Automated Grading of Handwritten Problem Sets
CS221 Final Project Proposal

Bruno De Martino, Alex Lin, Austin Ray

Introduction

Homework and problem sets are a staple of any academic environment. For professors and TAs, significant time is dedicated to grading assignments. Our project seeks to streamline this process. We are looking to take scanned PDFs of homeworks as input and, after identifying and processing the students' answers, output a graded version of their homework that the teaching staff can correct and comment on. Today, technology exists in the spaces of OCR and computer vision to identify and parse handwritten text, digits, and symbols. Technology for parsing handwritten equations is slightly more shaky, however, and only seems to be effective when handwriting is done on a tablet. We aim to build on existing technology to create a comprehensive homework processing and grading system.

At the most basic level, we hope to be able to identify and extract boxed numerical solutions from a student's work and to compare them with a pre-determined solution. Depending on the complexity and difficulty of implementing this system, we hope to augment our system to handle equation-form solutions and then written text solutions. Ideally, our project's final product will facilitate the complete conversion from a scan of written homework to a graded, digitized, fully editable and commentable version of that homework. The last part of that sentence is important, as we accept that while our system will handle the majority of the tedious, repetitive work a teacher usually does, it will not be perfect. Accordingly, we want to allow professors and TAs to correct auto-grading mistakes, give partial credit, and add comments after our system is finished. We hope to leverage existing code-style grading software used in CS107/110 to handle this post-system-touch-up task. We hope to make a genuinely useful tool to allow teachers from all levels of schooling to spend less time grading assignments and more time working with their students.

Milestones and I/O Behavior

Given the size of this project, we decided to break it down into several different steps, so we can evaluate our performance during each milestone, and have valuable results even if we are not able to reach our ideal solution. Our primary focus is on homeworks from courses in mathematics. To do most of the learning and testing, we will create our own datasets of handwritten problem solutions by hypothetical students, paired with a digitized solution set. We are also hoping to leverage archives from previous CME, MATH, and CS tests and homeworks, as well as homeworks used by middle school and high school teachers with whom we still keep in touch.

- **Milestone 1)** <u>Basic physical to digital translation:</u> program takes handwritten symbol, and correctly translates it to the equivalent digital version. Given the existing packages we hope to leverage, such as OpenCV, this should be relatively straightforward (**see attachment 1**)
- **Milestone 2)** <u>Finding boxed solution:</u> Given a scanned, student-handwritten math solution (with a boxed answer) and a digital answer (a number), the program is able to correctly identify what the student's solution is, and analyze whether it is right or wrong. (**see attachment 2**)
- **Milestone 3)** <u>Handwritten equation to digital form:</u> Given a handwritten equation, the program is able to correctly identify it, parse the symbols, and construct it in digital form. This is likely the most complex step of our project thus far, as the most advanced packages in this area only handle equations written on digital mediums - tracking stylus strokes is essential to them. (**see attachment 3**)

- **Milestone 4)** Handwritten text to digital form: Given handwritten text, the program is able to translate it to digital form. Given the current packages and the milestones that will have been reached before Milestone 4, this shouldn't be a big challenge. (**see attachment 4**)
- **Milestone 5)** Parsing different sections: Given a handwritten homework with multiple digitized pages, the program is able to differentiate and label different sections as "title", "text", "equation", "final result" for each problem, even those spanning across multiple pages. (**see attachment 5**)
- **Milestone 6)** Putting it all together: Given a multi-page, handwritten homework as input, the program outputs a digital, graded version of the homework. (**see attachment 6**)
- **Milestone 7)** Human intervention: Same as Milestone 6 except the teaching staff can now correct grading mistakes made by our system, grant partial credit, and add comments. Our system would collect data from these interactions and use the data to tune its algorithms. (**see attachment 7**)

Success Metrics

Our program works well if it is able to correctly parse, grade, and digitize solutions from the handwritten homework. We can test the parsing and grading effectiveness of our solution by comparing the output to a hand-typed set of vectors that represent both the parsed elements from that page of homework and the actual solution set from that homework. Training and testing data will be obtained from existing homework submissions by students and the associated solution sets - we will try to obtain these through helpful professors and peers. We plan to supplement this data with simple problem/solution sets that we will manually create for training purposes. Due to the nature of our project, we have slightly more flexible bounds on the success rates of our program - the final product will serve as a tool to help grading happen more quickly, but the grader will be able to review and correct the output. Nonetheless, we will be satisfied if our project reaches a test success rate of 75%.

The baseline algorithm would be a system that 'symbolizes' homework pages by individual elements (characters, boxes, lines) using CV packages and then gives the student a final point value equal to the number of boxes (with other symbols inside of them) it sees divided by the number of problems in the homework. This baseline is obviously wrong, since it gives the student full points for each problem they boxed an answer for (hypothetically). Our oracle algorithm would simply be a human grading the problem sets by hand and typing the graded problem set up in LaTeX. The gap between the baseline and the oracle is primarily in the identification of 'sections' of the problem set (text, titles, equations, answers) and parsing the symbols in these sections into a form comparable with the solution set. Challenges for bridging this gap include identifying boxed solutions as such, separating equations from "regular" text, converting contours discovered by CV packages into ASCII and special equation symbols, and placing them into a coherent digitized format.

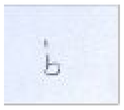
Existing Tools / Implementation Strategy

We are looking to use a variety of existing tools and packages to help implement our solution. Python OpenCV should provide a suite of tools to aid in visual pre-processing of input, separating out individual sections on the page as well as separating out the symbols and characters in each section. SciPy might also be useful for further image processing, and OCR package such as Tesseract could aid in converting symbols to real character objects. We may utilize equation parsers such as MyScript or the SESHAT package.

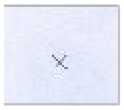
Once the handwritten document is converted into symbols, we can use machine learning algorithms and stochastic gradient descent to classify each section as text, equation, solution, etc., after which we can combine the individual characters recognized into strings and equations. We plan to process equations that exist in 2D space, which might require the implementation of a 2D Context Free Grammar & Machine Learning using SGD. After parsing equations, we can then convert to LaTeX form through sympy.latex(), which will output the equation in a nice, human-readable form.

Attachments:

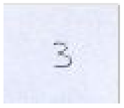
Attachment 1:


b

$\longrightarrow$ b

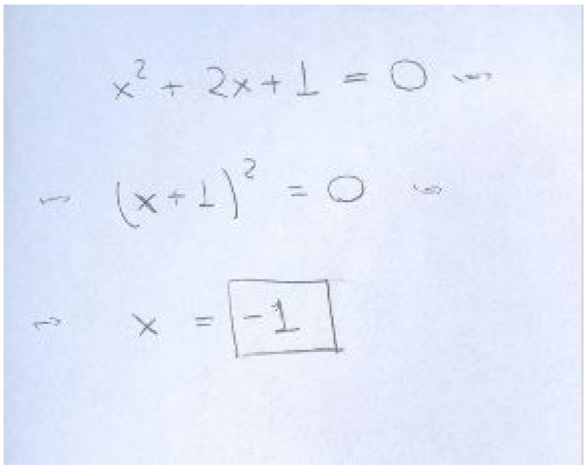

x

$\longrightarrow$ x


3

$\longrightarrow$ 3

Attachment 2:

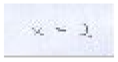

$x^2 + 2x + 1 = 0 \Leftrightarrow$

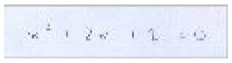
$\Leftrightarrow (x+1)^2 = 0 \Leftrightarrow$

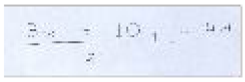
$\Leftrightarrow x = \boxed{-1}$

$\longrightarrow$ Final answer: -1
CORRECT

Attachment 3:

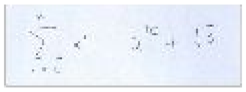

$\longrightarrow$ x = 2 $\longrightarrow$ $x = 2$


$\longrightarrow$ x^2 + 2x + 1= 2 $\longrightarrow$ $x^2 + 2x + 1 = 0$

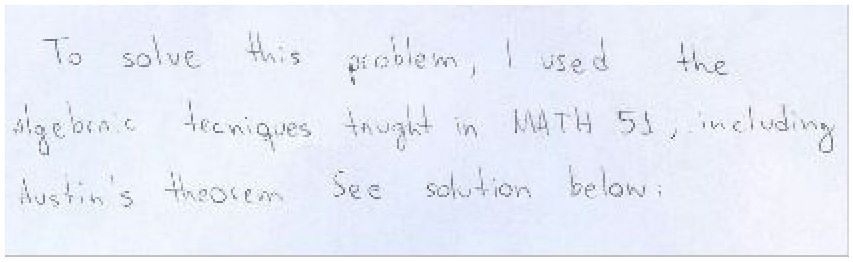

$\longrightarrow$ (3x + 10y)/z = 44 $\longrightarrow$ \dfrac{3x + 10y}{z} = 44

$\downarrow$

$$\frac{3x + 10y}{z} = 44$$

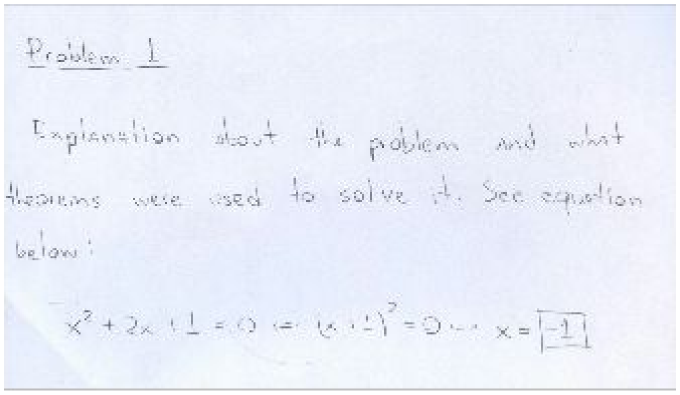

$\longrightarrow$ \sum\limits_{i=1}^n x^i = 2^{10} + \sqrt 3 $\longrightarrow$ $\sum\limits_{i=1}^{n} x^i = 2^{10} + \sqrt{3}$

Attachment 4:


To solve this problem, I used the algebraic tecniques taught in MATH 51, including Austin's theorem See solution below:

To solve this problem, I used the algebraic techniques taught in MATH 51, including Austin's theorem. See solution below:

Attachment 5:


Problem 1

Explanation about the problem and what theorems were used to solve it. See equation below:

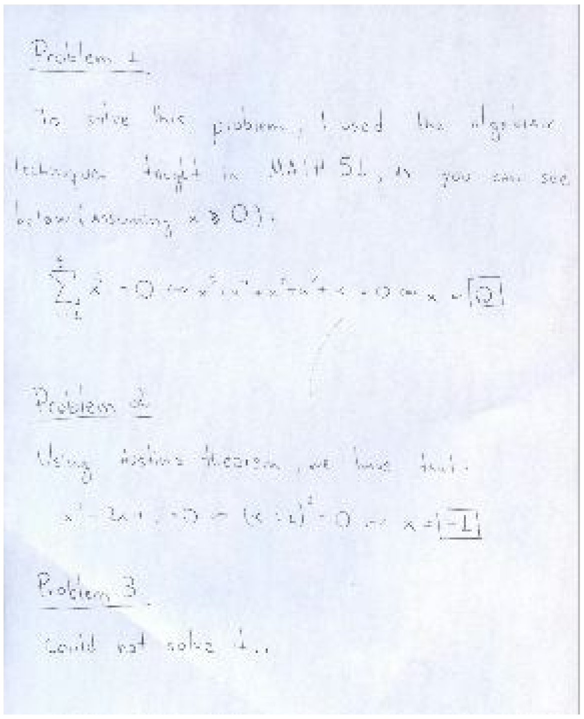
$$x^2 + 2x + 1 = 0 \Leftrightarrow (x+1)^2 = 0 \cdots x = \boxed{-1}$$

Section: "Problem one"

Text: "Explanation about the problem and what theorems were used to solve it. See equation below:"

Equation: "x^2 + 2x + 1 = 0 \Leftrightarrow (x+1)^2 = 0 \Leftrightarrow x = -1"

Attachment 6:



Problem 1

To solve this problem, I used the algebraic techniques taught in MATH 51, as you can see below (assuming $x \geqslant 0$):

$$\sum_{i=1}^{5} x^i = 0 \Leftrightarrow x^5+x^4+x^3+x^2+x = 0 \Leftrightarrow x = \boxed{0}$$

Problem 2

Using Austin's theorem, we have that:

$$x^2+2x+1=0 \Leftrightarrow (x+1)^2 = 0 \Leftrightarrow x = \boxed{-1}$$

Problem 3

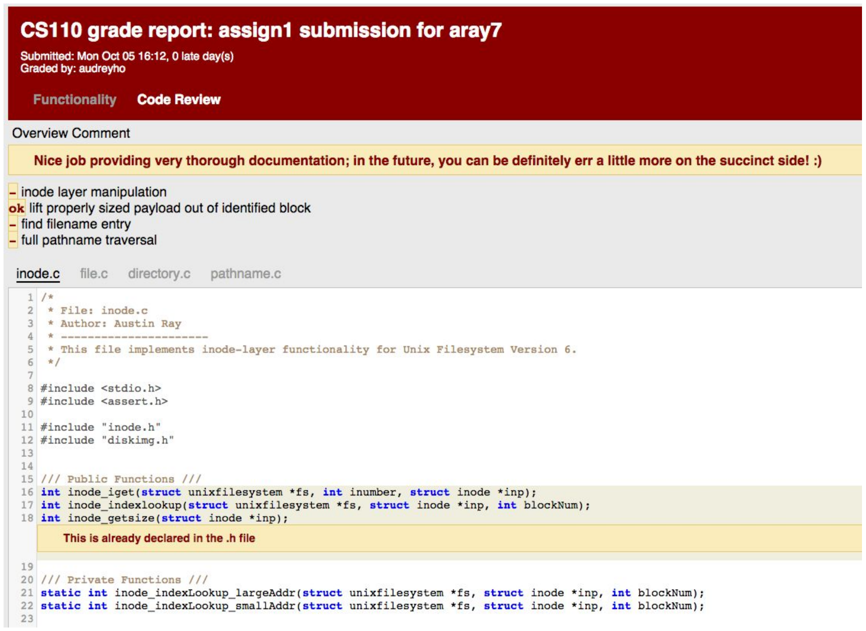
Could not solve it..

Section: "Problem 1"
Text: "To solve this problem, I used the algebraic techniques taught in MATH 51, as you can below (assuming x \geqslant 0):"
Equation: "\sum\limits_{i=1}^5 x^i = 0 \Leftrightarrow x^5 + x^4 + x^3 + x^2 + x = 0 \Leftrightarrow x = 0"
Final answer: 0
CORRECT

Section: "Problem 2"
Text: "Using Austin's theorem, we have that:"
Equation: "x^2 + 2x + 1 = 0 \Leftrightarrow (x+1)^2 = 0 \Leftrightarrow x = -1"
Final answer: -1
CORRECT

Section: "Problem 3"
Text: "Could not solve it..."
Equation
Final answer:
INCORRECT

Attachment 7:



CS110 grade report: assign1 submission for aray7

Submitted: Mon Oct 05 16:12, 0 late day(s)
Graded by: audreyho

Functionality **Code Review**

Overview Comment

Nice job providing very thorough documentation; in the future, you can be definitely err a little more on the succinct side! :)

- – inode layer manipulation
- ok lift properly sized payload out of identified block
- – find filename entry
- – full pathname traversal

inode.c file.c directory.c pathname.c

```
/*
 * File: inode.c
 * Author: Austin Ray
 * ----------------------
 * This file implements inode-layer functionality for Unix Filesystem Version 6.
 */

#include <stdio.h>
#include <assert.h>

#include "inode.h"
#include "diskimg.h"


/// Public Functions ///
int inode_iget(struct unixfilesystem *fs, int inumber, struct inode *inp);
int inode_indexlookup(struct unixfilesystem *fs, struct inode *inp, int blockNum);
int inode_getsize(struct inode *inp);
```

This is already declared in the .h file

```

/// Private Functions ///
static int inode_indexLookup_largeAddr(struct unixfilesystem *fs, struct inode *inp, int blockNum);
static int inode_indexLookup_smallAddr(struct unixfilesystem *fs, struct inode *inp, int blockNum);

```