Efficient User-Guided Ballot Image Verification

Arel Cordero
UC Berkeley
arel@cs.berkeley.edu

Theron Ji
UC Berkeley
therji@berkeley.edu

Alan Tsai
UC Berkeley
alan_tsai@berkeley.edu

Keaton Mowery
UC San Diego
kmowery@cs.ucsd.edu

David Wagner
UC Berkeley
daw@cs.berkeley.edu

Abstract

Optical scan voting systems are ubiquitous. Unfortunately, optical scan technology is vulnerable to failures that can result in miscounted votes and lost confidence. While manual counts may be able to detect these failures, counting all the ballots by hand is in many situations impractical and prohibitively expensive. In this paper, we present a novel approach for examining a large set of ballot images to verify that they were properly interpreted by the opscan system. Our system allows the user to *simultaneously* inspect and verify many ballot images at once. In this way, our scheme is significantly more efficient than manually recounting or inspecting ballots one at a time, providing the accuracy associated with human inspection at reduced cost. We evaluate our approach on approximately 30,000 ballots cast in the June 2008 Humboldt County Primary Election and demonstrate that our approach improves the efficiency of human verification of ballot images by an order of magnitude.

1 Introduction

In recent years, optical scan systems, where paper ballots are marked by a voter and read by a machine, have seen increasing adoption throughout the U.S. At present, nearly two-thirds of voters vote using optical scan technology [19]. Unfortunately, there is sometimes a disconnect between the way a voter (or election official) might interpret a mark and the way a machine interprets it. This disconnect can lead to lost or miscounted votes and even to a loss of trust if these errors become too frequent.

At a high level, optical scan systems work by scanning a paper ballot, determining its layout by reading a machine-readable code on the ballot (e.g., a barcode), and then interpreting the voter-marked regions on the ballot associated with each vote (the *voting targets*) [10]. Depending on the ballot layout, a filled voting target may indicate a vote for "Yes" or "No" on a contest or a vote for a particular candidate.

A simple and common approach to mechanically detect whether a voting target is marked is to compute the average pixel intensity value of the area inside the voting target, classifying it as marked if the average falls beyond a predetermined threshold. Typically, the vast majority of voting targets fall into one of two cleanly separated distributions, according to whether the voting target was marked or not (as in Figure 15). However, the threshold that optical scanners and their software use to determine whether a voting target is filled or empty can be somewhat arbitrary and is unknown to the end user. In addition, not all marks fall cleanly into the "filled" and "empty" classes. In the sample of 60,000 voting target images that we analyze in this paper, 150 (0.25% of the total) are *marginal*, in that their average intensity is neither near the typical value for a filled target nor near the typical value for an empty target (for examples, see Figure 1). Current opscan systems often have trouble interpreting these marginal marks accurately.

Another serious problem is that the classification of voting targets as filled or empty is not a perfect indication of the voter's intent. For example, a voter, perhaps unaware of the mechanical process used to interpret his or her ballot, may have marked the ballot reasonably, but in a way that is misinterpreted by the scanner. We call these *mark-sensing failures*. See Figure 2 for two examples of mark-sensing failures. Mark-sensing failures have impacted elections in the past [1, 4].

A second category of optical scan system failures is *configuration failures*. Beyond sensing marks, an optical scan system must determine which candidate or contest choice each mark corresponds to, and whether the marks should be counted as valid votes. For example, in a vote-for-one contest, marking two selections should count as vote for neither (i.e., an *overvote*). The logic regulating these decisions can get complex, leading to inadvertent errors. Errors in the opscan configuration can cause votes to be attributed to the wrong candidate, cause valid votes to be mistakenly treated as overvotes, or cause overvotes

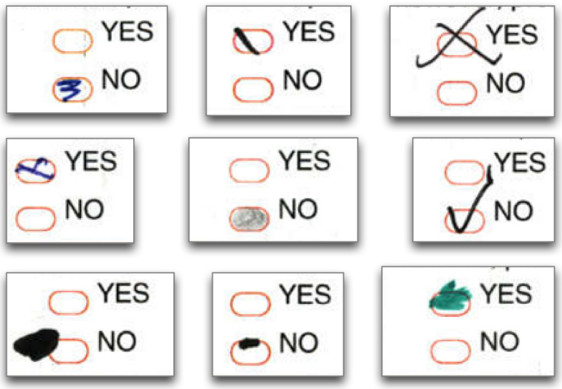

Figure 1: Examples of *marginal marks* from our data set according to our classification procedure (see Figure 15). Commercial opscan systems generally do not make clear where the boundary lies between marks that will be classified as filled and those that will be interpreted as empty. Marks near that boundary, such as these, may not be reliably (or even consistently) classified the way the voter intended.

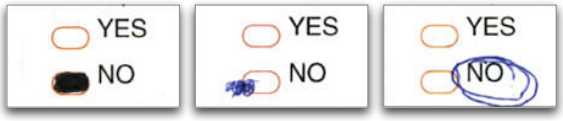

Figure 2: Three real-life examples of "No" votes: a correctly classified filled voting target (left), a marginal mark (center), and a mark that was incorrectly classified as an empty voting target because there is little ink inside the voting target itself (right). The latter two cases are examples of potential mark-sensing failures of an optical scan system.

to be mistakenly interpreted as valid votes. Configuration failures have impacted elections in the past [16, 21, 2].

These kinds of failures have the potential to change election outcomes, so it is important to have some way to detect and correct them. For example, the 2008 Minnesota Senate race was initially decided by a margin of 206 votes out of 2.9 million optical scan ballots cast [14, 5]. After a lengthy recount of all the ballots and challenges over voter intent, the outcome of the election was reversed.

As in Minnesota, one approach to verifying an election is to count the paper ballots manually in either a full recount or a statistical random audit [6]. Unfortunately, full recounts are expensive and rare, and random audits do not detect all instances of mark-sensing and configuration failures.

Another approach (pioneered by the Humboldt County Election Transparency Project [7]) is to independently rescan the paper ballots with an off-the-shelf document scanner and publish the digitized images. While now the digitized images must be trusted, this allows other interested parties to verify the classification of these images into votes, either manually, or by writing their own software to do so.

Publication of ballot images enables an independent automated recount or manual analysis of the ballots. The hope is that two independent systems will fail independently, thus reducing the risk of undetected errors. However, this hope is not always borne out in practice, due to the risk of common-mode failures and correlated errors. For example, two systems that both interpret voting targets as filled or empty based on the image intensity inside the voting target may both fail to correctly identify the voter's intent when the voter has expressed his or her intent in an unanticipated way (see Figure 2).

In this paper, we focus on trying to confirm whether the opscan system has accurately interpreted a set of ballot images. For this paper, we define accuracy in terms of how a human examining those images would interpret the voter's intent on each ballot. The difference between human interpretation and machine interpretation can be significant, particularly when interpreting marginal or ambiguous marks. Unfortunately, while human interpretation can be very effective at ascertaining voter intent, it is also time-consuming and prohibitively expensive to perform at large scale. Reducing the cost of human interpretation is the goal of this work.

1.1 Problem statement

Given a set of ballot images of an election and the opscan system's interpretation of each ballot, we want to allow the user of our system to efficiently check the machine's interpretations. We want to detect every error that would have been found through individual inspection of every ballot image, but we want to do this more efficiently. In other words, our goal is to make it easier for a user to confirm that all ballot images were interpreted accurately, verifying for him or herself that all votes have been counted correctly and that all anomalous ballots are accounted for. The user should not have to take it on faith that the optical scanner interpreted the ballots correctly: the user should be able to detect any and all ballots that may have been interpreted inaccurately, whether due to buggy code or to imperfect classification algorithms.

Currently, the most rigorous way to establish that a set of ballot images has been counted accurately is to count them one by one. However, manual recounts are time-consuming and difficult. It is possible to develop software to analyze and automatically classify votes, flagging anomalous or marginal votes for later inspection. However, the accuracy of the software's interpretation of

the ballots and the effectiveness of the software's algorithms must be trusted by the user. In addition, such software is also subject to the same mark-sensing and configuration failures as optical scanners, because everything is automated. We seek some way for the voter to confirm that the software interpreted the ballots accurately. *It is hard to establish the accuracy of the count without actually* ***inspecting*** *the ballots.*

We make a few assumptions for the purposes of this work. First, we assume that the given ballot images are authentic, and that they accurately represent the complete set of ballots to be investigated. Verifying this is an orthogonal issue best addressed through other mechanisms (e.g., random auditing and chain-of-custody protections). Second, we assume our user's subjective interpretation of a ballot is the ground truth. Our goal is simply to present the data efficiently for the user to verify. We make no attempt to determine how other users might interpret the ballot images, and we do not claim that our techniques will resolve disputes in cases where different people interpret the same ballot differently.[1] Third, we make no attempt to defend against malicious code, malicious insiders, or malicious attacks on the voting system. Our focus is on detecting common errors that may occur despite the best intentions of all parties.

Contributions of this paper. We propose a method of visualizing and checking for errors in the interpretation of a collection of ballot images, by applying simple operators to the images and displaying them in superimposed form. We develop an interactive verification process that significantly improves efficiency and reduces the burden on the user of the system while at the same time allowing individual treatment of each anomalous ballot. We demonstrate the effectiveness of our approach by using it to obtain a human-verified interpretation of 30,000 votes cast in Proposition 98 in Humboldt County, California. We use this interpretation to identify a number of anomalies and unusual marks found in these ballots and quantify their prevalence.

2 Superimposing ballot images

The primary contribution of this paper is a method of verifying a large set of ballot images by inspecting a small number of superimposed images. In this section we introduce the idea and theory of two types of superimposed images, *min-images* and *max-images*, and how they apply to verification. To introduce this concept we briefly review the basics of digital images.

[1] It is not unusual for two people to differ in their interpretation of voter intent when the stakes are high, as is evident in the 2008 Minnesota Senate election recount [14].

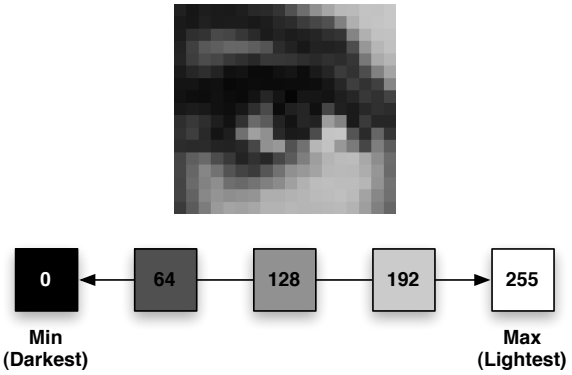

Figure 3: An image represented by a matrix of pixels (top). The intensity at each pixel is represented by an 8-bit value ranging from 0 to 255 (bottom). The minimum of a set of pixel values is therefore the darkest pixel, while the maximum is the lightest.

A digital image (e.g., a scanned paper ballot) may be thought of as a matrix of *pixels*, each corresponding to the color or intensity of the image at that position. In 8-bit grayscale images, pixel values are numbers between 0 and 255 that represent the brightness, or *luminance intensity*, of the pixel (see Figure 3). Color images are similar, except that they are usually composed of multiple *channels* (e.g., red, green, and blue channels), instead of just one (luminance).

2.1 Min-images and max-images

Consider a set of N grayscale images of the same contest, each $I \times J$ pixels in size and aligned to one another (or *registered*). Two interesting operations we can consider on these images are the minimum and maximum operations. As shown in Figure 4, for each pixel position (i, j), we compute the minimum and maximum at that position in each of the N images. The result of these operations are stored in the *min-image* and *max-image*, respectively.

If all the images are identical, the value of the min- and max-images would also be identical. However, variation in any pixel of the N images will result in differing minimum and maximum images, and the difference between the minimum and maximum at any pixel location defines a range in which all pixels in the set necessarily lie (see Figure 5).

We can think of how the minimum and maximum operations would appear visually for a given pixel location: because pixel values correspond to luminance intensity, the smaller the value, the darker the pixel becomes. Therefore, the minimum operator on a set of pixels is equivalent to taking the darkest pixel in the set. For example, if even just one pixel is black, the resulting min-

```
MIN-AND-MAX-IMAGES(images)
1  min_image ← new solid white image of size I × J
2  max_image ← new solid black image of size I × J
3  for j ← 0 to J − 1
4      do for i ← 0 to I − 1
5             do for image in images
6                    do min_image_{i,j} ← min(image_{i,j}, min_image_{i,j})
7                       max_image_{i,j} ← max(image_{i,j}, max_image_{i,j})
8  return (min_image, max_image)
```

Figure 4: Pseudocode for calculating min- and max-images.

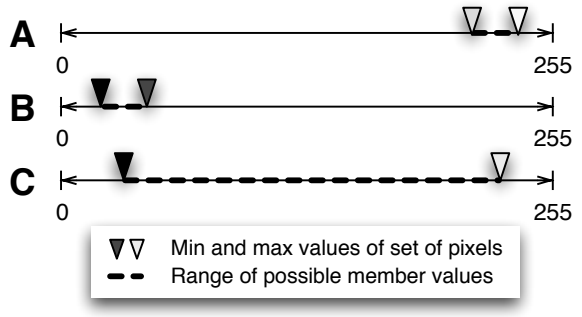

Figure 5: Three example cases, representing the minimum and maximum intensity values of a set of images at three pixel locations, A, B and C. In cases A and B, the min- and max-images have almost the same value at that pixel location, indicating that there is little variation among the pixel values at that location from image to image. Case C gives us fewer guarantees about the member values; all we know is that there is broad variation between at least one pair of images at that pixel location. For instance, this variation could be due to a stray mark at that pixel location in one ballot image, which might warrant further investigation.

imum pixel will be black. Similarly, the maximum pixel value corresponds to the lightest among a set of pixels. If any pixel in the set is white, the maximum will also be white.

2.2 Using superimposed ballot images for verification

We leverage the min- and max-images for simultaneous verification of large sets of aligned ballot images.

The min-image. If we make the assumption that when a voter marks a blank ballot, he or she *darkens* the image, we can consider the min-image to consist of the *union* of all of the voters' marks, superimposed on a single image. In other words, if one ballot image contains an extraneous mark, the min-image will also contain that mark, as well as any other marks that exist in the set of ballot images.

If we see *no* questionable stray marks in the min-image, we are guaranteed that *none* of the ballot images contain stray marks. If we do see one or more questionable marks, we know that there exists at least one stray mark in the ballot images and we must investigate them further.

Beware: a min-image that looks like a correctly marked ballot (e.g., a valid "Yes" vote) does not guarantee that all (or any) of the ballots in the set are correctly marked. The min-image can be used to verify the *absence* of any marks at a given location across all of the ballots, but it cannot be used to verify the presence of a mark in all ballot images.

The max-image. While the min-image can be considered the union of all voter marks, the max-image can be thought of as the *intersection*. In other words, if any ballot is white at a particular location, then the max-image will also be white at that location. This property can be used to detect whether the collection contains a ballot with whitespace where it should not be, such as an incompletely filled voting target or missing text that was supposed to be printed on the ballot but is not present in the scanned image. Therefore, if we see a valid mark in the max-image, we are guaranteed that *all* ballots in the set also contain that mark.

Verifying ballot images. The use of min-images and max-images can thus let us establish strong guarantees about the superimposed ballots:

- If we see a pair of min- and max-images that are similar (or identical) we know that all ballots contained are similar. For example, see Figure 6.
- If we see a pair of min- and max-images that are dissimilar (e.g., because of a stray mark or under-

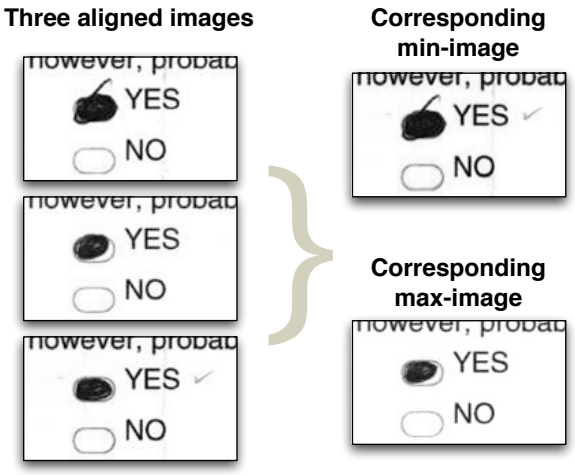

Figure 6: A visual example of the min and max image operators. From the min-image we notice a subtle check mark, indicating the existence of such a stray mark in one of the superimposed ballots. We also notice that none of the superimposed images have any marks whatsoever that could be construed as a "No" vote. The max-image shows us what the ballots have in common: that they all have a significant amount of ink in the "Yes" voting target. From the two superimposed images, we can reasonably conclude that all three ballots are unambiguous "Yes" votes.

filled voting target) then we cannot verify the ballot images, and further investigation is required. For example, see Figure 7.

The min/max operations can be applied to any number of images. For example, take the two images shown in Figure 7. From the image on the left we learn that at least one of the 800 ballots contains an anomalous arrow pointing to the "Yes" label. Similarly we see that at least one ballot appears to contain the initials "CC" We also see that at least one ballot contains a hesitation mark in the "No" voting target.

From the image on the right we learn that all 800 ballots contain at least as much shading in the "Yes" voting target as is shown in that image. We also learn that all 800 ballots correctly contain the same textual content; no ballot in the set omits or alters any text.

From these two images, barring the extraneous marks that we may want to investigate, we can reasonably conclude that all votes in the stack should be classified as "Yes" votes. Through the use of these two images we are able to establish bounds on the amount of variation that exists in the 800 images. This enables a user to efficiently verify a large set of ballot images simultaneously.

2.3 Overlay images

One slightly unwieldy aspect of examining min- and max-images is that the operator must examine two images at the same time. However, it is possible to improve the presentation by summarizing all the information in a single image: the *overlay image*. The key observation is that the max-image is a "subset" of the min-image: every pixel in the min-image is at least as dark as the corresponding pixel in the max-image. Because we have elected to work with grayscale images, it is possible to combine the min- and max-images into a single image by assigning one color to the max-image and another color to the min-image and superimposing the two (see Figure 8). We call the superimposed false-color image an *overlay image*. Example overlay images are shown in Figure 9.

The examples in this paper use red for the min-image and blue for the max-image.[2] Therefore, blue regions of the overlay image correspond to locations where every ballot is equally dark (e.g., due to marks or printing present on every ballot in the set), and red regions correspond to locations where at least one ballot is dark and at least one is light (e.g., due to stray marks or variation from ballot to ballot). Text printed on the blank ballot appears in blue on the overlay image. The surrounding red "halo" is due to imperfect registration and alignment of images.

3 Verifying elections using overlay images

We envision that overlay images could form part of a system for auditing and verifying the interpretation of marked ballots. The process, illustrated in Figure 10, can be split into three parts, each with a different set of considerations:

1. *Image scanning*: This is the process of taking paper ballots and scanning them into an image format for use by our system. Several factors here may influence accuracy later, such as skew angle, image resolution, scanner noise, and artifacts from image compression. The details of how images are scanned is beyond the scope of this paper, and we consider the scanned images as inputs into our system.

2. *Image analysis*: This is the process of grouping and aligning the ballots, identifying and segmenting the regions for each contest of interest, and tentatively classifying the marks of the voter. Note that automatic classification of voter marks is not the primary focus of this paper; our goal is to *verify* the

[2]Note that red and blue are parameters to our overlay image approach; any two colors may be used instead to account for the perceptual needs or preferences of the user.

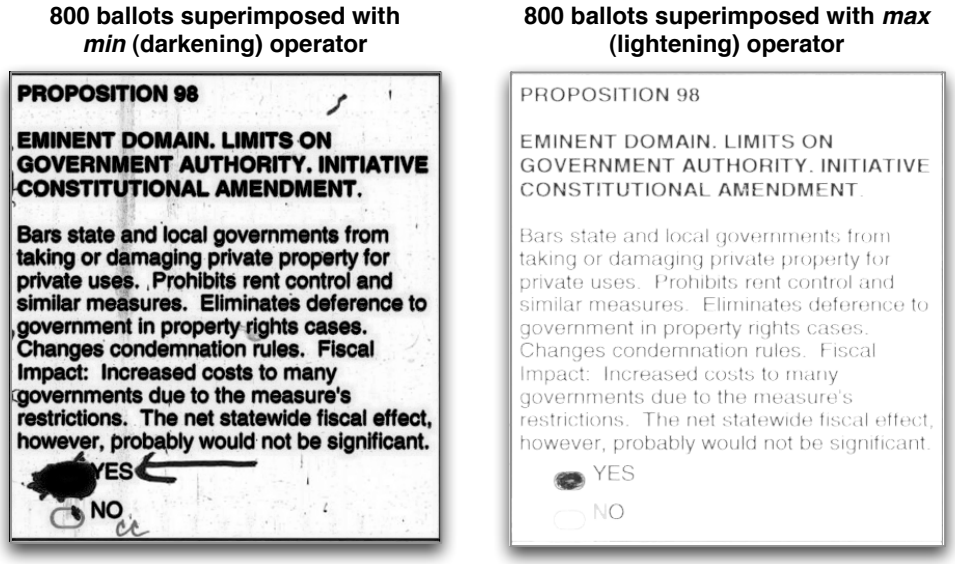

Figure 7: A superposition of 800 ballots, all initially classified as "Yes" votes. The min-image is shown on the left, and the max-image on the right. From these two images we conclude that further investigation is required to identify the ballots with stray marks. However, we also learn from the left image that *none* of the 800 voters (completely) filled the "No" choice, and we learn from the right image that *all* 800 voters filled the "Yes" choice at least that much, so we can reasonably predict that the superimposed ballots are all "Yes" votes.

OVERLAY-IMAGE($min\_image, max\_image, min\_color, max\_color$)
1 $overlay\_image \leftarrow$ new $I \times J$ color image
2 **for** $j \leftarrow 0$ **to** $J - 1$
3 **do for** $i \leftarrow 0$ **to** $I - 1$
4 **do** $\alpha_{min} \leftarrow min\_image_{i,j} \,/\, 255$
5 $\alpha_{max} \leftarrow max\_image_{i,j} \,/\, 255$
6 $t \leftarrow (1 - \alpha_{min}) \cdot min\_color \; + \; \alpha_{min} \cdot$ *white*
7 $overlay\_image_{i,j} \leftarrow (1 - \alpha_{max}) \cdot max\_color \; + \; \alpha_{max} \cdot t$
8 **return** $overlay\_image$

Figure 8: Pseudocode for creating a single overlay image from a given min- and max-image. An overlay image is created by interpolating or *alpha blending* first between the background color (white) and the *min-color*, and then between the resulting color and the *max-color*, in other words treating the min- and max-images as *alpha layers*. In Appendix A we describe and compare this to an alternate overlay image algorithm.

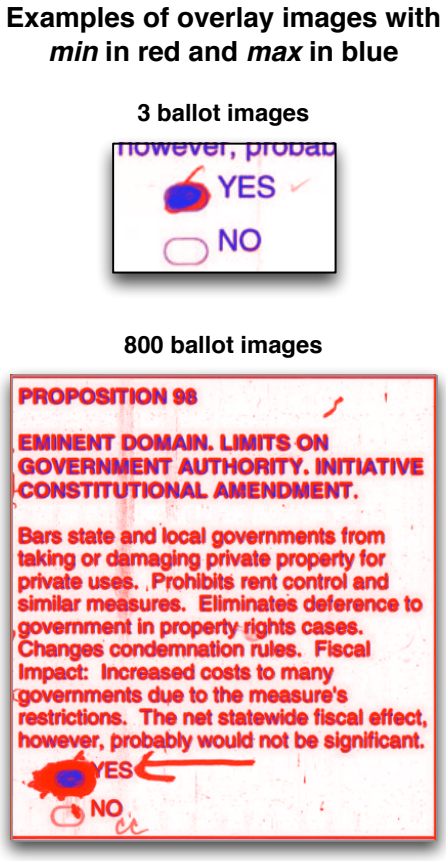

Figure 9: Three images from Figure 6 represented by a single overlay image (top). The result of superimposing the min-image (as red) and max-image (as blue) from Figure 7, to obtain an overlay image (bottom).

accuracy of such a subsystem. We could have, for instance, replaced the classification procedure with another software component or with the official optical scan system.

3. *Image verification*: In the image verification stage, the user (e.g., a voting official certified to make judgment calls) checks the accuracy of the interpretations made in the image analysis stage. Our system provides an interactive tool that enables the user to review, verify, and if necessary correct the interpretation of ballot images as votes. We implement a simple user interface to let the user efficiently validate large batches of ballots and hone in on problem or anomalous ballots. The image verification stage is the core of our contribution, and we expand below on the major elements of this approach to image verification.

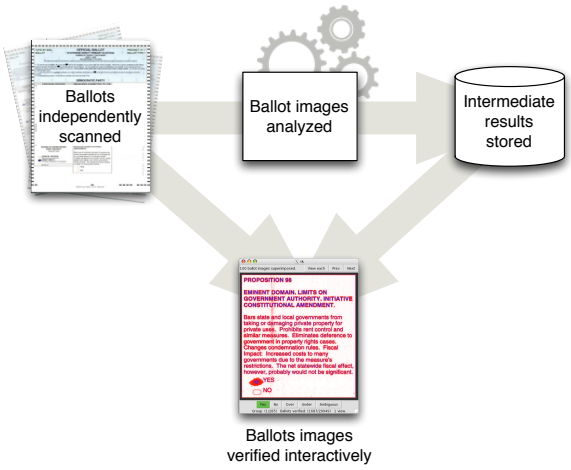

Figure 10: The proposed workflow for our system. Ballots are first scanned and analyzed. The ballot images are tentatively classified as votes for a particular choice or candidate (see Section 4.2). The tentative classifications are then verified using our interactive image-overlay tool (see Section 3).

3.1 Grouping ballots into verifiable sets

Given a set of aligned and classified ballot images, we turn our attention to the process of verifying the classification of each image as a set of votes.

Before we can verify ballots using our min/max overlay approach, we must group ballots into visually similar sets. For instance, we group all the "Yes" votes, "No" votes, overvotes, and undervotes together. We will verify each of these four groups separately. Note, each group may have many images in it. In our experiment described below there were over 10,000 ballots in each of the "Yes" and "No" groups.

If the ballot images are very well aligned, and if there is very little noise in the images, it may be possible to overlay all the ballots in one group simultaneously. However, in practice, there is a limit to the number of images that can be superimposed before the noise becomes too great. Thus, it is helpful to break down each group into smaller groups (e.g., of maximum size 200). This still lets us verify many ballots at a time, while limiting the effect of noise.

3.2 User interface to verify ballot groups

Once we have grouped ballot images into visually similar groups, our tool allows the user to step through them, one group at a time, to verify them. To do so we created a graphical user interface that sequentially presents the user with the min/max overlay image for each group.

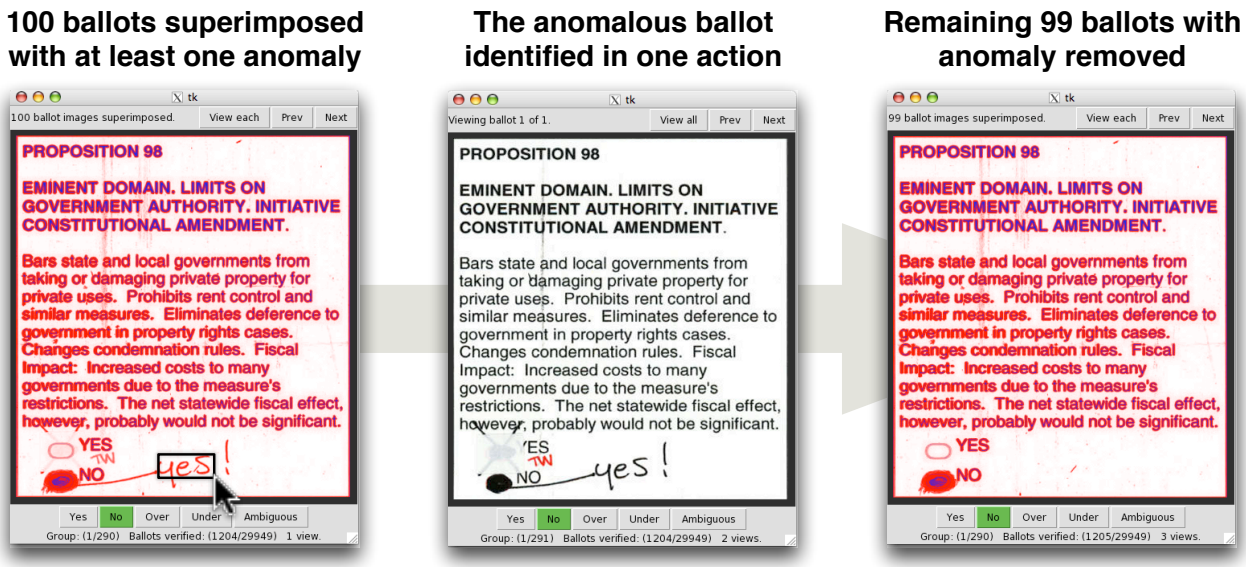

Figure 11: Our user interface showing how an anomaly can be easily identified and separately verified. 1.) An overlay of 100 "No" ballots presented; the user notices an anomaly (a written "Yes!") and selects it by drawing a box (left). 2.) Our system finds the one ballot containing the highlighted anomaly and displays it for the user to verify separately (center). 3.) The user verifies the remaining 99 ballots, now clean of any questionable marks (right).

The verification tool begins with all the ballot groups in a queue, presenting the overlay image for each group one at a time. If the overlay image looks correct to the user, he or she can verify the group of ballots with one click. When a group is verified, it is removed from the queue. When the queue is empty, all ballots have been verified. The interesting part of the process happens when an anomaly is detected, or when a group cannot immediately be verified.

3.3 Dealing with anomalies

In Figure 11 we see an example of the user interface displaying the overlay image for a group with an anomaly in it. We see that at least one ballot has a big "Yes!" written on it. In order to verify this group we need to deal with this anomalous ballot.

We approach this problem by letting the user easily partition groups of ballots based on a region of interest. We allow the user to select a rectangular region in the overlay image, and then we split the group of ballot images into two subgroups based upon their contents within this rectangular region. In the example shown in Figure 11, when the user draws a rectangle around the "Yes!", the initial group of 100 ballots is split into two groups (in this case, one group of one ballot, shown center, and one group of 99 ballots, shown right). This can be effectively accomplished by examining the average intensity for each ballot in the region of interest. A standard clustering algorithm can be used to partition the group. We used K-means [12], with the number of clusters equal to two. The algorithm for partitioning ballot groups is given in Figure 12. The original group becomes two disjoint groups in the queue of unverified ballots.

This approach is quite general. The user, for example, is not limited to selecting regions with stray marks. Because K-means simply clusters images by similarity in average image intensity at a region, the user may select any region as a basis for clustering. This approach allows a user to interactively partition a large set of images, if necessary, into smaller groups that can then be verified. It is important to note that the quality of the initial ballot groupings affects only the *efficiency* of verification—*not the correctness*—because the user may repartition groups into smaller and smaller groups (individual ballots, if necessary) until he or she is satisfied with the verifiability of the resulting groups.

4 Evaluation

In this section we discuss an experiment we ran to evaluate the practicality of our approach.

4.1 Dataset

We evaluate our system using ballot image data from the June 2008 Humboldt County California Primary Election. The ballot scans are JPEG-compressed color

```
PARTITION-IMAGES(images, region)
1  sum_intensities ← new ASSOCIATIVE-ARRAY
2  for image in images
3      do sum_intensities[image] ← 0
4         for j ← region.y to (region.y + region.height − 1)
5             do for i ← region.x to (region.x + region.width − 1)
6                    do sum_intensities[image] ← sum_intensities[image] + image_{i,j}
7  (cluster_1, cluster_2) ← K-MEANS(sum_intensities, k = 2)
8  return (cluster_1, cluster_2)
```

Figure 12: Pseudocode for the approach we use to find a group of anomalous ballots, given a set of images and a region of interest. In this code, K-MEANS refers to the standard clustering algorithm that, in this case, partitions a set of scalar values into two clusters centered around two group means in a way that best explains the data [12].

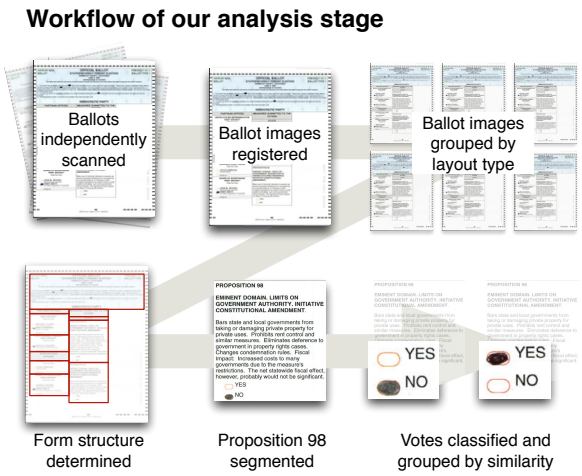

Figure 13: An illustration of the techniques we use for analyzing ballot images and interpreting the marks on these images. The images are registered (mapped to a universal coordinate system and aligned), classified by ballot layout type, segmented into regions for each contest and voting position, classified as votes for candidates, and finally broken down into smaller groups, optionally by image similarity.

images at a resolution of 150 DPI, which we obtained from the Humboldt County Election Transparency Project [7]. They include scans of both mail-in and precinct-cast ballots for a Diebold/Premier optical scan system. As this was a primary election, they include ballots for multiple parties. We limit our scope to evaluating one statewide contest, Proposition 98, that appears in each of the 29,949 ballot images in our data set.

4.2 Image analysis

The goal of the image analysis stage is to register the images to the same coordinate system, locate the contest of interest on each image (Proposition 98 in our case), and classify the votes as *Yes, No, undervote,* or *overvote* (see Figure 13). These classifications will be verified in the verification stage.

Image registration. For our overlay approach to work we require precise alignment (or *registration*) of the images to one another. To register the images, we first select features we expect to appear in the same place on all our ballots. The features we used in our experiment were the inside corners of the four outermost hash marks (as depicted in Figure 14). Next, for each ballot image, we locate the selected features and compute an affine transformation that best maps these points to the four corners of the common coordinate space. We then apply this transformation to the ballot image, which accomplishes the necessary rotation, translation, and scaling necessary to align all the images. Note that an incorrect transformation can be detected in the verification stage.[3]

We took the approach of registering the entire ballot, and then cropping it to retain the region of interest (i.e., Proposition 98 in our example). In hindsight, it may have been beneficial to re-register the images after cropping, because small errors introduced in locating the various regions accumulate when the cropped images are overlaid.

Form extraction. After registering the images, we determine the structure of the ballots through *form extraction* [3], and use this to locate the coordinates of every box in the image. In our case study, we used a heuristic to identify the location of Proposition 98 on each bal-

[3]We also found we could reliably detect mis-registered ballots before the verification stage, by examining the *residual* of the affine transformation; the residual was high whenever our simple feature detection algorithm failed to locate the correct features, typically due to the existence of stray marks. This affected 26 of the 29, 949 ballots, which we then manually registered.

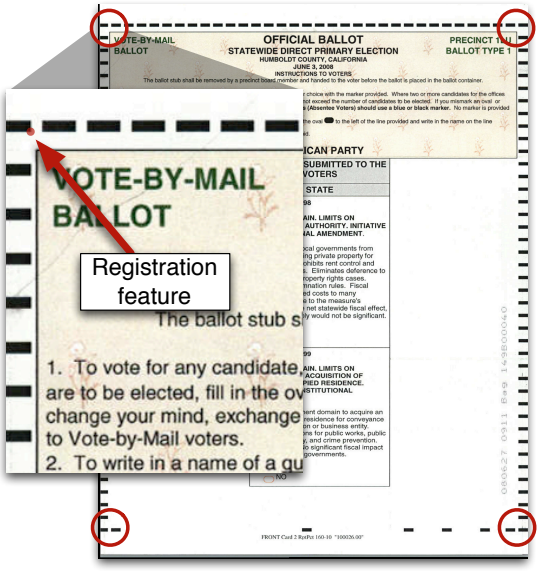

Figure 14: To precisely register the ballot images in our set, we first locate the innermost corners of the four outermost hash marks. Given the location of these four points we compute an affine transformation that best maps these four points to corresponding locations in a known coordinate system.

lot. We made an assumption (subsequently verified using our approach) that the size and shape of the contest box was constant, no matter where it appeared on the ballot.[4] For Proposition 98, we use the size of the contest box to locate it on the page, but it may be useful to examine other information as well, such as voting target positions, OCR'd text, or other pattern-matching techniques. Finally, given the coordinates of the contest of interest, we crop the image and identify the voting targets.

Initial classification. To classify the ballot images, we calculated the average pixel intensity within each voting target region, and applied a threshold to classify that voting target as filled or empty. A histogram of the average pixel intensities is shown in Figure 15. The threshold we used in our experiments was the halfway-point between labels B and C.

We initially categorized each Proposition 98 contest into a predicted category: *yes*, *no*, *undervote*, or *overvote*. The classification here is not meant to be final, as it will be verified later. We use the classifications to create overlay images of similarly voted ballots for the user to more efficiently verify.

[4]In our case study, Proposition 98 appeared in exactly six locations on the ballot, despite there being many more ballot types (784 as counted by unique barcodes); experimentally, the box dimensions for Proposition 98 varied from ballot to ballot by 1–2 pixels.

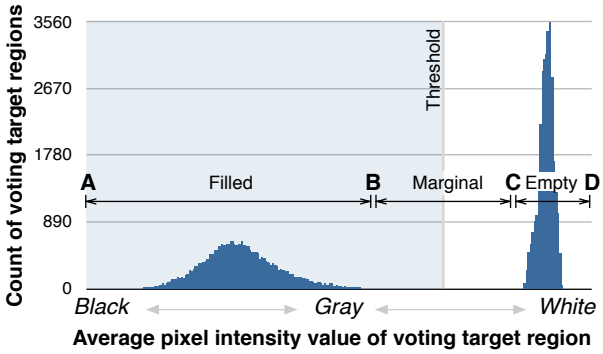

Figure 15: Histogram of average pixel intensity values of 59,898 voting target images taken from Proposition 98 of the June 2008 Primary Election in Humboldt County, California. Despite the apparent separation between modes, 150 (or 0.25%) voting target images fall between labels B and C.

Grouping ballots into verifiable sets. Within each of the four categories, we further grouped the ballots into groups of 200 randomly selected images. The number 200 here was heuristically chosen to balance efficiency and minimize the effect of noise. We did not attempt to optimize the group size for this work. We expect we could gain more efficiency by grouping ballots in a smarter way, for instance by clustering images based on visual similarity, as we discuss later.

After grouping the images into smaller, visually similar groups, our system has all the information it needs to present the results to the user for interactive verification using our overlay image method. In the next section, we evaluate this approach.

4.3 Evaluating verification efficiency

We ran two experiments. First, we counted all ballots using our proposed method and user interface. Second, we compared this to counting the ballots individually. For the latter case, for the sake of time, we visually inspected 1,000 of the 29,949 ballots. We extrapolate from this an estimate of the total time it would take to individually inspect all ballots.

Results counting one by one. To have something to compare our results to, one of the authors examined 1,000 ballot images, one by one. This took 23 minutes. Extrapolating that rate to the full data set means we could have verified the entire set in 11 or 12 hours.

Results using our system. Using our system, the same author visually inspected *all* the Proposition 98 ballot images. We inspected the ballot images in groups of

All ballot images in data set

Before verification \ After verification	Yes	No	Under	Over	Ambiguous	TOTAL
Yes	**11639**	0	0	0	1	**11640**
No	0	**16890**	0	0	6	**16896**
Under	15	18	**1375**	0	0	**1408**
Over	2	0	0	**3**	0	**5**
TOTAL	**11656**	**16908**	**1375**	**3**	**7**	**29949**

All ballot images excluding 150 with marginal marks

Before verification \ After verification	Yes	No	Under	Over	Ambiguous	TOTAL
Yes	**11579**	0	0	0	1	**11580**
No	0	**16824**	0	0	6	**16830**
Under	0	11	**1375**	0	0	**1386**
Over	0	0	0	**3**	0	**3**
TOTAL	**11579**	**16835**	**1375**	**3**	**7**	**29799**

The 150 ballot images with marginal marks

Before verification \ After verification	Yes	No	Under	Over	Ambiguous	TOTAL
Yes	**60**	0	0	0	0	**60**
No	0	**66**	0	0	0	**66**
Under	15	7	**0**	0	0	**22**
Over	2	0	0	**0**	0	**2**
TOTAL	**77**	**73**	**0**	**0**	**0**	**150**

Figure 16: The results of our verification of the Humboldt County Proposition 98 ballot images. The initial tallies from our image analysis stage are compared with the tallies after human verification. The results are further broken down by whether they include marginal marks, as described in Section 1. By verifying all votes visually we are able to more accurately interpret some votes that would otherwise have been miscounted.

at most 200 images at a time, superimposed using the min/max overlay method described in Section 2.3. For every anomaly we discovered, we used the group partition method from Section 3.3 to identify the anomalous ballots and interpret them individually.

We visually inspected all the Proposition 98 ballot images in *1 hour and 23 minutes.* This represents over an 8× improvement in the time it would take to inspect ballots one at a time. We examined 1,326 images in total, representing a 22× improvement over examining the 29,949 ballot images individually. The results of our verification are shown in Figure 16. We acknowledge these are preliminary results—we did not conduct a comprehensive user study—but we argue this is good evidence that our method indeed improves human verification time by an order of magnitude.

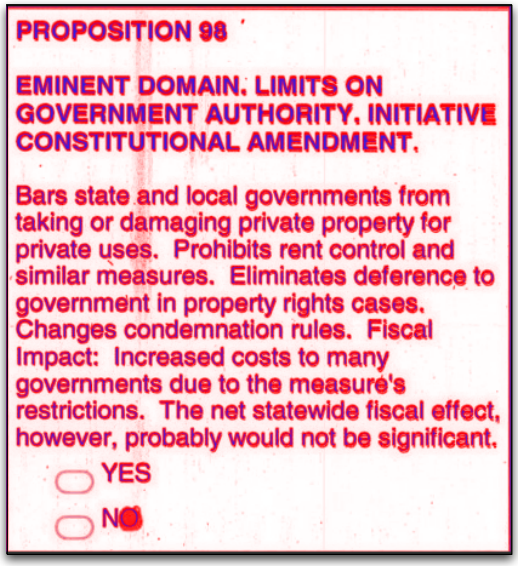

Figure 17: An overlay image that shows at least one anomaly in the "o" in "No."

4.4 Observations

Using our approach, we were able to verify all steps of the image analysis stage: the image registration, the form extraction, and the vote classification. Had a step failed for any one of the ballots, an anomaly would have been evident in its overlay image. In addition, we detected several interesting patterns of anomalies in the ballots.

One curious pattern was a tendency for a number of voters to fill the incorrect oval when voting for the "No" choice. In particular, they seem to have confused the "o" in "No" for the voting target. These ballots were classified as undervotes by our initial classification, and most likely also interpreted as undervotes in the official canvass. Figure 17 shows an example of how this appeared to us in our user interface. This was not an isolated case. Figure 18 shows examples of ten voters who consistently mistook the "o" for the voting target.

Another pattern of anomalies we noticed was the recurring tendency to make corrections by drawing arrows to a voting target then annotating it with either "Yes" or "No" (see Figure 19). The voter's intent in some of these cases is unclear.

Last we also noticed cases where extraneous marks appear to be identifying. In Figure 20 we show two cases where the voter appears to have initialed their changes, which arguably should invalidate their ballot.

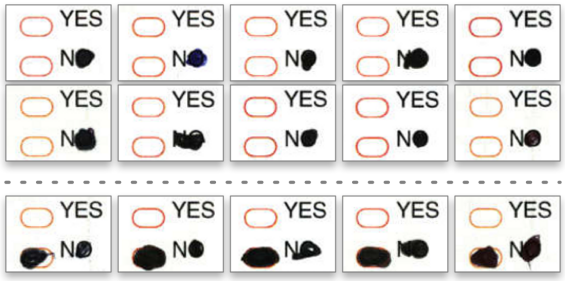

Figure 18: A pattern of anomalies we discovered while visually verifying all ballots. This example shows marks that were clearly intended as "No" votes, where the voter filled in the "o" in "No" rather than filling in the voting target (top). We found evidence that this is a replicable failure of ballot design: the "o" in "No" is mistaken for a voting target by a fraction of the voting public. The top two rows show all 10 examples of this phenomenon we found among the undervotes in this race. Some voters even filled in both the "o" in "No" as well as the voting target (bottom).

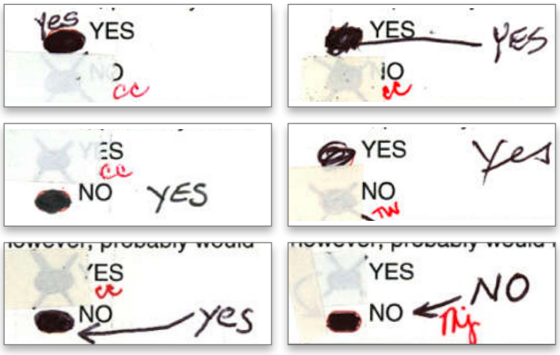

Figure 19: Examples of potentially ambiguous votes we found in our data set. In each example, the voter appears to be overriding or correcting his or her first mark. However, the interpretation of votes remains highly subjective. The initials in red (CC, TW, and MJ) are from voting officials who judged the voter intent and *enhanced* the ballots by covering some regions with tape. In each example, both voting targets are filled and one is crossed out; during the enhancement process, one of these targets was covered with tape and thus is only faintly visible in the images above. Because enhancement was performed before the ballots were scanned, we have only the scans of enhanced ballots, not the original ballots.

5 Discussion

In this section we discuss some of the results and consequences of our approach to verification.

5.1 Trust in ballot images

We acknowledge that our approach still requires trust in the ballot images, and in the software displaying them [15]. Other techniques must be used to gain confidence that the set of ballot images accurately reflects the paper ballots as they were marked by voters. Our approach is designed to minimize the complexity of the software required for verification, and is designed to be intuitive enough for a user to comprehend what is going on. In particular, the minimization and maximization operators were chosen to be simple operations for a machine, as well as simple for a person to reason about. While the software in the image analysis stage—the stage that determines the structure of the ballot images and performs the initial vote classifications—may be complicated, the software necessary to verify a set of images is separable, and can be kept simple. In this way, our system strives to be transparent, a necessity when approaching the problem of verifying elections.

5.2 Applications

Our system lets a person efficiently interpret a large set of ballot images. As such, it can be used to detect and correct many common failure modes under typical-case assumptions,[5] such as mark-sense failures—failures of machines to correctly recognize marks made by the voter—and configuration failures—failures of machines to interpret the detected marks correctly (see Section 1).

The primary benefit of our approach is that it greatly increases the number of ballots an individual can reasonably inspect for him or herself, enabling each interested individual to arrive at his or her own interpretation of an entire contest. This can lead to interesting possibilities. For instance, because the output of our verification scheme is a classification—*from the user*—of every individual ballot, if multiple people examine the same contest, their classifications of individual ballots could be compared. Ballots for which the users' interpretations disagree could be easily discovered.

Another application of our approach may be to enable researchers to efficiently establish ground truth for evaluating novel opscan systems. For instance, researchers studying the accuracy of a new voter-intent classification scheme, or a stray-mark detector, could use our approach to test their system on an order-of-magnitude more human-verified ballot images than before.

[5] We make a distinction between *typical-case* assumptions in which the optical scan system may fail non-maliciously, for instance as a result of operator or programmer error or imperfect algorithms, and *worst-case* assumptions in which the scanning equipment and operators (among other stakeholders in the election) may act maliciously.

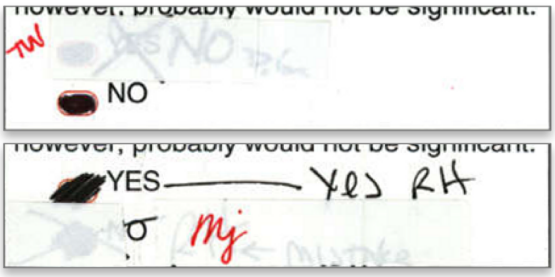

Figure 20: Examples of potentially identifying initials we discovered while verifying the election. The initials in black (DG, faintly visible on top after ballot enhancement, and RH on the bottom) appear to come from the voter. The initials in red (TW on top and MJ on the bottom) are from officials who judged voter intent.

Figure 21: Two example anomalous ballots we discovered with our overlay approach that are *not* enhanced by voting officials, indicating that they went undetected. A potential explanation for this is that the opscan system detected these as "Yes" votes—rather than overvotes—triggering no further review of the ballots. (In our evaluation both votes were subjectively labeled "Yes" votes, after initially being classified as overvotes.)

Our approach may also make it feasible to do post-election logic and accuracy tests of current optical scan equipment by analyzing the equipment's interpretation of ballot images. Today, logic and accuracy testing typically uses a relatively small sample of test ballots, and test ballots are often marked in an unrealistically accurate fashion (e.g., test ballots may come pre-marked by the ballot printer, saving time but failing to anticipate and exhaust all the ways a voter will interact with a ballot). It would be possible for an election official to use our technique to verify the opscan system's interpretation of all ballot images, after the election, and use this to detect configuration errors and assess the system's accuracy. This may provide a better foundation for post-election logic and accuracy testing.

5.3 Limitations

Our approach is based on verifying ballot *images*, and is thus limited by the accuracy of this representation. In particular, our approach requires us to trust that the images are faithful and authentic scans of the paper ballots. As such, *we do not suggest our approach as a replacement for manual audits of paper ballots.* While one can envision auditing the scanning process to establish stronger trust in the images, we consider this an orthogonal issue and leave it as future work.

Working with ballot images poses other limitations as well. For instance, artifacts like ink bleed-through or damage to the paper ballot may be more difficult to recognize or diagnose from a scanned image than from the original paper ballot. In these cases, examining the physical paper ballot would remain important.

Some types of contests also pose issues for our approach. For example, a "vote-for-N" contest, in which the voter is allowed to mark more than one choice, complicates grouping ballots into a small number of visually similar groups that can be verified simultaneously.[6] While our system would still yield accurate results, its efficiency would be affected.

Similarly, issues such as ballot rotations—in which the order of choices may vary ballot by ballot—also complicate our approach. In this case, it is no longer safe to assume the position of a voting target corresponds to a particular choice or candidate, so ballots must be grouped taking into account the particular voting arrangement. However, because groupings are verified visually, this again only results in a loss of speed, and not correctness.

A specific limitation of our evaluation is that the ballots in our data set were *enhanced* by voting officials prior to scanning. In other words, many anomalous ballots discovered by the officials (e.g., overvotes rejected by the scanner) have been corrected either with felt pen or tape. While this prevented us from working with the original anomalies in those cases, we note that we can easily detect enhancements due to the initials that accompany them,[7] and our data set also contains anomalies that went undetected—or at least were not enhanced—by voting officials (see Figure 21).

Another specific limitation of our evaluation is that the number of images given to us did not exactly match the number of ballots cast in the official canvass [8]. The official canvass included 74 additional ballots not in our data set. This limited our ability to compare our human-verified results with the results of the official canvass.

5.4 Future work and open problems

It is an interesting open problem to further reduce the time needed to verify a set of ballot images, without sac-

[6] In the worst case there can be $n!$ combinations, where n is the number of candidates (though this is bounded by the total number of ballots).

[7] Enhancements are initialed by the official making the correction.

rificing accuracy. One possible improvement would be to implement a *clustering algorithm* when selecting ballots to overlay. Instead of placing ballots into groups randomly, we could cluster these images into groups of visually similar ballots. The hope is that if all ballots in a group are largely consistent in marking, it will be easier to verify the group of ballots, and it will be possible to increase the number of ballots in each group. In practice, we suspect the number of anomalous ballots will be small, and consistently marked ballots large, enabling the user to step through large groups quickly.

While this paper focuses on verifying a set of ballot images, another area for future improvement is in the initial image analysis stage of our system. For example, it may be possible to automatically predict which ballots will need to be treated individually because they contain interesting anomalies. Similarly, machine learning techniques could be applied to better classify marginal or otherwise interesting marks. It is possible that analyzing the ballot images more thoroughly in advance may enable verification to proceed more efficiently.

6 Related work

Others have proposed using scanned images of ballots to analyze and independently recount ballots. The Humboldt County Election Transparency Project rescanned all ballots cast in Humboldt County in several elections using a commercial off-the-shelf document scanner [7]. Our evaluation relies upon a collection of ballot images scanned by the Humboldt County Election Transparency Project. We are indebted to them for making this data set available to us.

Mitch Trachtenberg developed BallotBrowser, open-source software to independently recount all contests by analyzing these ballot images. BallotBrowser introduces a novel technique to help the user quickly examine and verify BallotBrowser's interpretation of many marks, by displaying a cropped version of the voting target from many ballots simultaneously on the screen. This lets BallotBrowser display many marks on the screen at once, exploiting human parallelism. However, the user must still visually examine each such mark to gain confidence that all were properly interpreted. We superimpose ballot images to reduce the number of images that the user must examine. Like other optical scan systems, BallotBrowser relies on configuration information that tells it where on the ballot to look for voter marks and how to interpret each mark.

VotoScope is an early open-source software project for analyzing a collection of ballot images from an optical scan system to provide an independent tally of the ballots [9]. Like other optical scan systems, both VotoScope and BallotBrowser are susceptible to mark-sensing and configuration failures. We were inspired and motivated by VotoScope and BallotBrowser. Our work is complementary to their work, as our techniques could be used to efficiently verify the results produced by their systems. In some cases, if the official optical scan system can export the set of all ballot images as well as its interpretation of those ballot images, our techniques could be applied directly to the output of the official opscan system, thus potentially taking the place of VotoScope or BallotBrowser.

Recently, the PERFECT project has pioneered the application of document analysis techniques to automated analysis and interpretation of optical scan ballots [20, 17, 13]. Our work is distinguished from theirs largely by our focus on interactive tools (as opposed to entirely automated algorithms for interpreting ballot images) as well as our focus on verifying the accuracy of a set of ballot interpretations presented to us (rather than building algorithms to produce those interpretations in the first place).

Our work was also inspired by research on interactive computer vision [11, 18], which solves hard computer vision problems by letting the user of a tool give interactive hints to the algorithm. We take a similar approach: instead of attempting to perfectly infer the correct interpretation of every single ballot, we allow the user to interactively guide his or her examination of ballot images through the partitioning operations.

7 Conclusion

This paper introduces techniques to help a human verify the interpretation of a large collection of ballot images, allowing the user to confirm that all ballots were interpreted accurately and to identify anomalies and ambiguous cases. While our approach requires trust in the ballot images and in the software displaying them, it has the benefit of significantly reducing the cost of recounts.

Our experiments suggest that the approach has promise. We found that our approach provides an order-of-magnitude improvement in speed, compared to a ballot-by-ballot recount. We also demonstrated that our approach is able to find a variety of anomalous ballots, many of which would have been undetected and possibly misinterpreted by an optical scan machine. These results suggest that our approach is practical and has the potential to improve the accuracy and transparency of elections.

Acknowledgments

We thank Deirdre Mulligan, Jitendra Malik, Doug Tygar, Adrian Mettler, Peter Bodík, Blaine Nelson, Matt

Piotrowski and the anonymous reviewers for helpful comments. This work was supported by the National Science Foundation under grant CNS-0524745. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Bush v. Gore. 531 U.S. 98, 2000. http://supreme.justia.com/us/531/98/case.html.

[2] Debra Bowen. California Secretary of State Debra Bowen's Report to the Election Assistance Commission Concerning Errors and Deficiencies in Diebold/Premier GEMS Version 1.18.19, March 2009.

[3] Jiun-Lin Chen and Hsi-Jian Lee. An efficient algorithm for form structure extraction using strip projection. *Pattern Recognition*, September 1998.

[4] Peter Fimrite. Surfer-mayor wannabe apparently wipes out; Judge disqualifies write-in votes with bubbles unshaded. *San Francisco Chronicle*, November 2004. http://www.sfgate.com/cgi-bin/article.cgi?f=/c/a/2004/11/23/BAGM9A02VA1.DTL.

[5] Curtis Gilbert. Coleman Concedes to Franken in MN Senate Race. *Minnesota Public Radio*, December 2008.

[6] Joseph Lorenzo Hall, Philip B. Stark, Luke W. Miratrix, Melvin Briones, Elaine Ginnold, Freddie Oakley, Martin Peaden, Gail Pellerin, Tom Stanionis, and Tricia Webber. Implementing Risk-Limiting Post-Election Audits in California. In *Electronic Voting Technology (EVT'09)*. USENIX Association, 2009. http://www.usenix.org/event/evtwote09/tech/full_papers/hall.pdf.

[7] Humboldt County Election Transparency Project. http://humtp.com.

[8] Humboldt County Elections and Voter Registration. Elections and Voter Registration, June 2008. http://co.humboldt.ca.us/election/content/content.asp?pg=election_results.htm.

[9] Harri Hursti. Votoscope software, October 2005. http://vote.nist.gov/comment_harri_hursti.pdf.

[10] Douglas W. Jones. *Towards Trustworthy Elections*, volume 6000/2010 of *Lecture Notes in Computer Science*, chapter On Optical Mark-Sense Scanning, pages 175–190. Springer Berlin / Heidelberg, 2010.

[11] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. *ACM Transactions on Graphics*, 23(3), 2004.

[12] S.P. Lloyd. Least-Square Quantization in PCM. In *IEEE Transactions on Information Theory*, volume 28, pages 129–137, 1982.

[13] Daniel Lopresti, George Nagy, and Elisa Barney Smith. Document Analysis Issues in Reading Optical Scan Ballots. In *DAS '10: Proceedings of the 8th IAPR International Workshop on Document Analysis Systems*, pages 105–112, New York, NY, USA, 2010. ACM.

[14] Perry Bacon Jr. Coleman Concedes to Franken in MN Senate Race. *The Washington Post*, June 2009. http://voices.washingtonpost.com/capitol-briefing/2009/06/minn_supreme_court_declares_fr.html.

[15] Eric Rescorla. Understanding the security properties of ballot-based verification techniques. In *Electronic Voting Technology (EVT'09)*. USENIX Association, 2009. http://www.usenix.org/event/evtwote09/tech/full_papers/rescorla-ballot.pdf.

[16] Tim Rohwer. Faulty Voting Machines Delay Results; Counting Under Way. *The Daily Nonpareil Online*, June 2006.

[17] Elisa H. Barney Smith, Daniel Lopresti, and George Nagy. Ballot Mark Detection. *19th International Conference on Pattern Recognition*, 2008.

[18] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. *ACM Transactions on Graphics*, 23(3):315–321, 2004.

[19] Verified Voting Foundation. The Verifier, 2008. http://www.verifiedvoting.org/verifier.

[20] Pingping Xiu, Daniel Lopresti, Henry Baird, George Nagy, and Elisa Barney Smith. Style-Based Ballot Mark Recognition, 2009.

[21] Kim Zetter. Voting System Adds Nearly 5,000 Ballots to Tally, June 2009. http://www.wired.com/threatlevel/2009/06/voting-machine-adds-nearly-5000-ballots-to-tally.

OVERLAY-IMAGE($min\_image, max\_image, min\_color, max\_color$)

1 $overlay\_image \leftarrow$ new $I \times J$ color image
2 **for** $j \leftarrow 0$ **to** $J - 1$
3 **do for** $i \leftarrow 0$ **to** $I - 1$
4 **do** $\alpha \leftarrow (max\_image_{i,j} - min\_image_{i,j}) \,/\, (255 - min\_image_{i,j})^{\dagger}$
5 $hue \leftarrow \alpha \cdot min\_color + (1 - \alpha) \cdot max\_color$
6 $\beta \leftarrow min\_image_{i,j} \,/\, 255$
7 $overlay\_image_{i,j} \leftarrow \beta \cdot \mathit{white} + (1 - \beta) \cdot hue$
8 **return** $overlay\_image$

Figure 22: Pseudocode for an alternative approach to create an overlay image from a pair of min- and max-images. Pixels in the resulting overlay image will vary in hue between *max-color* (i.e., blue in our examples) and *min-color* (i.e., red) depending on how far apart the darkest and lightest pixels are at a particular location. The whiteness of an overlay pixel depends on the intensity of the darkest pixel at a particular location. †In the case that $min\_image_{i,j} = max\_image_{i,j} = 255$, let $\alpha = 0$ (hue drops out anyway on line 7).

Appendix A Alternate overlay algorithm

Two requirements for an overlay image are that it capture the intensity of the min- and max-images, and that they be differentiable from one another. When we plot the spectrum of possible colors that result from our proposed algorithm we get the graph on the left in Figure 23. In particular, note the three corners of the spectrum that denote the common cases: the bottom-left corner is blue, corresponding to a region of foreground where all the pixels in the set are black; the top-right corner is white, corresponding to a background region where all pixels are white; and the top-left corner is red, corresponding to the case where at least one pixel is black and another is white, commonly because of a stray mark.

Another property of this spectrum to consider, however, is the interpolation of colors between corners. In particular, consider the edges. Ideally, along the top edge, we would like a smooth interpolation between red and white, which corresponds to regions that are white in at least one ballot, but dark in another—likely stray marks. The gradient on the diagonal, on the other hand, corresponds to regions that are identical in all ballots, even if they vary in shade between black and white. In this case, we would like a smooth interpolation between blue and white.

Our original overlay algorithm does not vary smoothly from blue to white along the diagonal, resulting in a more *conservative* overlay image: images that are possibly identical may be perceived as containing more variation than present, requiring more work to verify.[8] To address this shortcoming, we have designed an alternative algorithm (see Figure 22) to ensure that the diagonal varies smoothly from blue to white, with no red, as we see in the graph on the right in Figure 23. Our experiments in Section 4 used the algorithm in Figure 8, but in hindsight the algorithm in Figure 22 may have been a better choice.

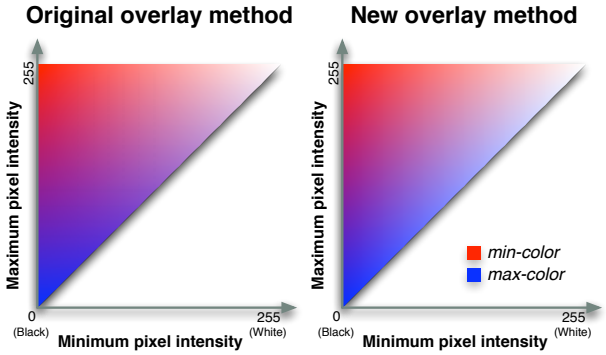

Figure 23: A comparison of the resulting range of colors a pixel may take, as produced by our two proposed OVERLAY-IMAGE algorithms. The image on the left is produced by the algorithm in Figure 8. The image on the right is produced by the algorithm in Figure 22. Note the differences in color, particularly along the diagonal: the left image shows a gradient from blue to purple to white, while the right image shows a gradient from blue to white.

[8] The difference between both algorithms is most evident on images with a lot of gray; in practice, though, both algorithms resulted in similar looking overlay images as our ballots contained mainly black and white features.