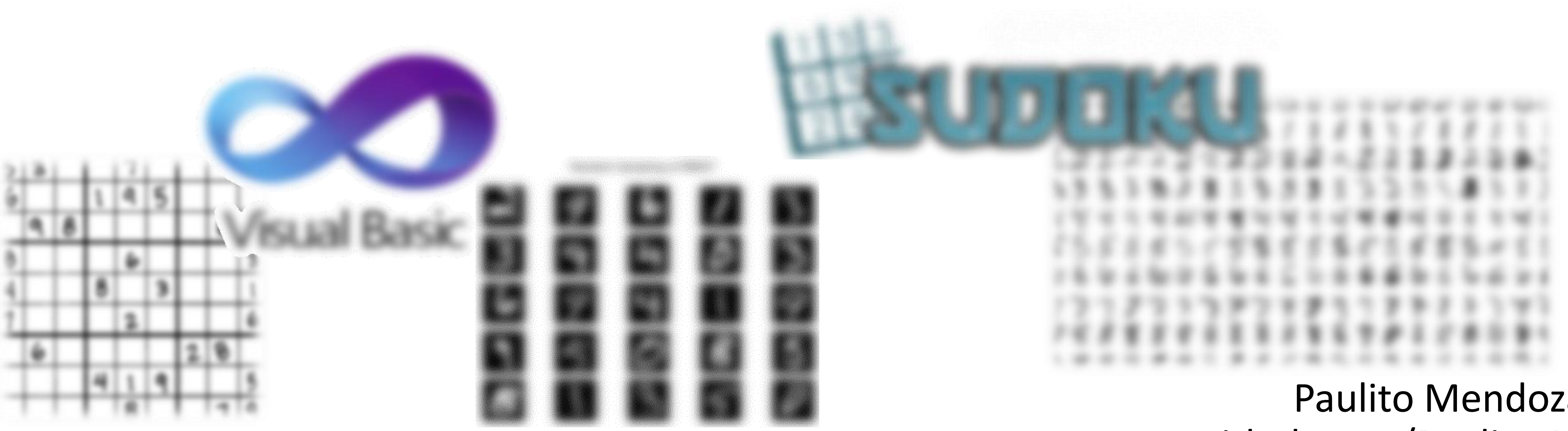
Solving Sudoku:
An Application of Handwritten Digit Classification



Paulito Mendoza
github.com/Paulito-M

Agenda

- Overview
- MNIST digit database
- Algorithms
- Sudoku Puzzle Solver: DEMO
- Conclusions

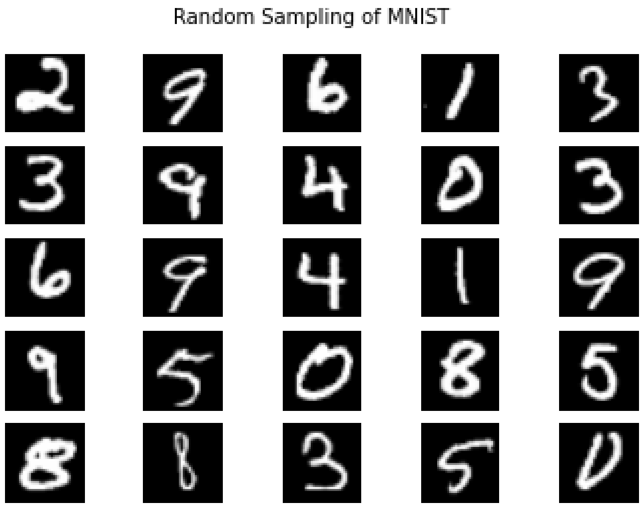
Overview

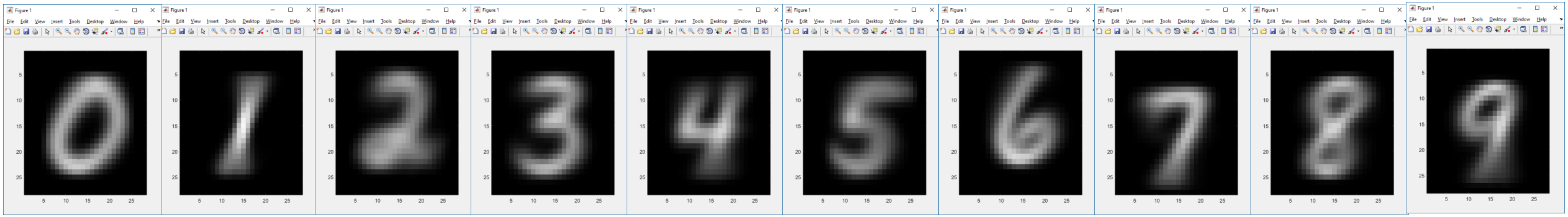
- *Motivation: why are we doing this?*
  - Building puzzle solvers is more interesting than solving puzzles manually
  - Wrote a Sudoku solver some years ago; it needs
    - A GUI
    - Some automated input so you don't have to type in the puzzles
      - *We need a classifier algorithm to read digits*
- *Approach: what are we doing?*
  - Prototype and test (at least one) algorithm to recognize handwritten digits
  - Write a Windows GUI to
    - Implement algorithm
    - Apply algorithm to scanned Sudoku image(s) to instantiate puzzle data
    - Attempt to solve using existing Soduku solver algorithm

VB.NET : not trendy, but reliable

Modified NIST (MNIST) Digit Database

- Original MNIST database:
  - Each image: 28x28 pixels
  - 60K training samples, 10K test samples
  - http://yann.lecun.com/exdb/mnist/
- Abbreviated MNIST database:
  - http://cis.jhu.edu/~sachin/digit/digit.html
  - 10K training samples, 1K for each digit 0-9
  - Decomposed into easier file format





Algorithms
Bayes Rule

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- Used in spam filtering:

$$P(IsSpam|SpamWords) = \frac{P(SpamWords|IsSpam)P(IsSpam)}{P(SpamWords)}$$

$$= \frac{P(SpamWords|IsSpam)P(IsSpam)}{P(SpamWords|IsSpam)P(IsSpam) + P(SpamWords|IsHam)P(Ham)}$$

- Used in digit recognition:

$$P(Digit\_i|Bitmap\_x) = \frac{P(Bitmap\_x|Digit\_i)P(Digit\_i)}{P(Bitmap\_x)}$$

Algorithms

Bayes Rule for Digit Recognition: MAP Estimator

Assume all digits have equal probability

$$Digit = \arg\max_i P(Digit\_i \mid Bitmap\_x)$$

$$= \arg\max_i \frac{P(Bitmap\_x \mid Digit\_i)\cancel{P(Digit\_i)}}{\cancel{P(Bitmap\_x)}}$$

$$= \arg\max_i P(Bitmap\_x \mid Digit\_i)$$

Find expression P(bitmap) for each possible digit!

Algorithms

Binary naïve Bayes

- For digit *i*, probability of observing Bitmap_x

$$= P_i(x_0, x_1, x_2 \ldots, x_{783})$$

- "naïve": assume each pixel is independent

$$= P_i(x_0)P_i(x_1)P_i(x_2)\ldots P_i(x_{783})$$

$$= \prod_{j=0}^{783} P_i(x_j)$$

- Where
  - $P_i(x_j) = p_{i,j}$ probability that pixel *j* of digit *i* is ON
  - $P_i(x_j) = (1 - p_{i,j})$ probability that pixel *j* of digit *i* is OFF
  - $p_{i,j}$ is calculated by summing the pixel values of pixel *j* over all samples of digit *i*
- Avoid underflow:

$$= \sum_{j=0}^{783} \ln P_i(x_j)$$

MATLAB: ~83% success

Algorithms

Gaussian naïve Bayes

- Similar, but: use pixel values (0-255), calculate Gaussian distribution for each pixel of each digit!
- mean $\mu_{j,i}$: average value of pixel *j* of digit *i* over all samples
- Variance $\sigma_{j,i}$: average value of ($\text{pixel}_{j,i} - \mu_{j,i}$)^2 over all samples

$$P(Bitmap\_x \mid Digit\_i) = \prod \frac{1}{\sqrt{2\pi\sigma_{j,i}{}^2}} e^{-\frac{(x_{j,i}-\mu_{j,i})^2}{2\sigma_{j,i}{}^2}}$$

$$\ln P(Bitmap\_x \mid Digit\_i) = \sum \ln \frac{1}{\sqrt{2\pi\sigma_{j,i}{}^2}} e^{-\frac{(x_{j,i}-\mu_{j,i})^2}{2\sigma_{j,i}{}^2}} = \sum \left(\ln \frac{1}{\sqrt{2\pi\sigma_{j,i}{}^2}} - \frac{(x_{j,i}-\mu_{j,i})^2}{2\sigma_{j,i}{}^2} \right)$$

MATLAB: ~79% success

Algorithms

Other algorithms…

- Multivariate Gaussian
  - Each image is a vector of 784 pixels
  - 784-dimensional Gaussian
  - Mean is vector mean of 784 pixels
  - Calculate covariance matrix
- Neural net approaches

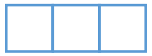
MATLAB: ~63% success
2.5 hrs to run
(PBKAC)

Neural net approaches: < 1% error rate

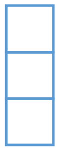
Sudoku Solver

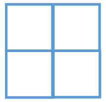
- C++ implementation of Sudoku Puzzle Solver
- Architecture: three classes of objects
  - cell

  - gridset
    - row, column, or square of cells

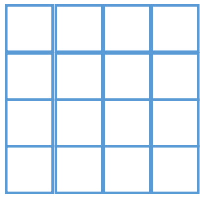


  - grid
    - NxN cells aggregated into rows, columns, squares





  - Each cell…
    - EITHER has a value (1..N)
    - OR has a list of possible values
    - Belongs to a row, a column, and a squares
  - Each grid…
    - Is a set of N x N cells
    - Has N rows
    - Has N columns
    - Has N squares



Sudoku Solver

- Recursive Backtracking Algorithm
  - **DETERMINISTIC SOLUTION:**
  - For each unpopulated cell in the grid
    - define that cell's possible values, based on the values already present in the other cells contained in that cell's row, column, and square
  - For each row, column, and square
    - If there is a cell with a unique possible value (ie it does not exist in the other cells for that gridset), populate that cell with that value, and restart the algorithm
  - **RECURSIVE ITERATION:**
  - At this point: there are no cell values that can be deterministically be defined
    - For each unpopulated cell in the grid
      - Loop through its possible values:
        - Populate the cell with the possible value
        - Duplicate the whole grid
        - Recursively call this algorithm with the new grid
          - If NOT SUCCESS: continue loop
      - If loop ends without SUCCESS: exit
  - If NOT SUCCESS: grid is unsolveable

1	4	3	
3		1	4
2	3		
	1		3

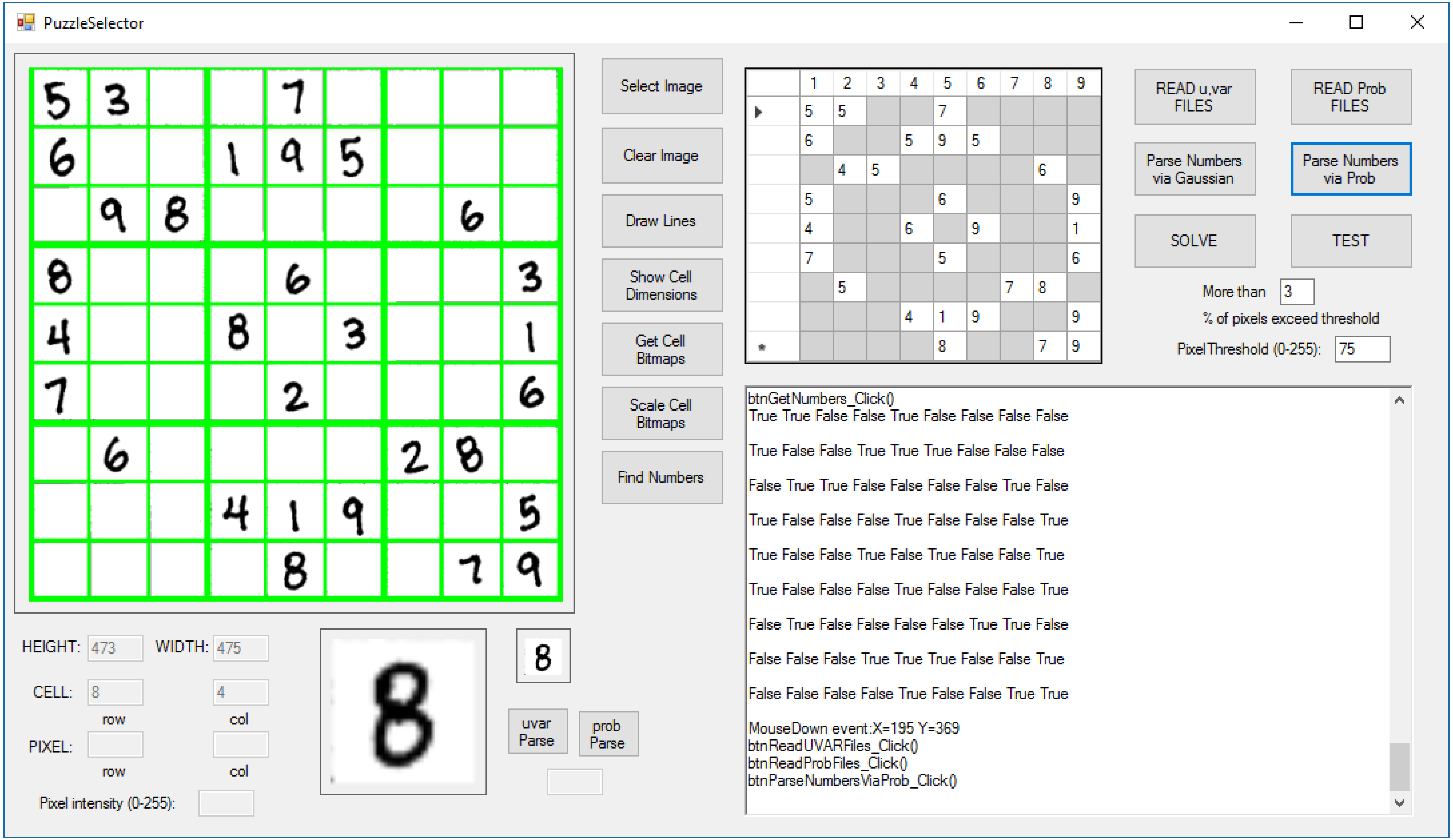
1	4	3	2
3	2	1	4
2	3	4	1
4	1	2	3

Sudoku Solver

DEMO

Conclusions

- 83% success on MNIST != good results
  - With (3) scanned puzzles
  - Penstroke thickness; scan quality; …
- 80/20 Rule aka Pareto principle
  - 80% of the effects come from 20% of the causes
  - 80% of the development time comes from 20% of the features
- VB.NET is very slow; C++ DLL better
  - Recursive algorithm + garbage collection
  - VB.NET is “managed code”
- *Questions? Comments?*


PuzzleSelector
Select Image
Clear Image
Draw Lines
Show Cell Dimensions
Get Cell Bitmaps
Scale Cell Bitmaps
Find Numbers
READ u,var FILES
READ Prob FILES
Parse Numbers via Gaussian
Parse Numbers via Prob
SOLVE
TEST
More than 3
% of pixels exceed threshold
PixelThreshold (0-255): 75
HEIGHT: 473
WIDTH: 475
CELL: 8 4
row
col
PIXEL:
row
col
Pixel intensity (0-255):
uvar Parse
prob Parse
btnGetNumbers_Click()
True True False False True False False False False
True False False True True True False False False
False True True False False False False True False
True False False False True False False False True
True False False True False True False False True
True False False False True False False False True
False True False False False False True True False
False False False True True True False False True
False False False False True False False True True
MouseDown event:X=195 Y=369
btnReadUVARFiles_Click()
btnReadProbFiles_Click()
btnParseNumbersViaProb_Click()