processes each input image and corresponding ground truth with the selected automation strategy. After each automation step is taken, the segmentation is updated, and accuracy is computed against the ground truth using the selected accuracy measures. For the non-deterministic automation strategies, the process is repeated the desired number of times.

We set an upper limit of 100 steps for each automation strategy. This limit is imposed not only to ensure that the automation strategies terminate in a reasonable amount of time, it is also important during the analysis of the results, as we show in the next section. To effectively evaluate repeatability, evaluation using the non-deterministic strategies needs to be repeated several times to simulate different users; we used five repetitions for our experiments.

An upper limit of 100 steps may seem quite excessive, given that real users usually provide less than 50 or 60 interactions, and only rarely as many as 100 (see Figure 9). The size of these interactions may vary widely however, from single point clicks, to wide scribbles and curves. Our automation strategies also vary in the amount of input provided per step, and so we deliberately set

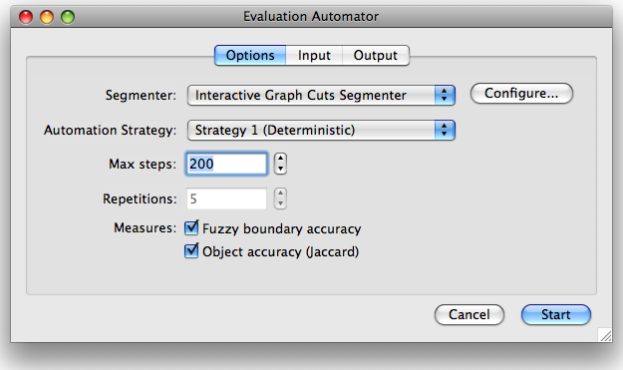

Figure 8: Screenshot of the configuration window for the automated evaluation tool.

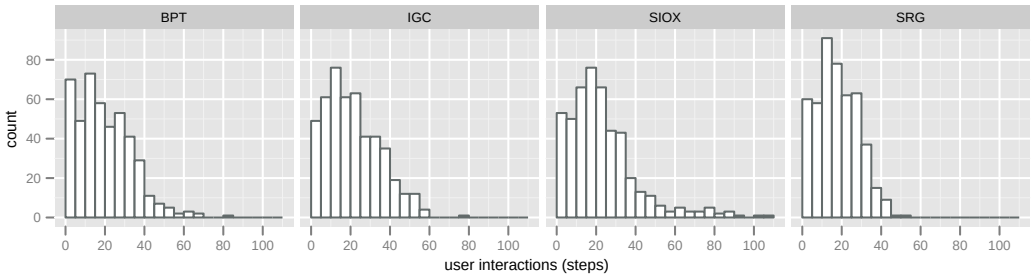

Figure 9: Histogram of the number of interactions per task provided by users during the user evaluation experiments.

this limit high for now to observe the behavior of these strategies after many interactions. Since we measure accuracy over time, we can always truncate the results if they stabilize earlier. We recommend a more realistic upper limit that can be used for practical experiments in Section 6.

The time required to run the automated experiments depends on the automation strategy used, on the algorithm being evaluated, on the number of images in the dataset, and on the number of repetitions when using a non-deterministic automation strategy. Strategy 1 is the least computationally intensive automation strategy, and strategy 4 is the most intensive. To give an idea of the typical time required to run an automated evaluation, Table 1

	Strategy	Algorithm	Reps.	Time
Server	1	IGC	1	17 min
Server	4	IGC	5	2 hr 18 min
Desktop PC	1	IGC	1	46 min
Desktop PC	4	IGC	5	7 hr 13 min

Table 1: Indicative runtimes of the automation strategies. Both strategies were run for a maximum of 100 steps on all 100 objects in the database.

shows the runtimes for two automation tasks on a server[4] and desktop PC.[5]

5. Analysis

The objective of our analysis is twofold. First, since there is a large amount of data generated by the experiments, we need to develop a effective means to reduce and interpret this data. We can then apply this to each evaluation strategy to investigate the characteristics of the evaluated algorithms. Second, we want to compare each of the evaluation strategies, and determine which best approximates full user experiments.

Every ground truth object evaluated results in a time series of object accuracy and boundary accuracy values. Figure 10 shows two such time series from our experiments. Each interactive segmentation algorithm evaluated produces 100 of these time series (one for each ground truth object). This is increased to 500 for the non-deterministic strategies, since the evaluation is run five times. To allow us to more easily interpret these data, we aggregate it in two ways.

The first way is to examine the average accuracy over all images as a function of time. We refer to this as the time-accuracy profile curve. The aim is to determine how, on average, accuracy varies over time for each evaluated algorithm. To compute time-accuracy profile curves, we first need to expand the time series data to the maximum number of steps. This is done by duplicating the final accuracy measured (i.e. the accuracy when the

[4]Dell™ Poweredge™ 1900, 3GHz Intel® Xeon® 5160 CPU, 1333MHz front-side bus, 8GB 667MHz RAM, Linux kernel 2.6.20 x86_64

[5]Dell™ Optiplex™ GX520, 3GHz Intel® Pentium® 4 CPU, 800MHz front-side bus, 1GB 533MHz RAM, Linux kernel 2.6.31 i386

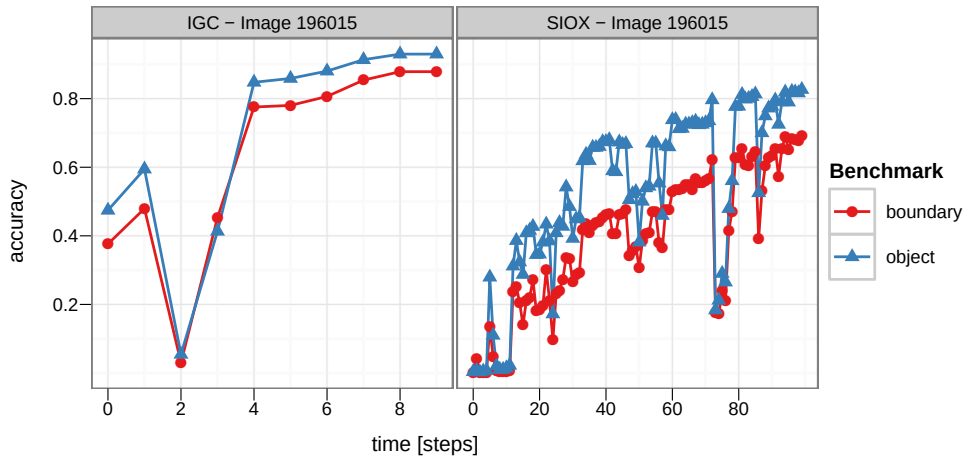

Figure 10: Sample time series data for individual images. The time series were created by evaluating the IGC algorithm (left) and the SIOX algorithm (right) with automation strategy 1 against the same ground truth image.

automation terminates) for each subsequent step up to the maximum. Once the time series data have been expanded to equal length, they are sampled at regular intervals using a fixed sampling window. These samples are then averaged to produce the time-accuracy profiles.

The second way we aggregate the data is by computing scalar features of the individual time series, and averaging these features across the collected data. We use two scalar features. The first is *final accuracy*, defined as the accuracy measured when the automation terminates; it gives an indication of the accuracy that can be achieved in a reasonable amount of time using a given segmentation algorithm. In Figure 10 this is simply the last value on the right hand side of the curves.

The second feature we compute is termed *integrated accuracy*. Consider again the time series in Figure 10. The time series on the left clearly indicates better performance than the time series on the right: it achieves a higher accuracy in less steps. Observe that algorithms that are performing well tend to produce curves that increase quickly at first and then gradually level-off; algorithms that are performing poorly tend to have more gradual, "choppy" curves. One useful way to reduce these curves to a scalar value is to examine the area under the curves.

If we expand each time series data so that they are of equal length (in

the same way as when computing the time-accuracy profile curves) then the area under the time series curve is a good indication of the overall performance of an algorithm. A large area relative to another indicates that one segmentation algorithm maintains a higher average accuracy than the other, usually as a result of achieving an accurate segmentation faster and subsequently maintaining, or slowly improving upon, this segmentation. Furthermore, this area is bounded by the area of the unit height rectangle that is the same width as the expanded time series, and can therefore be easily normalized to the range (0..1).

It is possible to approximate the area under a time series by summation when the data points for the time series are unit spaced. The data points collected during an automated evaluation experiment *are* unit spaced: accuracy is measured after each step in the automation process. The area beneath this time series can, therefore, be approximated by the sum of all data points in the series.

We also need to calculate the integrated accuracy feature for the user experiments so that we can determine how well they correlate with the automated experiments. This again necessitates determining the area beneath the time series curves. However, the data points from the user experiments are non-unit spaced: accuracy is measured at different points in real time, i.e., every time the user refines the segmentation by marking additional pixels as object or background. To approximate the area under these time series using summation, linear resampling can be used to coerce the series to one that *is* unit spaced. It is also possible to approximate the area under non-unit spaced points using the trapezoid rule for numeric integration [27].

Denoting $\mathcal{A}(a)$ the area under the expanded time series $a = (a_1, a_2, \ldots, a_k)$, as computed using one of the above procedures, we define the integrated accuracy feature as this area $\mathcal{A}(a)$ normalized by the area of the minimal unit height rectangle that encloses these points. For our experiments, we resample each time series so that all data points are unit spaced, then approximate the area under the curves using the summation method. In this case, the integrated accuracy feature $\mathcal{I}(a)$ is equivalent to the area under the expanded time series normalized by the number of data points (i.e., it is the mean of the expanded time series):

$$\mathcal{I}(a) = \frac{1}{k} \sum_{i=1}^{k} a_i \tag{30}$$

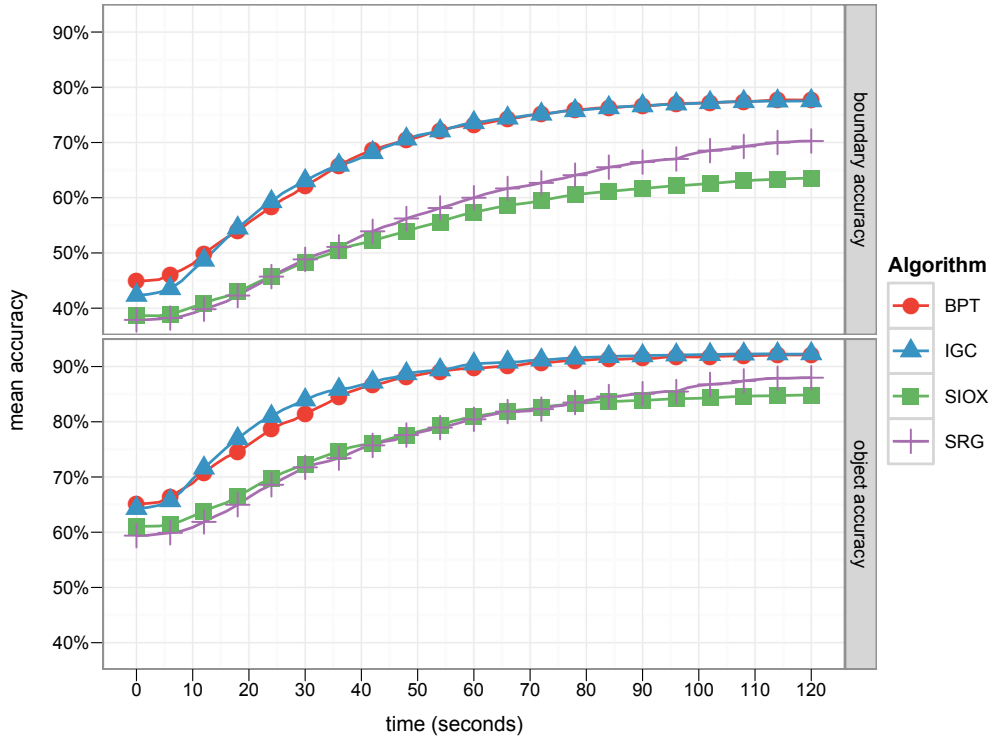

Figure 11: Mean accuracy over time as measured during the user experiments.

Once the final accuracy and integrated accuracy features for each image have been computed, they can be averaged across all objects in the dataset to obtain an indication of the overall performance of the evaluated algorithm. Furthermore, they can be used to compare the output of the automation strategies with the user experiments. Section 5.2 uses these scalar features to investigate how well the four automation strategies approximate a real user.

5.1. Experiments

We evaluated each interactive segmentation algorithm using all four automation strategies. When computing the time-accuracy profiles and the aggregate features, each time series is expanded to the maximum number of steps by duplicating the final accuracy value. The time series values for each of the non-deterministic automation strategies (strategies 2, 3, and 4) is averaged across all the repetitions. Figure 11 shows the time-accuracy profiles

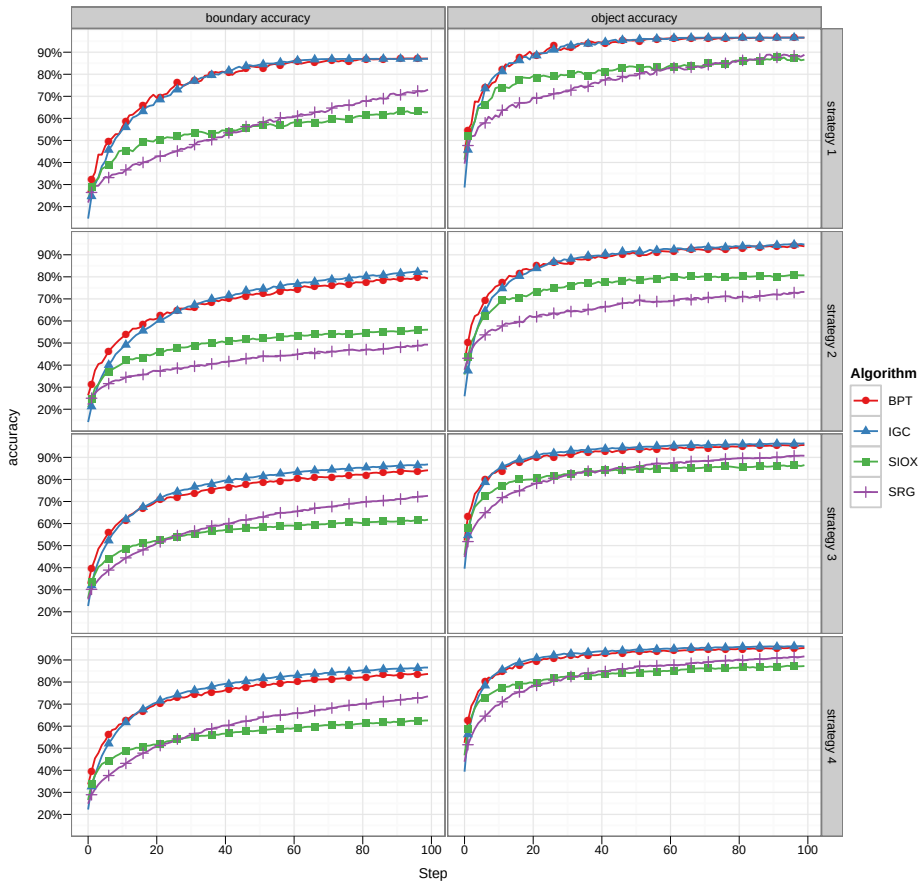

Figure 12: Mean accuracy over time for each of the automation strategies.

from the user experiments, and Figure 12 shows the step-accuracy profiles for each of the four automation strategies. Figures 13 and 14 show the average final accuracy and integrated accuracy features.

Visually comparing the profile curves from strategy 1 with the profile curves for the user experiment indicates that strategy 1 does indeed give similar results to the user experiments. Furthermore, we can draw similar conclusions from the profile curves from strategy 1 as we did from the user experiments. The strategy 1 profile curves again indicate that the BPT and IGC algorithms are comparable, both demonstrating the best overall performance. The SIOX algorithm initially performs better than the SRG algorithm. After about

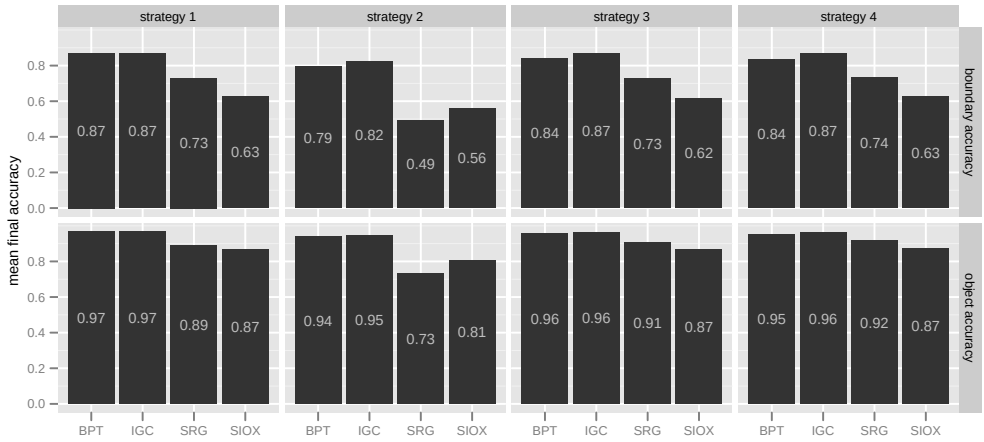

Figure 13: Mean final accuracy after 100 steps for each of the automation strategies

60 steps the performance of the SIOX and SRG algorithms are comparable. The SRG algorithm surpasses the SIOX algorithm in terms of boundary accuracy after about 60 steps, again indicating that the SRG algorithm is more receptive to iterative refinement.

Visual comparison shows that strategy 2 is less effective that strategy 1 at approximating the results of the user experiments. In particular, the time-accuracy profile for strategy 2 suggests that the SIOX algorithm consistently outperforms that SRG algorithm, a conclusion not supported by the user experiments. Strategy 3 and 4 rectify this, giving the most satisfactory visual correspondence with the profile curves from the user experiments. The same conclusions can be drawn from these profile curves as were found in the user experiments [1].

Figure 13 shows the final accuracy values for each of the strategies. The results from strategy 1, 3, and 4 largely agree. The final accuracy values for the BPT and IGC algorithms are comparable. The BPT and IGC algorithms give higher final accuracy than the SIOX and SRG algorithms. For strategy 3 and 4 the SRG algorithm gives higher final accuracy than the SIOX algorithm. Again, strategy 2 gives different results, showing the SIOX algorithm to have higher final accuracy than SRG. The integrated accuracy features in Figure 14 give similar rankings to the final accuracy features.

The reduction in accuracy between strategy 1 and 2 can be attributed to strategy 1 producing seed points closer to the center of the candidate

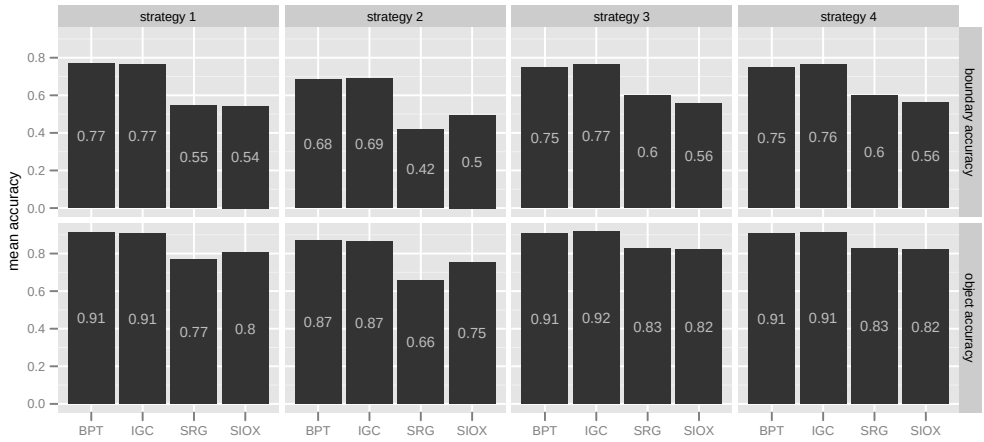

Figure 14: Mean integrated accuracy for 100 steps for each of the automation strategies

regions than strategy 2; seed points close to the center appear to produce better results for the algorithms tested than seed points nearer the boundaries. Similarly, the increased accuracy for strategy 3 and 4 can be ascribed to their producing many more seed points than strategy 2. Our primary goal, of course, is not necessarily to create an automation strategy with high accuracy, but rather one that facilitates a realistic evaluation, the results of which correlate well with those of the user experiments.

Figure 15 shows the average number of steps executed before the automation strategy terminated, for each of the segmentation algorithms evaluated. From the figure, it is clear that first strategy—that of deterministically selecting the centermost candidate pixels for each interaction—is an effective interaction pattern for the algorithms tested; on average it obtains a more accurate segmentation with less interactions than the other three strategies. This is somewhat expected, however, since the other strategies are non-deterministic, and will therefore produce interactions closer to the boundaries of the candidate regions. The overlaid bars in Figure 15 show the confidence intervals for the mean, which were computed using bootstrapping ($N = 1000, p = 0.95$). As expected, these are wider for strategy 1, since there are no repetitions, and therefore less samples.

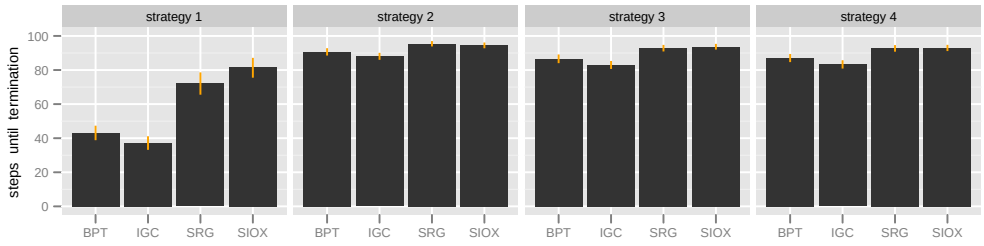

Figure 15: Average number of steps executed before the automation strategy terminates.

5.2. Correlation and validation

It is difficult to formally compare the time-accuracy profiles from the user experiments (Figure 11) with those from the automated experiments. This is due to the difficulty in aligning the time series—such alignment would require equating effort spent by an automated algorithm with effort spent by a human user in a meaningful way. The y-axis on the time-accuracy profiles is designed to quantify this effort. For the user experiments, the actual elapsed time, in seconds, is a meaningful indicator of effort. For the automated experiments, however, the number of steps is more appropriate, as each step invokes the same procedure. To compute a direct correlation between the time-accuracy profiles for the user experiments and the same profiles for the automated experiments necessitates aligning the data in some way.

To sidestep the alignment problem somewhat, we can assume that the amount of effort required by each user interaction should, on average, be roughly equivalent to the amount of effort required by each algorithmic interaction if the algorithm is roughly approximating user behavior. This is a significant assumption, and requires some further justification. First, note that while users may vary widely in the amount of effort, or markup, provided per interaction, the average value of this effort should be relatively stable (by the central limit theorem, assuming the effort-per-interaction is normally distributed across users). For an automation strategy to be effective at emulating human interactions it should produce interactions that closely correspond to those an average user would perform: the average effort-per-interaction for an automation strategy should closely correspond to the average effort-per-interaction of a user. From this, we can conclude that if an automation strategy is effective, then the accuracy profile will, on average, be well aligned with the user accuracy profile at the interaction (or step)

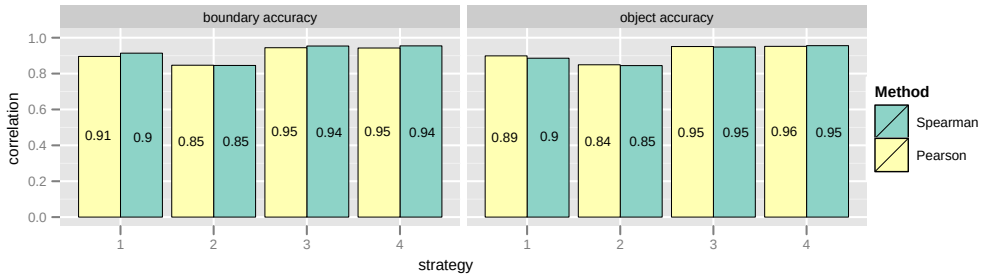

Figure 16: Stepwise mean accuracy correlation between the user experiments and the automation strategies.

level. A direct comparison between the step-accuracy profiles from the user experiments and those from each automation strategy will therefore produce higher correlation values if a given strategy is effectively emulating human interactions, since the profiles will be better aligned and the accuracy values at each step will be more similar.

Based on the above, we measured the correspondence between each of the automation strategies and the user experiments using the correlation between accuracy measures, taking as observations the mean accuracy value after each interaction, or step. The correlation was computed based on the first 60 steps only, since the majority of real users provide fewer than 60 interactions (Figure 9), and the average accuracy for the automation strategies does not change significantly after this point (Figure 12)[6]. Any missing values that arise from either the user or the automation strategy completing before 60 steps were taken to be equal to the final accuracy value.

Figure 16 compares the step-accuracy profile correlation statistics for the Spearman coefficient and the Pearson product-moment coefficient. It is clear from the Figure that the difference in correlation between strategy 3 and 4 is not significant, and that both strategy 3 and 4 correlate better with the user experiments than strategy 1 and 2.

Another method of examining the correspondence between interactions provided by real users and those provided by the automation strategies is to examine the correlation between aggregate features of the profile curves. Suppose a user is tasked with extracting two separate objects from different

[6]The extension to all 100 steps gives very similar results.

images using a particular segmentation algorithm. Usually, one of these objects will be more difficult to extract than the other; extracting it will require more time and more interactions. If an automation strategy is effective at emulating user behavior, then we would expect the same objects to be proportionately difficult for the automation strategy.

The previously discussed aggregate time series features—integrated accuracy and final accuracy—can be interpreted as being indicative of the difficulty of a segmentation. High integrated accuracy and final accuracy values indicate that segmenting a particular image is easier, low values indicate that it is more difficult. Therefore, the rank correlation over all images between the features produced by an automation strategy and the user experiments should be relatively high if the automation strategy is successfully emulating user interactions.

To perform the comparison, we proceed as follows. First, we compute the integrated score and final accuracy features for each time series generated from the user experiments. When several users have segmented the same image with the same segmentation algorithm, we average the features across the different users. This gives us two integrated score values (one for boundary accuracy and one for object accuracy) and two final accuracy values for each image and algorithm evaluated. We follow a similar procedure for each automation strategy, this time averaging over runs for the non-deterministic strategies.

For each algorithm and image pair, the result is a set of four feature values for each of the automation strategies: mean final boundary accuracy, mean final object accuracy, mean integrated boundary accuracy, and mean integrated object accuracy. From this we calculate the correlation between the automation strategies and the user experiments, for the two integrated score values and the two final accuracy values. We used two correlation coefficients: Spearman's ρ to measure rank correlation, and Pearsons's product-moment coefficient to measure linear correspondence.

Figure 17 demonstrates a high rank correlation between aggregate accuracy features in the user experiments and the automated experiments. Specifically, when tasks are ranked based on the average final accuracy and integrated score measures, the ranking produced by the user experiments exhibits a high correlation with that of the automated experiments. If we relate these scores to task difficulty (i.e. if we assume more difficult tasks result in lower aggregate accuracy values) then we can conclude that the real users and the simulated users generally agree on how difficult a segmentation task is. This correlation

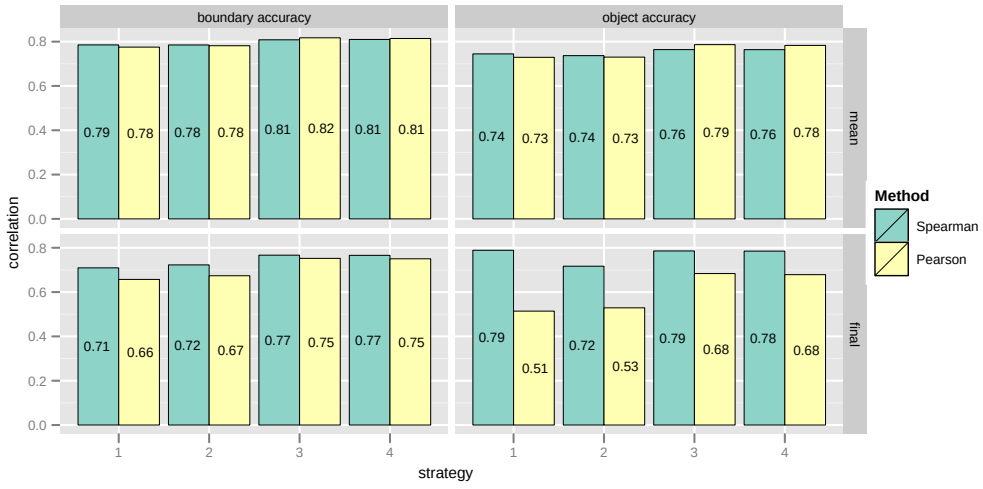

Figure 17: Correlation between aggregate features from the user experiments and each of the automation strategies.

is evidence that the automation strategies are, indeed, approximating user behavior in some useful way. The Spearman values are very similar for each of the strategies, strategies 3 and 4 giving the best overall correlation with the user experiments for integrated boundary accuracy, integrated object accuracy, and final boundary accuracy. The Pearson values show that strategies 3 and 4 give the highest linear correlation for all four aggregate features.

Figure 18 shows a scatterplot of the accuracy features from the user experiments plotted against the same features from the automated experiments. Systematic deviations from the regression line (dashed black) indicate disagreement between the accuracy values produced by the user experiments and those produced by the automated experiments. The figure clearly shows that there are fewer such deviations for strategies 3 and 4 than there are for strategies 1 and 2: further evidence that the more complex strategies better approximate real users. This is also supported by the Pearson coefficients in Figure 17.

6. Conclusion

When introducing a new interactive segmentation algorithm it is important to be able to compare its performance with the state-of-the-art. In our previous paper we developed a set of benchmarks and software for supervised evaluation

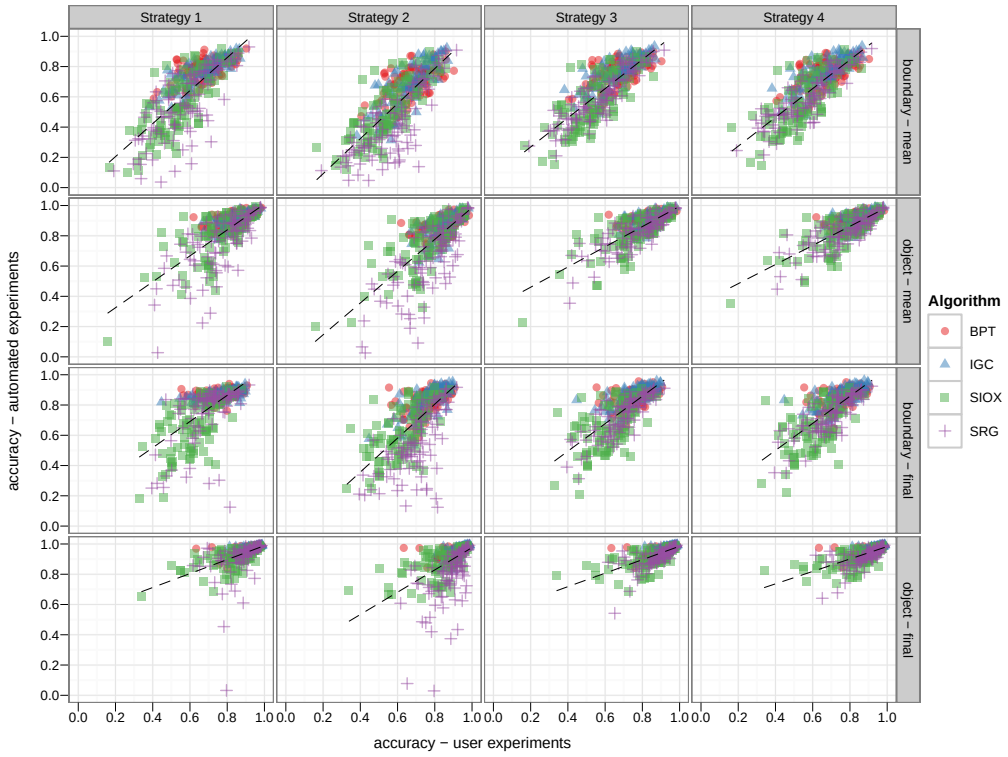

Figure 18: Accuracy features for the user experiments plotted against the same features from the automated experiments. The panels depict the automation strategies from left to right, and the accuracy features from top to bottom.

of interactive segmentation using user experiments. Carrying out these user experiments is, however, a time-consuming and labor-intensive exercise, often prohibitively so.

This paper focused on eliminating the need for user experiments. To this end, we investigated four strategies for automating the evaluation of interactive segmentation algorithms. The objective of these strategies is to simulate interactions that would normally be provided by a human operator using the ground truth and current segmentation error.

The first of these strategies is a simple, deterministic strategy: it always produces the same set of interactions given the same segmentation algorithm and input. The remaining three strategies are non-deterministic, and therefore also allow evaluating the repeatability of an algorithm. Strategies 3 and 4

produce lines and curves instead of simple point interactions, aiming to more closely approximate the kinds of interactions usually produced by humans.

The experiments demonstrated that the results of the automated experiments are very similar to those of the user experiments. Evaluating the four segmentation algorithms using strategies 1, 3, and 4 all produced similar conclusions about the evaluated algorithms, and these conclusions agreed with the previously conducted user experiments. The correlation between the user experiments and the automation strategy using both stepwise accuracy and aggregate features indicated that strategy 3 and 4 are the most effective at approximating real user input.

Based on this analysis, we recommend using automation strategy 4 for practical experiments; of the four strategies, strategy 4 produced the time-accuracy profile curve that had the closest visual correspondence with the profile curves from the user experiments, and, along with strategy 3, produced aggregate features that had higher rank correlation with the user experiments. We recommend using a maximum of 60 steps; overall accuracy does not vary much after this point, and using too many steps can result in a rather unrealistic evaluation, since users rarely spend a lot of time correcting minor errors near the boundary of objects.

If user experiments are feasible, then they are certainly the most effective way to evaluate interactive segmentation: it is difficult to evaluate any interactive system without getting feedback from real users. The automation strategies that we discussed in this paper are perhaps most useful when used as a preliminary step in an evaluation process. They allow algorithm developers to experiment with different variants of an algorithm to determine which is the most effective, without having to re-conduct an entire set of user experiments each time. Automated evaluation also provides a means for researchers to determine if a particular approach to interactive segmentation appears to have practical merit, if it requires further consideration, if it needs modification, or if it should be abandoned, before expensive user-experiments are undertaken.

Of course, if they are feasible, user experiments should be performed for the final evaluation of an algorithm when comparing it against the state-of-the-art. In their absence, however, an automated evaluation, even if it is imperfect, will be more informative than no evaluation whatsoever. Our system enables researchers to perform a useful and informative evaluation of their algorithms, even when full user experiments are impracticable.

6.1. Future work

There are several aspects of the automation algorithms that could be improved. In particular, for strategy 3 and 4 the generated interactions are not as smooth as we would expect human interactions to be. This could be at least partially ameliorated by convolving the generated list of points with a Gaussian kernel, provided sufficient precautions are taken to ensure the convolved points do not intersect the set of non-candidate pixels.

We used sum-normalized distances in our a probability mass function for selecting candidate pixels (Eq. (19)); although this appears to give reasonable results in practice, alternative mass functions may be worth investigating. Other possible weighting functions for the shortest path algorithm used by strategy 4 (Eq. (29)) may also merit investigation.

A method for directly comparing real users interactions and simulated interactions would undoubtedly simplify any investigation of potential improvements to the automation strategies. Developing such a method is, however, far from straightforward. The user interactions and automated interactions are both reactive and non-deterministic, which precludes a direct spatial comparison. It may, however, be possible to compare interactions based on shape or some other feature. A detailed analysis of user interactions may provide more insight into how direct comparison may be realized.

Finally, although we use five repetitions to ensure repeatability in our experiments, it is unclear whether this number is sufficient or necessary. Analyzing the accuracy variance for increasing repetition counts my help answer this.

Acknowledgements

This material is based upon the work supported by the European Commission under Contract FP6-027026, K-Space: Knowledge Space of semantic inference for automatic annotation and retrieval of multimedia content. This work is supported by Science Foundation Ireland under Grant 07/CE/I1147 (CLARITY: Center for Sensor Web Technologies).

Appendix

The interactive segmentation tool, ground truth dataset, and instructions for developing plugin modules are available are available on our interactive

segmentation website:
http://kspace.cdvp.dcu.ie/public/interactive-segmentation

Software for performing and visualizing an automated evaluation are available on our interactive segmentation evaluation website:
http://kspace.cdvp.dcu.ie/public/segmentation-evaluation

References

[1] K. McGuinness, N. E. O'Connor, A comparative evaluation of interactive segmentation algorithms, Pattern Recognition 43 (2) (2010) 434–444. doi:10.1016/j.patcog.2009.03.008.

[2] J. K. Udupa, Y. Zhuge, Delineation operating characteristic (DOC) curve for assessing the accuracy behavior of image segmentation algorithms, Proceedings of the SPIE 5370 (1) (2004) 640–647. doi:10.1117/12.535808.

[3] J. K. Udupa, V. LeBlanc, Y. Zhuge, C. Imielinska, H. Schmidt, L. Currie, B. Hirsch, J. Woodburn, A framework for evaluating image segmentation algorithms, Computerized Medical Imaging and Graphics 30 (2) (2006) 75–87. doi:doi:10.1016/j.compmedimag.2005.12.001.

[4] X. Jiang, C. Marti, C. Irniger, H. Bunke, Distance measures for image segmentation evaluation, EURASIP Journal on Applied Signal Processing 1 (2006) 290–300. doi:10.1155/ASP/2006/35909.

[5] F. Ge, S. Wang, T. Liu, New benchmark for image segmentation evaluation, Journal of Electronic Imaging 16 (3) (2007) 033011. doi:10.1117/1.2762250.

[6] H. Zhang, J. Fritts, S. Goldman, Image segmentation evaluation: a survey of unsupervised methods, Computer Vision and Image Understanding 110 (2) (2008) 260–280. doi:10.1016/j.cviu.2007.08.003.

[7] M. Riedl, R. Amant, Toward automated exploration of interactive systems, in: Proceedings of the 7th International Conference on Intelligent User Interfaces, 2002. doi:10.1145/502716.502738.

[8] C. Foley, A. F. Smeaton, Synchronous collaborative information retrieval: Techniques and evaluation, in: Advances in Information Retrieval, Proceedings of the 31st European Conference on Information Retrieval, 2009, pp. 42–53. doi:10.1007/978-3-642-00958-7_7.

[9] S. Jung, C. Lee, K. Kim, M. Jeong, G. G. Lee, Data-driven user simulation for automated evaluation of spoken dialog systems, Computer Speech & Language 23 (4) (2009) 479–509. doi:10.1016/j.csl.2009.03.002.

[10] F. Mao, J. D. Gill, A. Fenster, Technique for evaluation of semi-automatic segmentation method, Proceedings of the SPIE: Medical Imaging 3661 (1999) 1027–1036. doi:10.1117/12.348496.

[11] E. Moschidis, J. Graham, Simulation of user interaction for performance evaluation of interactive image segmentation methods, in: Proceedings of the 13th Medical Image Understanding and Analysis Conference, 2009, pp. 209–213.

[12] E. Moschidis, J. Graham, A systematic performance evaluation of interactive image segmentation methods based on simulated user interaction, in: Proceedings of the IEEE International Symposium on Biomedical Imaging, 2010, pp. 928–931.

[13] T. Adamek, Using contour information and segmentation for object registration, Ph.D. thesis, Dublin City University, Ireland (2006).

[14] Y. Boykov, M.-P. Jolly, Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images, in: Proceedings of the 8th IEEE International Conference on Computer Vision, 2001, pp. 105–112. doi:10.1109/ICCV.2001.937505.

[15] R. Adams, L. Bischof, Seeded region growing, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (6) (1994) 641–647. doi:10.1109/34.295913.

[16] G. Friedland, K. Jantz, R. Rojas, SIOX: simple interactive object extraction in still images, in: Proceedings of the 7th IEEE International Symposium on Multimedia, 2005, pp. 253–260. doi:10.1109/ISM.2005.106.

[17] S. D. Olabarriaga, A. W. M. Smeulders, Interaction in the segmentation of medical images: a survey, Medical Image Analysis 5 (2) (2001) 127–42. doi:10.1016/S1361-8415(00)00041-4.

[18] A. X. Falcão, J. K. Udupa, S. Samarasekera, S. Sharma, B. E. Hirsch, R. de A. Lotufo, User-steered image segmentation paradigms: live wire and live lane, Graphical Models and Image Processing 60 (4) (1998) 233–260. doi:10.1006/gmip.1998.0475.

[19] J. Liu, J. K. Udupa, Oriented active shape models, IEEE Transactions on Medical Imaging 28 (4) (2009) 571–584. doi:10.1109/TMI.2008.2007820.

[20] D. Martin, C. Fowlkes, D. Tal, J. Malik, A database of Human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics, in: Proceedings of the 8th International Conference Computer Vision, Vol. 2, 2001, pp. 416–423. doi:10.1109/ICCV.2001.937655.

[21] F. Ge, S. Wang, T. Liu, Image-segmentation evaluation from the perspective of salient object extraction, in: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006, pp. 1146–1153. doi:10.1109/CVPR.2006.147.

[22] R. Fabbri, L. D. F. Costa, J. C. Torelli, O. M. Bruno, 2D Euclidean distance transform algorithms: a comparative survey, ACM Computing Surveys 40 (1) (2008) 1–44. doi:10.1145/1322432.1322434.

[23] J. Spolsky, User Interface Design for Programmers, Apress, Berkely, CA, USA, 2001.

[24] A. Meijster, J. Roerdink, W. Hesselink, A general algorithm for computing distance transforms in linear time, in: Proceedings of the 5th International Symposium on Mathematical Morphology and its Applications to Image and Signal Processing, Vol. 18, Springer, 2000, pp. 331–340. doi:10.1007/0-306-47025-X_36.

[25] W. Hörmann, J. Leydold, G. Derflinger, Automatic Nonuniform Random Variate Generation, Springer, 2004.

[26] T. H. Cormen, S. Clifford, C. E. Leiserson, R. L. Rivest, Introduction to Algorithms, 2nd Edition, MIT Press, 2001.

[27] K. E. Atkinson, An Introduction to Numerical Analysis, 2nd Edition, Wiley, 1989.