Placing QR Codes on Tactile Graphics

Fantactile Group: Catherine Baker, Lauren Milne and Kyle Rector

Abstract:
In the automated process described by Jayant et al. that converts textbook images to tactile graphics, the step that required the most human time was placing text labels on the tactile images [3]. In order to reduce the amount of human labor needed, we used two algorithms, simulated annealing and greedy, and tried various evaluation functions to place QR codes on tactile graphics. We found that while the two algorithms did not seem to produce different results, all three evaluation functions resulted in fewer QR codes that needed to be moved in the images. In addition, the user study showed that people preferred the output from the evaluation functions over the original QR label placement.

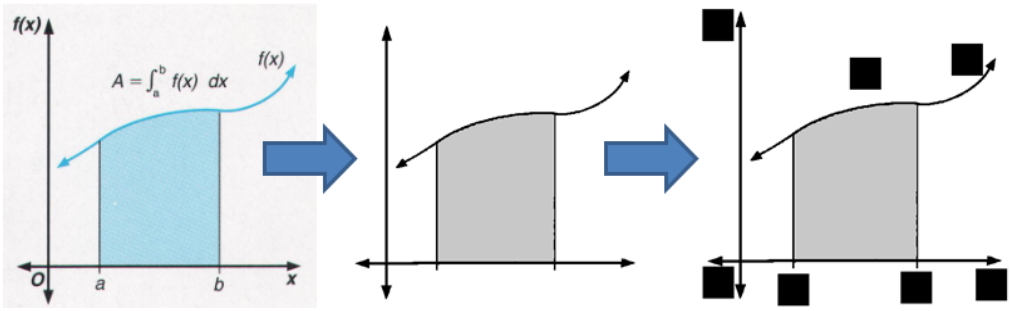
Introduction and Related Work:

Although the text in most books is now accessible because of screen readers and Braille displays, the images in books remain largely inaccessible. This is an especially big problem with textbooks in the Science, Technology, Engineering and Math (STEM) fields, as the textbooks typically have hundreds of images that are crucial for understanding the text. One way to make the textbooks accessible is to provide tactile graphics, where parts of the image are raised so it can be understood by touch. Unfortunately, converting all of the images by hand is generally an expensive and laborious process. In order to address this problem, Jayant et al. [3] created the Tactile Graphics Assistant, which helped automate the translation process by: 1) cleaning up the image, 2) identifying and removing the text, 3) resizing the image 4) translating the text into Braille, and 5) replacing the text with Braille in the new tactile image. Step 3 is important to account for the different resolution needed for tactile images. Our work is to replace steps 4 and 5 with the use of QR codes which read a text label aloud on a smartphone instead of Braille. There are multiple issues when replacing text with Braille; it takes up more space, and many people who are blind do not read Braille.

Jayant et al. found that most time consuming of the non-automated tasks was editing the placement of the Braille in the newly resized tactile images [3], therefore we are interested in using artificial intelligence to automate the process and reduce the amount of human editing needed. The authors mentioned that they tried an algorithm with a linear evaluation function to assist in label placement, but did not provide any results. Christensen et al. [1] used simulated annealing to solve a similar problem: placing text labels on maps. Because of this previous work, we decided to use both simulated annealing and a greedy algorithm to place the QR codes. We also compared output figures using three different evaluation functions. We wanted to answer the following research questions:

1 Is the amount of human labor needed to adjust the QR labels decreased after running the algorithms?
2 Did people prefer the algorithmic output over the original QR label placement?

Typical Use Case Scenario



a **b** **c**

Figure 1. This shows the steps required for a typical user: a) find an original textbook figure, b) remove the text labels using the Tactile Graphics Assistant [3], c) place QR labels on the graph using the greedy algorithm.

A typical user of our project would be a person who is interested in making a textbook figure accessible to someone who is blind. They would begin with a raw figure that they wish to convert (see Figure 1a). Using the Tactile Graphics Assistant [3], the user can remove the text labels from the image in preparation for placement of new labels (see Figure 1b). Because this step was outside our project scope, we will not have any further discussion. Finally, the image and text label data are used in conjunction with the greedy algorithm to determine the proper QR label placement (see Figure 1c). Ideally, we would like to use QR codes instead of a placeholder, but that is left for future work.

System Design:

The algorithms we used to place the QR codes were written in python. As input they took 1) a resized image from which the text had been removed, 2) a text file containing all the text that had been removed and 3) an xml file that had information about the text (including alignment and original placement) and both the x and y scale factors for the resizing of the image. Our code currently outputs image with black boxes in place of the actual QR codes. We ensured that the black boxes were the correct size for the amount of text in the QR code and had a buffer of forty pixels to give a white border around the QR code, so it could be read. Because we are using these boxes instead of QR codes, we will refer to them as QR labels for the rest of this paper.

The initial placement of the QR labels in the resized images was calculated by multiplying the x and y coordinates of the original image by the x and y scale factors. As the dimensions of the QR labels were significantly different than those of the original text, which x and y coordinates we used depended on the alignment of the original text: if the text was left-aligned, we used the top left corner, if it was right-aligned, we used the top right corner and if it

was centered, we used the top center. If any part of the QR label was initially placed off of the image, we moved it in until its edge was along the outside of the image. We used this initial placement as a base case for comparing the output generated by our algorithms.

For the greedy algorithm we used, we placed all the QR labels in a priority queue ranked by their evaluation function. We removed the worst-placed QR label (the one which had the highest evaluation score) from the queue and evaluated moving it in eight directions (up, down, left, right and along the diagonals) by ten pixels. Along the diagonals, we moved the label 10 pixels along both the x and y axes. We picked the direction that improved the evaluation function of that QR label by the most. If movement only worsened the evaluation function, the QR label was moved to the back of the priority queue and reevaluated only after another QR label was moved. To prevent thrashing, we also limited the number of times a QR label could be moved to 50.

For simulated annealing, we randomly picked a QR label and direction to move it. If the movement resulted in an improvement or no change in the evaluation function, we always accepted it. If not, we accepted it with some probability dependent on the temperature and how much worse the change made the image. The temperature decreased in stages.

We wrote a number of functions to evaluate the placement of a QR label. The functions were all linear equations of the form: $\sum_{features} W_i * f_i$, where W_i is the weight of each feature and f_i are features that we identified as possibly being important in the evaluation function. The features we considered were: distance from original placement (along both the x and y axes), overlap over other QR labels, and overlap over the image. In determining the image overlap we tried two functions: one computed the number of non-background image pixels that were overlapped, and the other computed the number of non-background image pixels weighted by the portion of the QR label that overlapped. Pixels near the center of the QR label had a higher weight than those on the outside. An example case where this would help is where the center of a QR label is overlapping the image. Moving the QR label 10 pixels may provide no improvement, but moving the QR label 20 pixels may improve the overall score. The three evaluation functions that we used for the formative user study focused on different features. Catie's evaluation function gave more weight to QR labels that overlapped the image, making it less likely that the QR labels would occlude the image. Lauren's evaluation function considered alignment of text and gave more weight to moving originally left or right aligned text to the left or right as opposed to up and down. Kyle's function gave more weight to QR labels that overlapped other QR labels.

Evaluation Setup:

This section explains the method used for selecting an algorithm and obtaining images for the study that could be used in a real world setting. We also show how these images were used in our tasks.

Choosing the Algorithm

We chose to evaluate the results of our different evaluation functions using the greedy algorithm. We choose to work with this algorithm over simulated annealing for two reasons: time and better QR label placement. Simulated annealing took significantly longer to run without much difference in results. It typically ran for 2-3 minutes versus the 30 seconds needed for greedy. Simulated annealing also had QR label placement issues, which was surprising because there were positive results with city placements on a map[1]. Unlike the placement of labels on a map, our algorithms did not have any hard constraints to keep the QR labels directly adjacent to the object to which they refer. With some probability the QR label would make a bad move, which could lead to wandering. This caused some QR labels that would be fine in their original placement to move to a new bad or awkward placement. Because we want to make sure we are not making the placement worse, we choose not to proceed further using simulated annealing.

Determining Math Figures for the Study

Our images were acquired from a Precalculus textbook [2] previously used for the Tactile Graphics Assistant. The images had already been digitized, and classified into different categories. We wanted to make sure we evaluated our function on a variety of types of images to make sure that we were not overfitting our evaluation functions to one type of image. Therefore we selected our images from two data sets, complex images which had a wide variety of images that were considered harder to place the QR labels on and clean lines which contained mostly graphs, a very common image type in the textbook. In order to get a representative sample, we selected every third image from these two sets. After using these two data sets, we had 27 figures to use in our evaluation.

Task Format

For each of the 27 figures, we created a task sheet for participants to reference. We placed the original image from the textbook at the top of the page. This image included the English text so the participants could tell what text each QR label contained. Below the original were the four images containing different QR label placements, one for each evaluation function as well as one for the original placement of the QR labels. Since determining the QR labels is left for future work, we used boxes as placeholders. The four images were placed in a randomized order and given labels A, B, C, and D so participants were unaware which image went with which placement type to prevent bias (see Figure 2).

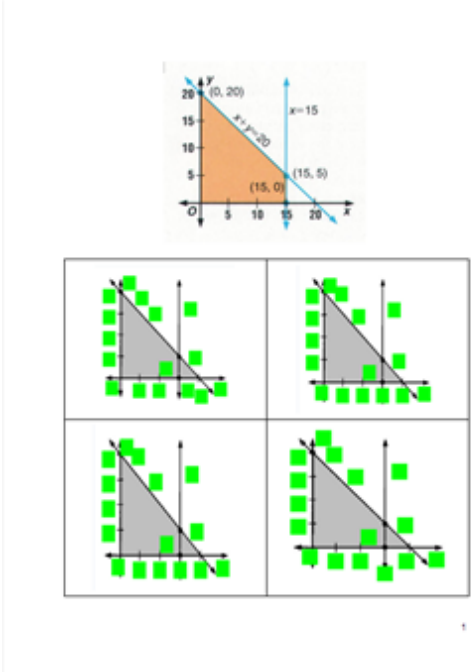


Figure 2. Sample task shown to participants. The output images are a random ordering of original figure with QR labels, and the output from three evaluation functions.

Evaluation Study:

In order to determine whether the above method decreases the amount of manual labor from a potential user, we conducted a small informal lab study with five participants. We used their feedback to assess the effectiveness of our three evaluation functions. In this section, we describe our study procedure.

Participants

We ran a within subjects quantitative study with five people. We recruited four non-experts and one expert to participate in our study. The participants evaluated the four different possible outputs for the 27 images using our tasks described above. We asked the participants for two types of information:

1 The number of QR labels they felt should be moved for the four output images
2 The rank of their preferences between the four images

The participants were told to use their judgment on whether a QR label should be moved. We gave them examples for moving a QR label such as the QR label was nowhere close to what it was referring to or if it unnecessarily overlapped the image.

Results:

Here we discuss the results of our evaluation study, specifically addressing each of our research questions. Because we did an informal study with only five users, we did not perform statistical analysis on our results.

1. Is the amount of human labor needed to adjust the QR labels decreased after running the algorithm?

In order to determine the amount of saved human labor, we used two different metrics. The first metric we looked at was what percentage of images required fewer QR labels to be moved than the original placement. Ideally, we would like to have every image require fewer QR labels to be moved or at least none to require more labels to be moved. We found that based on the expert evaluation, all the images had the same or fewer number of QR labels that needed to be moved and 77.7% of the time the images required fewer QR labels to be moved. The non-expert evaluations were also good, with approximately 80% of the images requiring the same or fewer movements than the original placements.

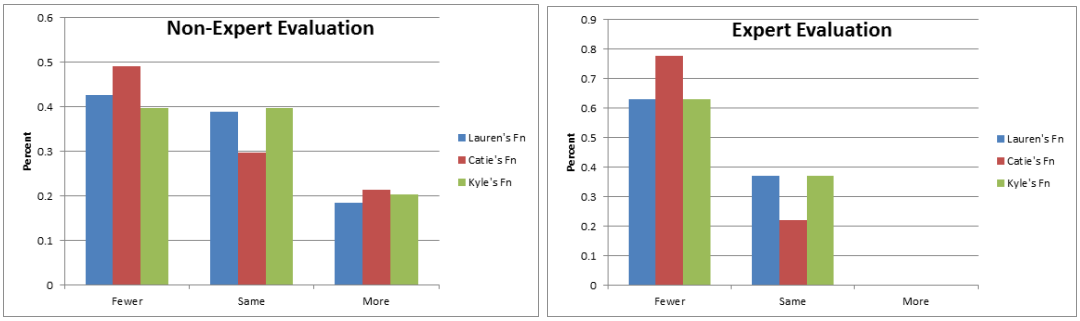


Figure 3. The percentage of times that the participants said that the image required more, equal number, or fewer QR labels to be moved when compared to the number of QR labels that need to be moved from their original placements based on both expert and non-experts opinions.

Once we knew that the majority of the images did not require any extra QR label moves, we wanted to see what the effect was on the total amount of work that needed to be done. We want to avoid an overall increase in work; the work reduced for most images could be less than the extra work needed for a few images. Therefore, we used the metric of average number of QR labels that need to be moved per image. We found that the total amount of work also decreased when compared to the original placement (see Figure 4). When considering all the images, the number of QR labels that needed to be moved decreased slightly, but the decrease was much more dramatic when looking just at the images deemed solvable (unsolvable images explained below). If we look at the best results of the evaluation functions, we see that for the non-experts, the number of QR labels that need to be moved was cut in half from .9 to .4 per image. For the expert, the difference was even more dramatic. The the number of QR labels that need to be moved went from 1.9 to .3 per image.

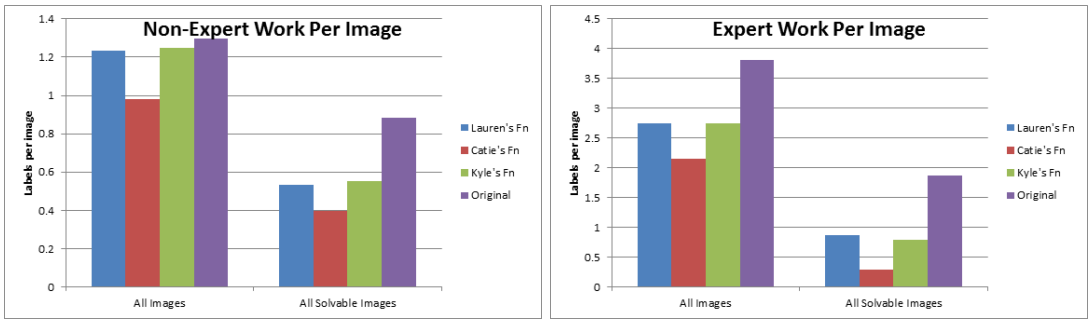


Figure 4. This displays the average number of QR labels that needed to be moved per image based on both expert and non-experts opinions.

Unsolvable Images:

There were some cases where the best possible placement was still not good. For example in Figure 5, the QR labels for the vertical axis do not fit in the space between the vertical axis and the left edge of the image. There are also too many QR labels to fit along the horizontal axis, so the QR labels must overlap each other. We classified these images as 'unsolvable', meaning that in order to properly place the QR labels on these images we would have to remove some of them. Because we felt there was still valuable information to be learned from them, these images were included in all of our results except when we separated out solvable images in Figure 4.

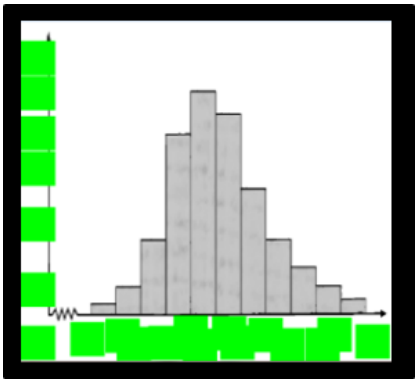


Figure 5. This is an example of an unsolvable image that could not be improved using the algorithm.

2. Did people prefer the algorithmic output over the original QR label placement?

We found that both the non-experts and the expert preferred all three evaluation functions over the original QR label placement. We assessed this by comparing the average rank between the four options (see Figure 6). The non-experts gave the three evaluation functions an average ranking of 1.75, while the original QR label placement had an average ranking of 2.48. The expert had a stronger preference, assigning the evaluation functions with an average ranking of 1.43 and the original QR label placement a ranking of 2.78. Though we cannot state causality, all of the participants preferred the evaluation functions that reduced the amount of labor.

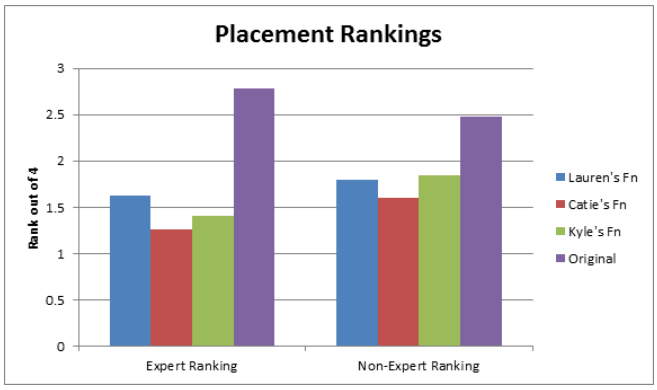


Figure 6. This displays the rank for the expert and non-experts for the original QR label placement, and the three evaluation functions.

Discussion and Limitations:

Although we tested out three evaluation functions that gave different weights to features, one limitation of our study is that we did not tease out the importance of each feature. In the future, it would be interesting to run evaluation functions that only use one feature or that systematically change the relative weights of each feature and compare them via user evaluations. Another limitation is that we used blank squares in the place of QR codes, which meant the users in our study evaluated a different output that the end user would. This meant that users in our study could have underestimated the number of QR labels that needed to be moved, and they had to guess if a QR code was placed in a way that made it readable.

Conclusion and Future Work:

We found that we could decrease the number of QR labels that needed to be moved using our evaluation functions with our greedy algorithm, and that users preferred our placements. In doing this study, we determined a number of future directions to explore. In order to address the problem of unsolvable images, during the resizing step we could include a buffer around the outside of the image to ensure that all QR labels fit on the image. We could automate a way to determine if there are too many QR labels to fit on an axis using information about label alignment and placement. We could then either manually remove some of the labels or attempt to automate that process. As mentioned in the discussion, it would be interesting to do a more in-depth study of the relative importance of each of the features we identified in the study.

References:

[1] Christensen, J., Marks, J., and Shieber, S. 1994. Placing text labels on maps and diagrams. In *Graphics gems IV*, Paul S. Heckbert (Ed.). Academic Press Professional, Inc., San Diego, CA, USA 497-504.

[2] Gordon-Holliday, B., Yunker, L., Vannatta, G., and Crosswhite. F. 1999. *Advanced Mathematical Concepts, Precalculus with Applications*. Glencoe/McGraw-Hill.

[3] Jayant, C., Renzelmann, M., Wen, D., Krisnandi, S., Ladner, R., and Comden, D. 2007. Automated tactile graphics translation: in the field. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility* (Assets '07). ACM, New York, NY, USA, 75-8.

Appendices:

Separation of Work:

Catie wrote the simulated annealing algorithm, her evaluation function, ran the user study and helped write the paper. Lauren wrote the greedy algorithm, her evaluation function, helped with user study and helped write this paper. Kyle did the initial work in figuring out how to work with the data, wrote functions in javascript, wrote the code for parsing the files in python, wrote her evaluation function and helped write the paper.

How to start and use our project:

To run our project:

1) Navigate to the directory containing the project. It should contain the file "greedyEval.py", as well as the "sample" folder containing the input files needed for two images (the text file containg the label text, the bitmap file containing the image cleaned of all text and the xml file that contains information about the initial placement of the text). It also contains a README.txt file with directions.

2) Depending on which evaluation function you choose to use, you will need to create a folder for the output images. The evaluation function is called from the commandline. If you choose 0, the output folder should be called outputKyle. If you choose 1, the output folder should be called outputLauren and if you choose 2, the output folder should be called outputCatie.

3) run with the command:

python greedyEval.py <directory> <0 - Kyle, 1 - Lauren, 2 - Catie> <gr - Greedy, sa - Simulated Annealing>

where <directory> is the folder containing the input image files, <0 - Kyle, 1 - Lauren, 2 - Catie> specifies which evaluation function you would like to use and <gr - Greedy, sa - Simulated Annealing> specifies which algorithm you would like to run.

4) Output will go to the specified folder. The output will consist of a .bmp file that shows where the text boxes were placed before running the algorithm, with a name of the form:

QRfigNAME.Resized.bmp

and a .bmp file that shows where the QR labels were placed after running the algorithm, named either:

QRfigNAME.gr.Resized.bmp

QRfigNAME.sa.Resized.bmp