Perceptual Hashing Image Similarity Tool

p.h.i.s.t.

Dennis Devey

d.m.devey@gmail.com

Inspiration

Originally developed this tool to identify Jihadist social media accounts

1. **Download profile pictures from Twitter**
2. **Strip metadata**
3. **Get MD5 hash of image**
4. **Compare against known database**

MD5 Hashing Review

- Hexadecimal representation of binary that composes the image
- One pixel change results in an entirely different hash
  - Avalanche effect
- Great* for checking file integrity
  - Terrible for looking for images

*No, no it is not.

Why MD5 Doesn't Work

Original Image:
488A9D6A6571F0420D8BBAFE170C46C5

Original Image w/ One Pixel Changed:
6140B693468264DA8316E140F604D507

Problem

Recognize if a given image, regardless of editing, is in a database and return data associated with that image

-File Format Agnostic

-Scale Invariant

-Color Invariant

-Rotation/ Horizontal Flip Invariant

-Handle Edits of up to ___ % of original picture

Perceptual Hashing

A perceptual hash is a function that is able to transform a given image into a hash of a specified length based off of the image's visual properties, which means that similar images return similar hashes and visually identical images return matching hashes. Queries to a database of these hashes returns the hash of the image that is most similar to the queried one.

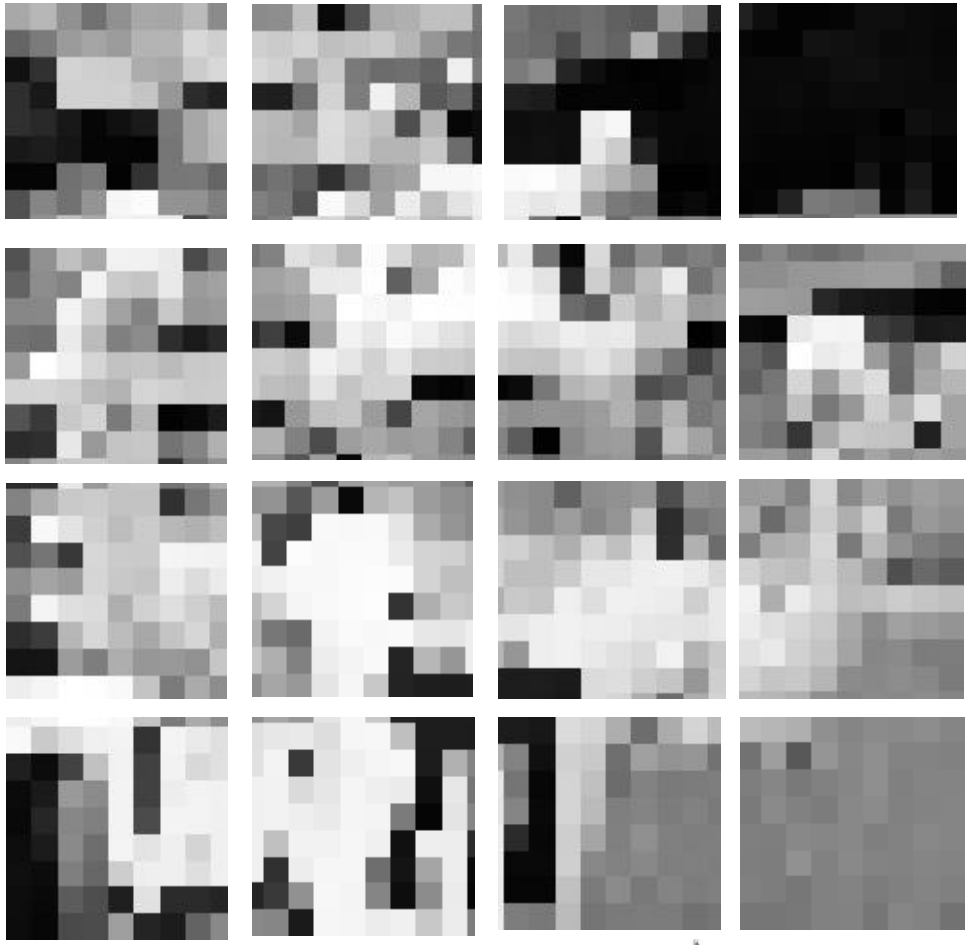
Original Image

Grey Scale

Resize

Resize (Lowers Resolution)

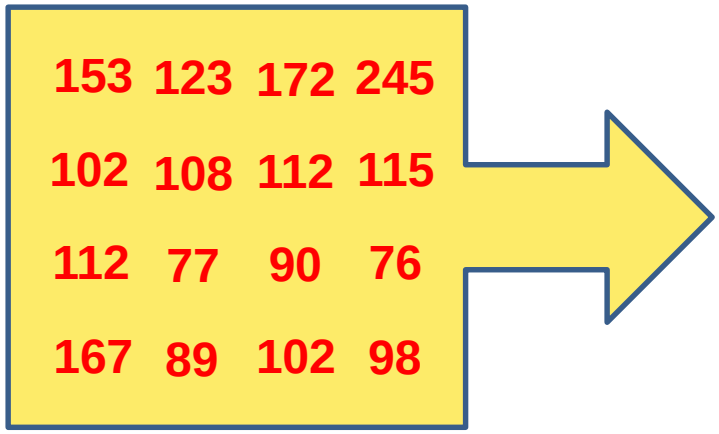
Split



Find Average Pixel Value

153	123	172	245
102	108	112	115
112	77	90	76
167	89	102	98

Hashing Function



Many ways to turn that grid into a matrix:

- Average = Compare each square's avg. color to grid avg
- Gradient = Compare each square's avg. color to adjacent square(s)
- Space Filling = Do one of the previous functions but follow a space-filling curve
  - Increases entropy of result, allows greater accuracy

Same functions can be applied to grid after Discrete Cosine Transform(very accurate)

Matching Examples

Gradient Hash
-7000fcbb7ef44da0db64ff07d06d280d868d

Discrete Cosine Transform
-03a43f42ff1e36f31e2939901c9899db8778

Gradient Hash
-7000fcbb7ef44da0db64ff07f30d282d0687

Discrete Cosine Transform
-03e43f52fd3e34f21e2979901c98899b876b

Matching Examples

03fa42cf36ef2903

03fa42ef34ef2903

03fa42ef162fad03

03fa42cf36ef2903

03fa42ef36af2903

Images It Doesn't Recognize

Matching Limitations

- Rotations and Reflections
  - Trivial to simply rotate/reflect image and requery
    - Not efficient
  - Hashes are binary arrays
    - Implement reflection/rotation at that level
- Major Edits
- Cropping
  - Can handle *some* cropping, but not reliably

Hashing Function Wrap Up

Different applications will require different levels of accuracy.

The more matches you find, the more false positives.

Hash lengths are totally arbitrary.

The longer you make a hash, the more accurate it is.

If your tool requires zero false positives, that will require a very different function than a tool used for discovery.

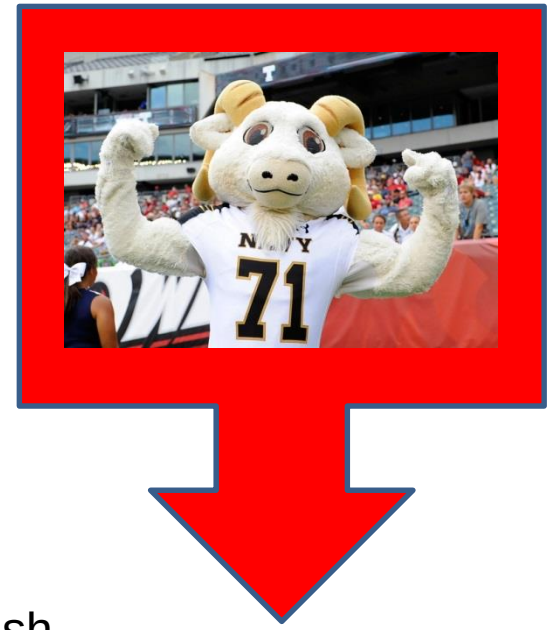
Data Structures

- **Existing Industry Implementations**
  - **Hash Table – Requires exact match = O(*1*)**
  - **K-Dimensional Tree**
    - **O(*log(n)*) time does not scale, especially with high dimensionality**
  - **Linear search for ‘closeness’**
    - **Must check every item in database = O(*n*)**

I find matches in near constant time

Same speed to search for a match in 10,000,000 images as 10,000

Basic Query Flow



16 Byte Hash

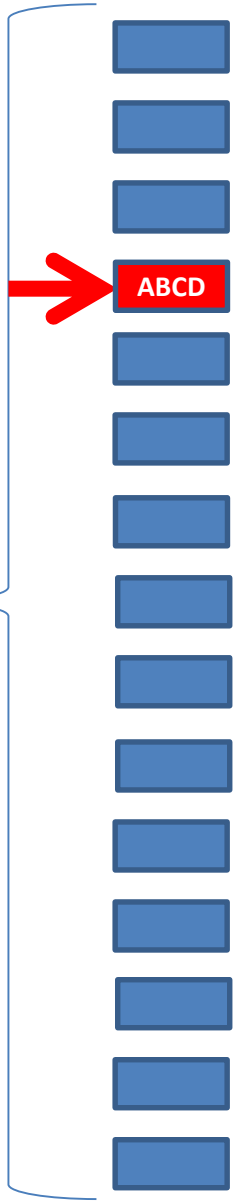
A123B123C123D123

4 Byte Hash

ABCD

Database

Hash Lengths are Variable, Effect Speed of Data Structure and Accuracy of Function.

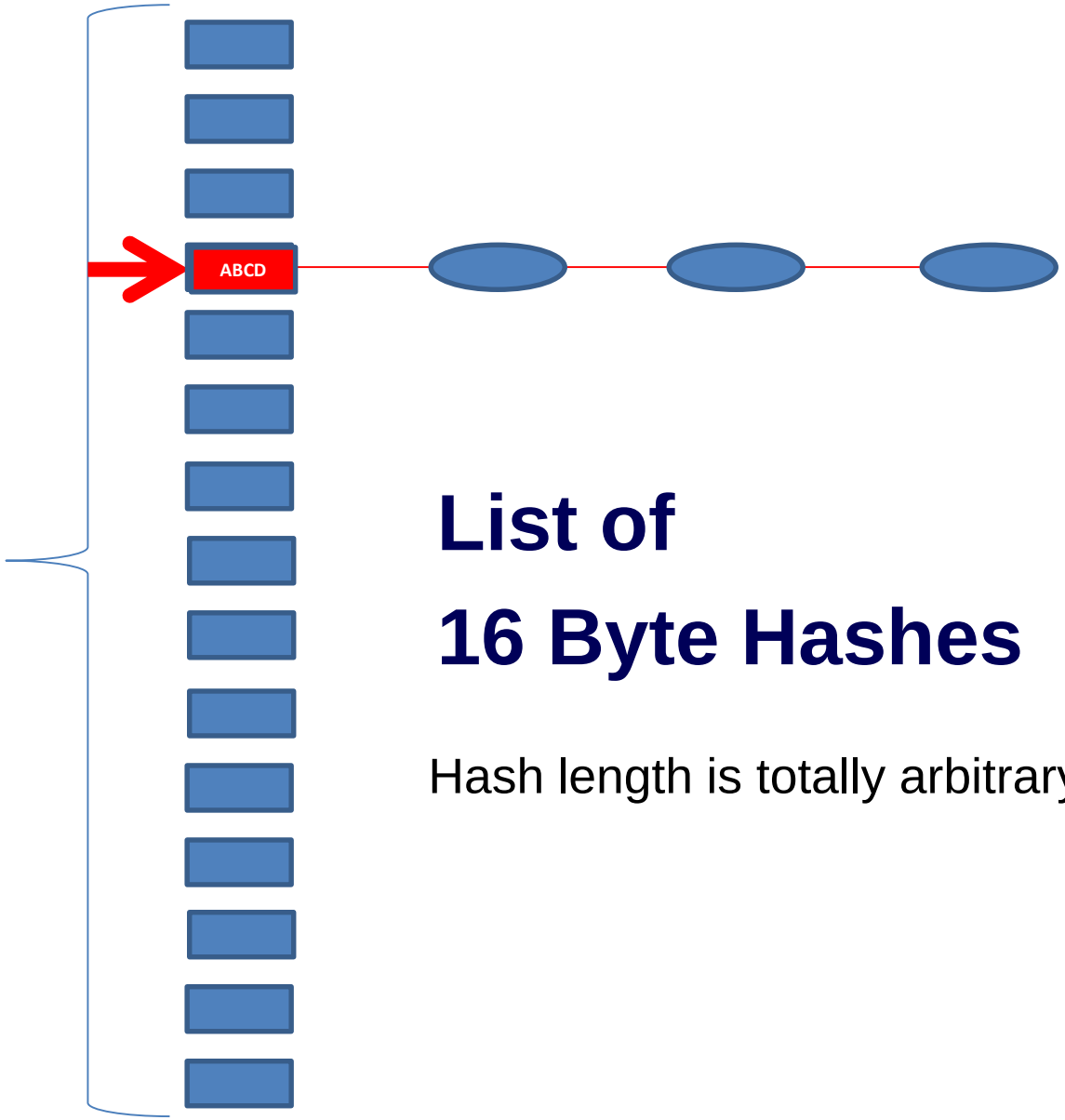


ABCD,
A123B123C123D123

Check 4 Byte Hash in Dictionary

The longer the “Short” Hash is, the more buckets we will have (16^n). The more buckets, the more images we can store at near constant time.

ABCD,
A123B123C123D123



List of
16 Byte Hashes

Hash length is totally arbitrary

Check if 16 Byte Hashes are Within Hamming Distance of 3

(totally arbitrary distance)

Hamming Distance

- **Minimum number of substitutions required to change one string into the other**

For Example, the strings ‘AB’ and ‘BB’:

‘AB’ = 1010 1011

‘BB’ = 1011 1011

Hamming Distance

‘AB’ = 1010 1011

‘BB’ = 1011 1011

Hamming distance of 1

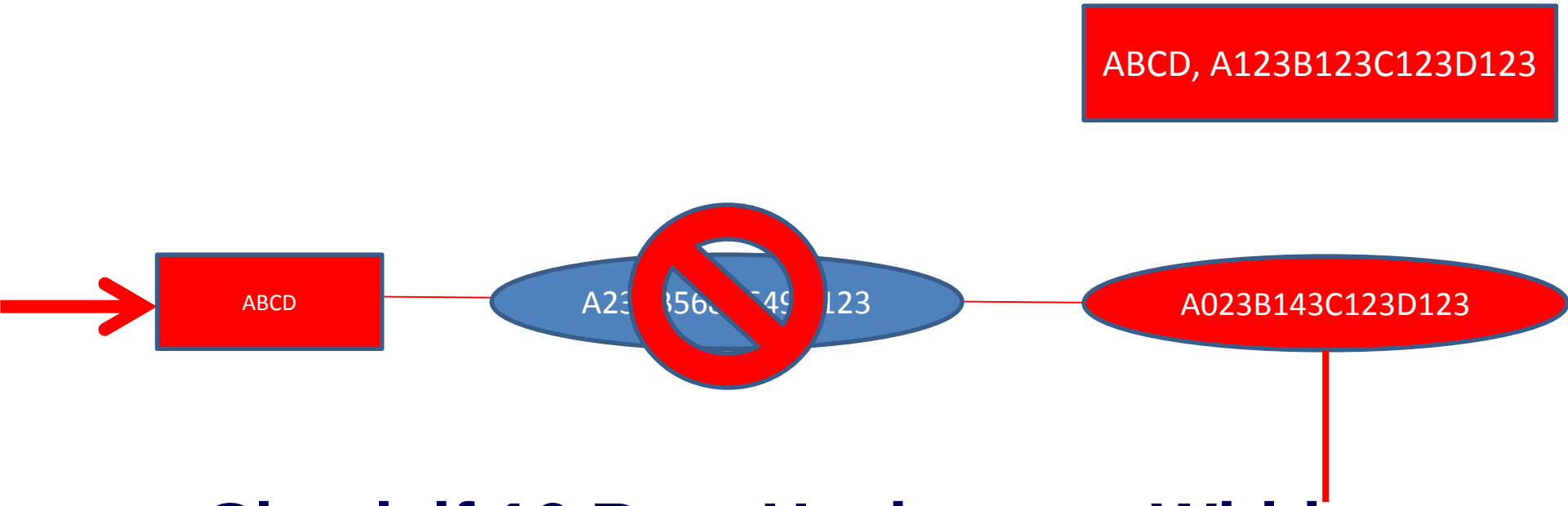
- **Various ways to find hamming distance**

Check if 16 Byte Hashes are Within Hamming Distance of 3

(totally arbitrary distance)

Check if 16 Byte Hashes are Within Hamming Distance of 3

(totally arbitrary distance)



Check if 16 Byte Hashes are Within Hamming Distance of 3

ABCD. A123B123C123D123

Add to Back


A023B143C123D123
A023B143C123D123
A123B123D123
A123B123C123D123

Add to Back

Improvements

- **Optimize bucket query speed**
  - **Test various methods of finding hamming distance**
  - **Sorting**
- **Replace buckets with k-d trees or MVP trees**

Data Structure Wrap-Up

You are going to have to write your own database implementations for whatever application you choose to integrate this with

Different applications will have different numbers of images being queried, different databases of targeted images, and different tolerances for false positives/ false negatives.

Make sure whatever you choose scales and still gets the job done.

Perceptual Hashing Applications

- Scraping Social Media
- Crawling Websites
- Forensics
  - Image Deduplication
  - Check against database of illegal images
- Phishing Landing Pages (screenshot and check)
- Identifying Malware C2 Domains (screenshot and check)
- Video Screenshotting
- Sound byte fingerprinting

My Favorite Open Source Implementations

phist

https://github.com/deveyNull/phistOfFury

Python Functions

https://pypi.python.org/pypi/ImageHash

C Functions

http://www.phash.org/download/

C# Function

https://github.com/jforshee/ImageHashing

My Side Project's Side Project's Side Project

BONUS CODE!

PERCEPTUAL HASHING BINARY SIMILARITY TOOL

p.h.b.s.t.

Fork of phist that matches files in a similar manner.

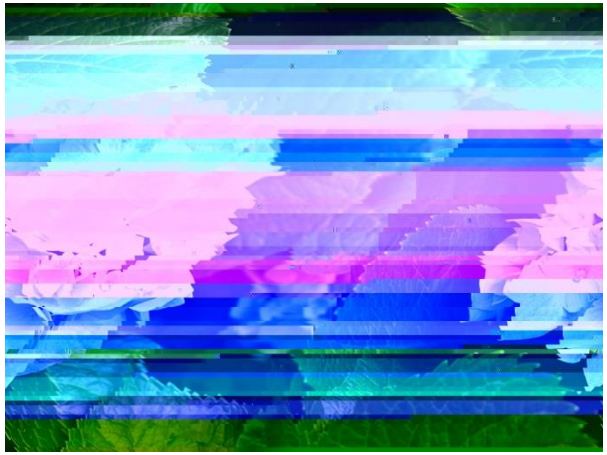
Gimmicky, but it works.

Block based hashing of the actual content of binary

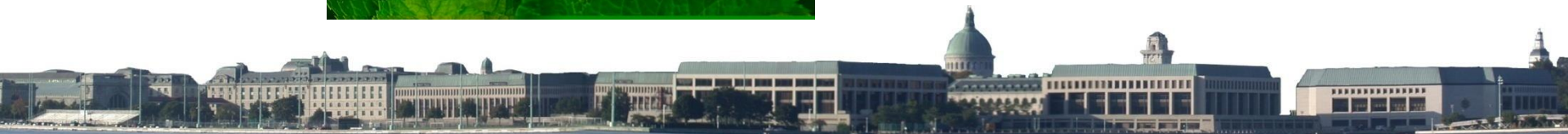
https://github.com/deveyNull/phbst

Now available in C# !!!

p.h.b.s.t. Finds Matches



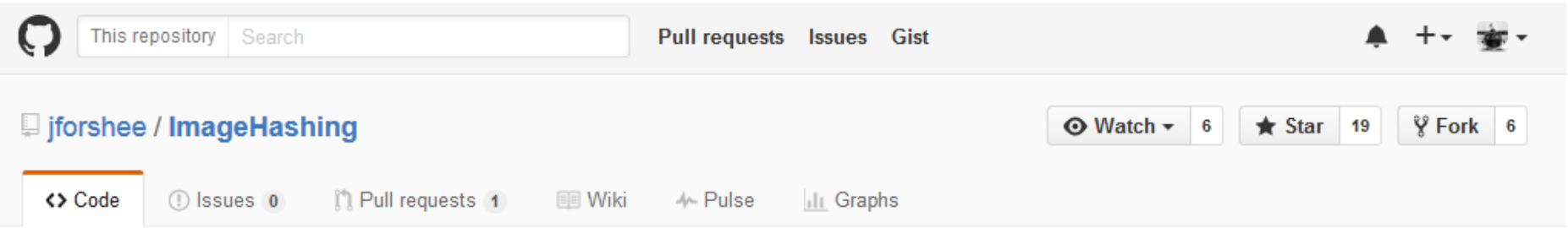
Unaffected by corruption or bit flips



DEMONSTRATION

Code Borrowed From

http://www.codeproject.com/Articles/28512/Duplicate-Files-Finder



https://github.com/jforshee/ImageHashing

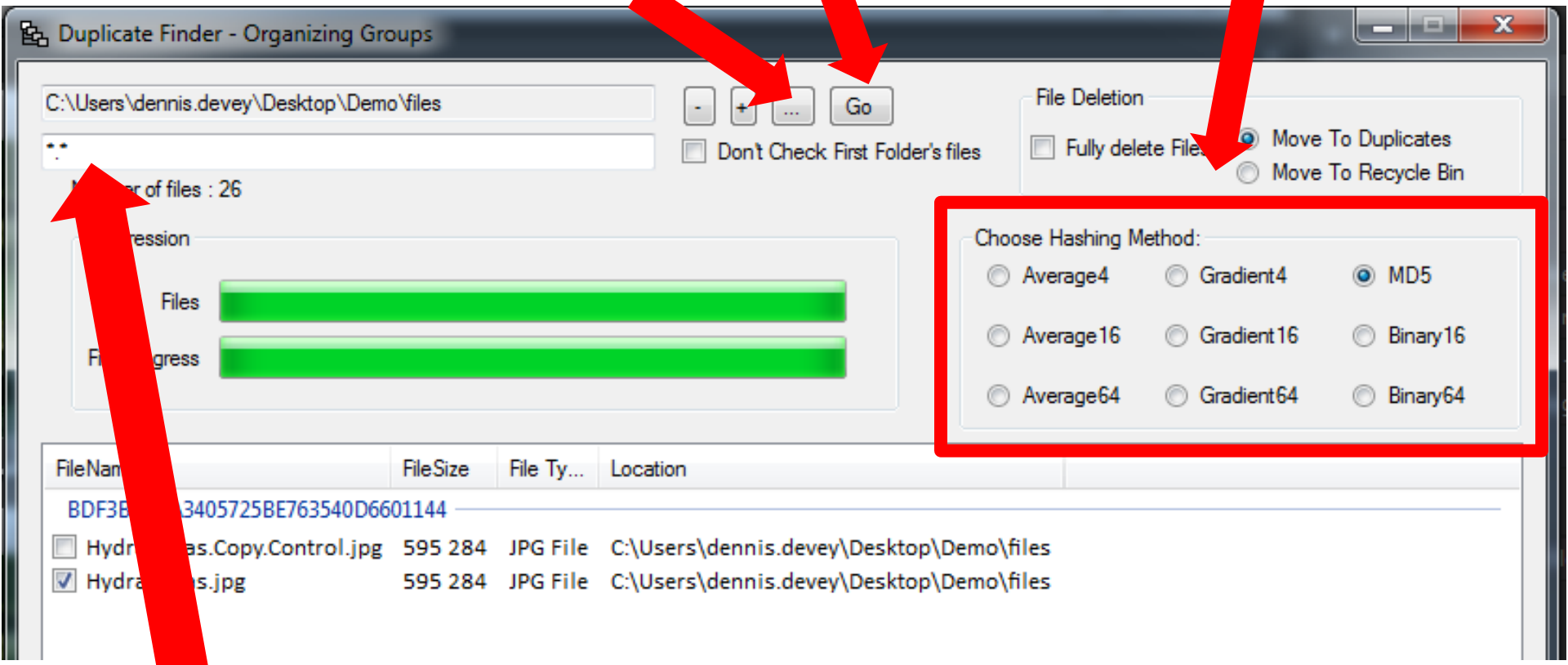
I hollowed out the GUI and borrowed the basic hash logic. I don't know C#, sorry.

How It Works:

Run

Folder Select

Hash Function Select



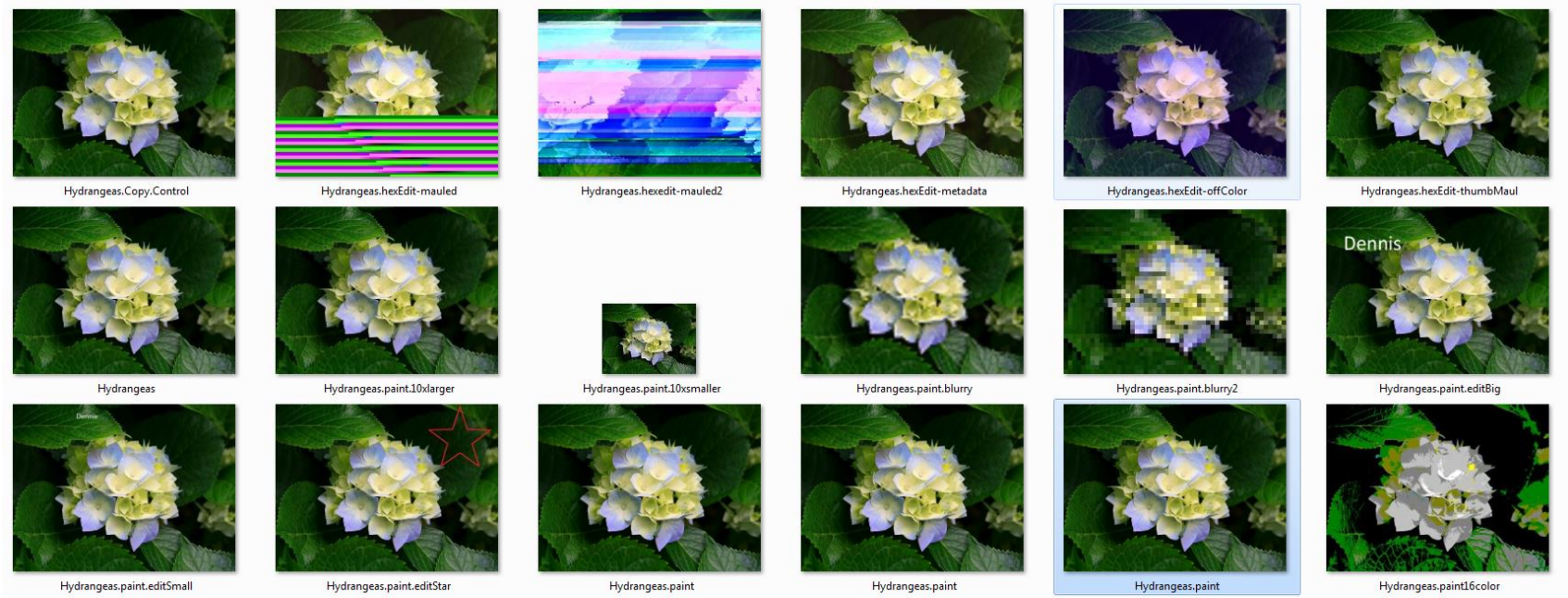
File Format Select

Can do various logic(;, |, !)

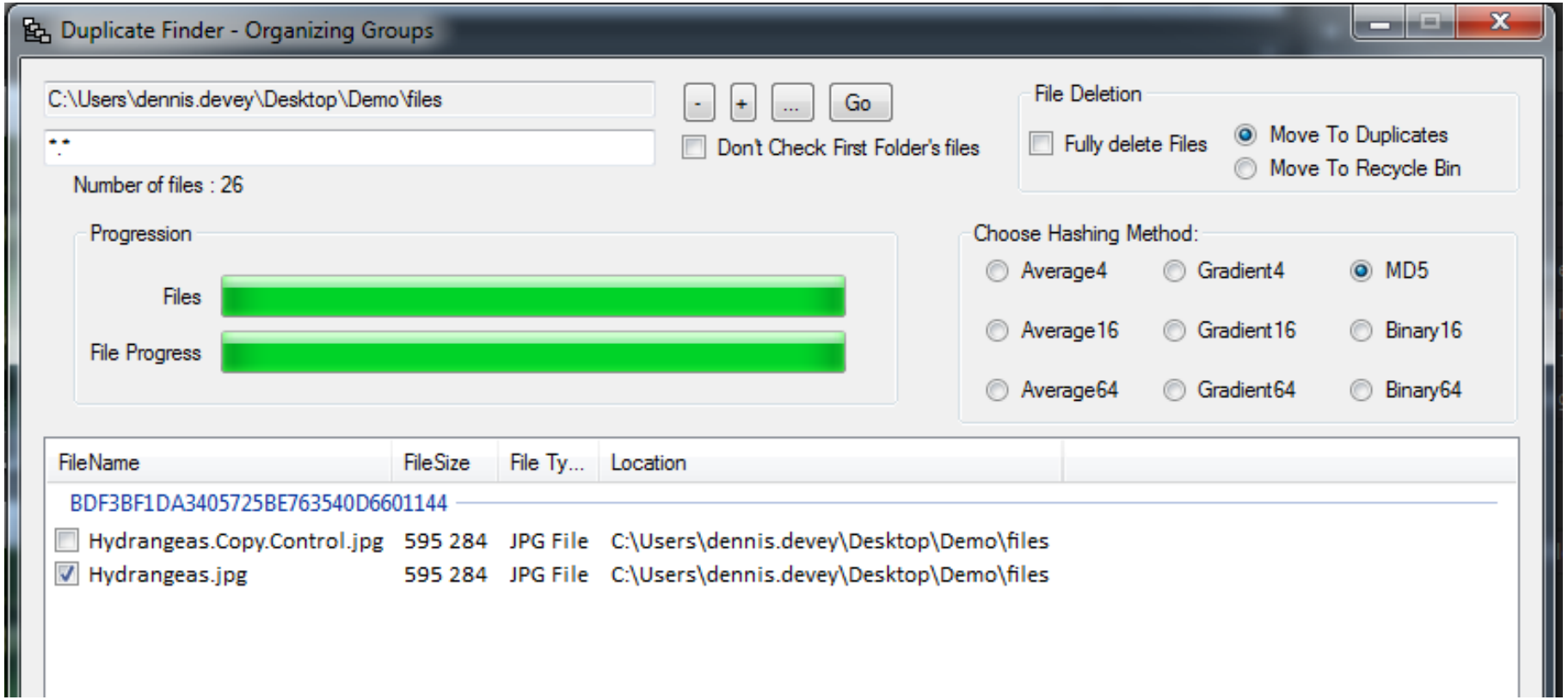
Test Images

Directory of Slightly Different Images

Quality, Color, Size, Format …



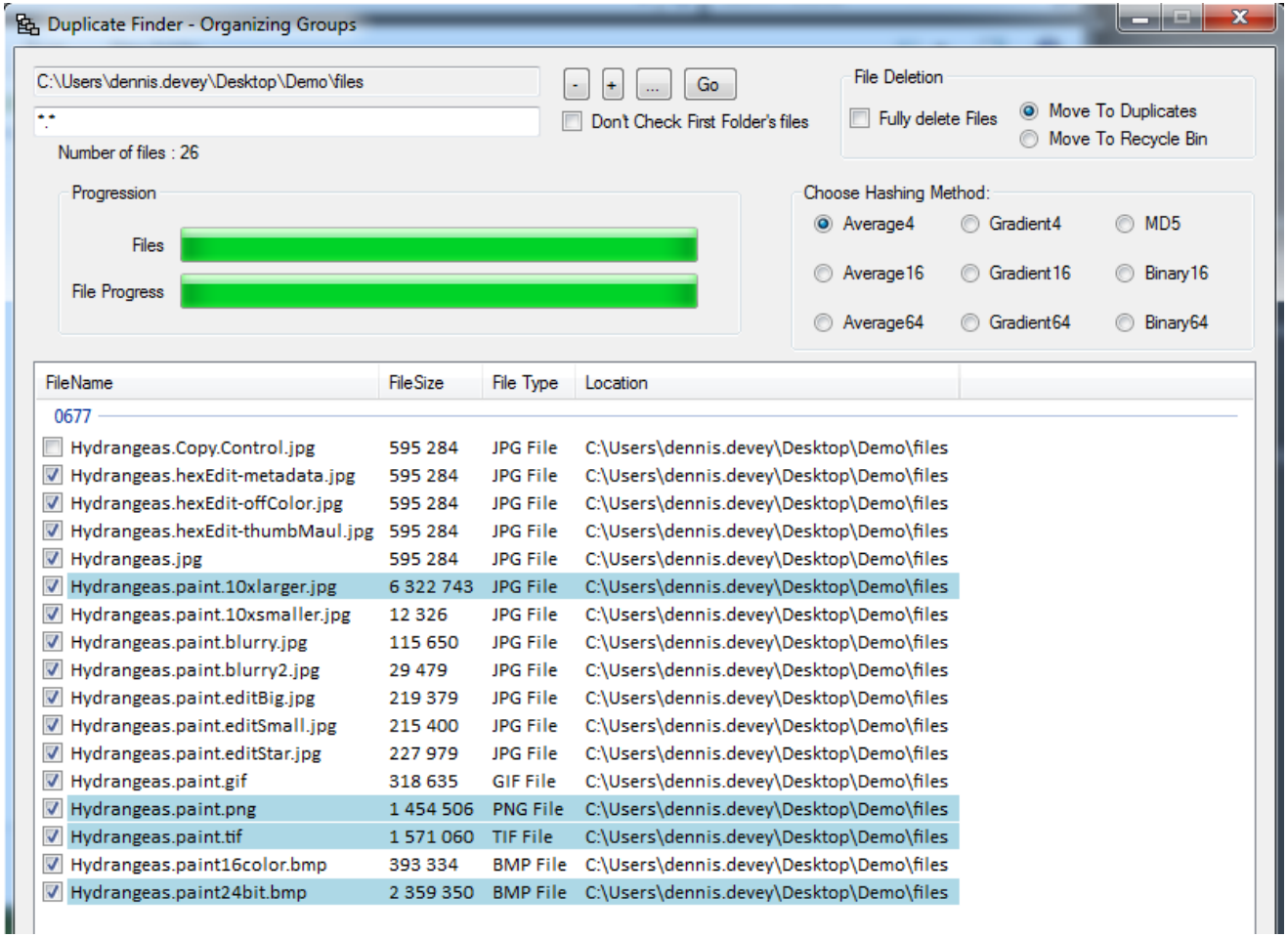
MD5



Function Retained From Original Tool

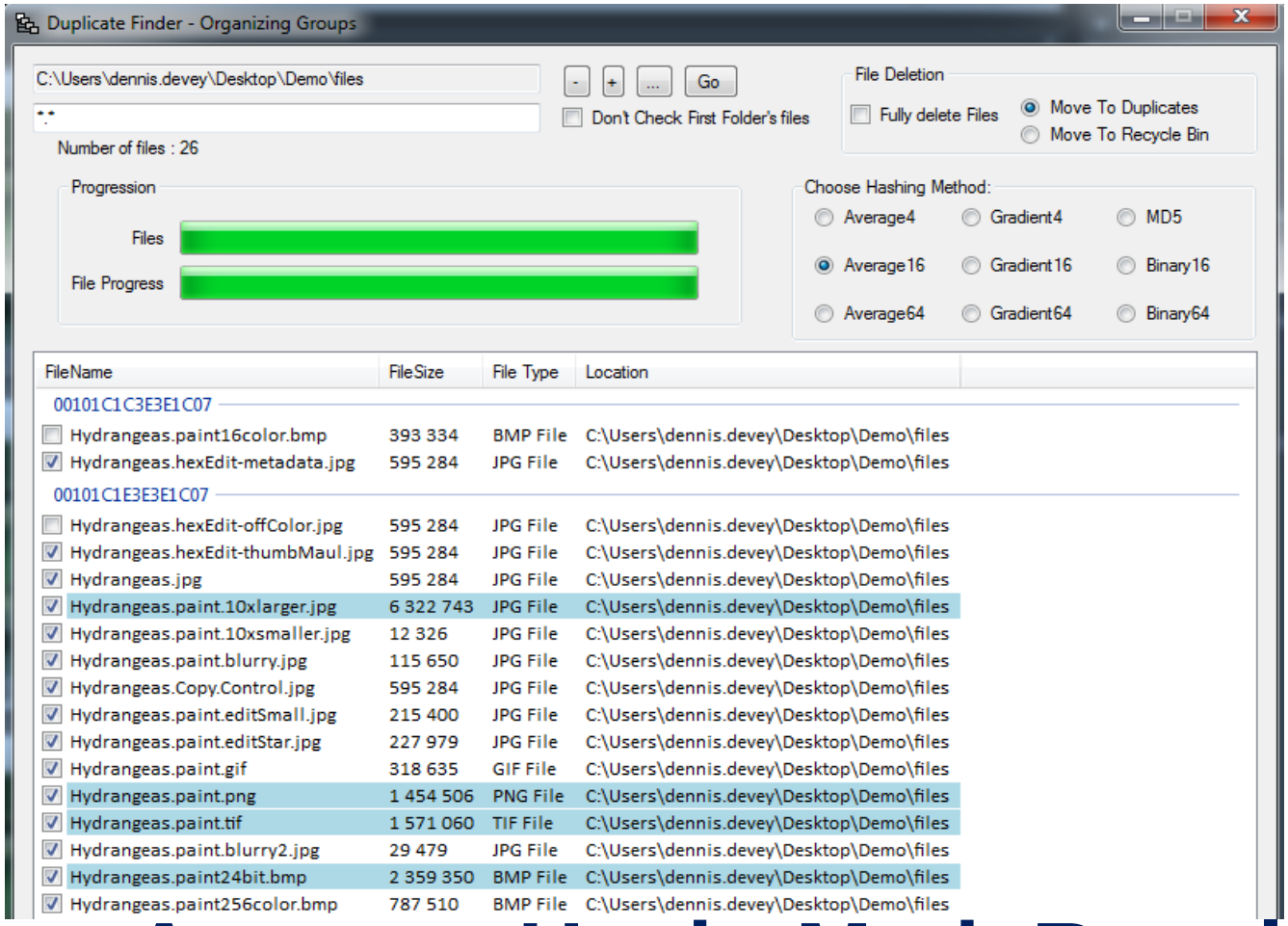
MD5 is good For Checking… File Integrity?

Average4



4 Byte Average Hash, Low Resolution
Best for Initial Bucketing

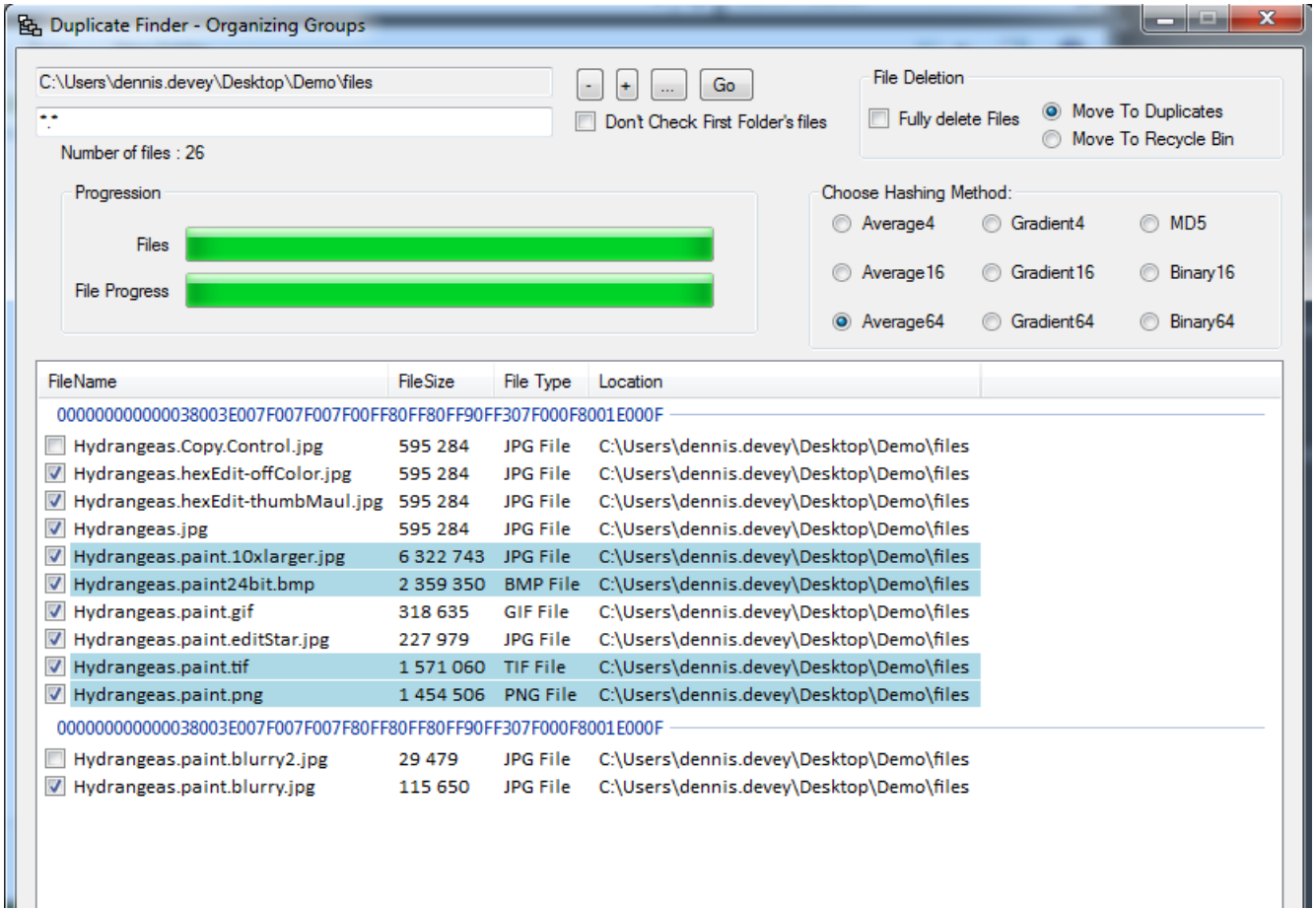
Average16



16 Byte Average Hash, Med. Resolution

Good For Discovery

Average64



64 Byte Average Hash, High Resolution
Accurate, Low False Positive

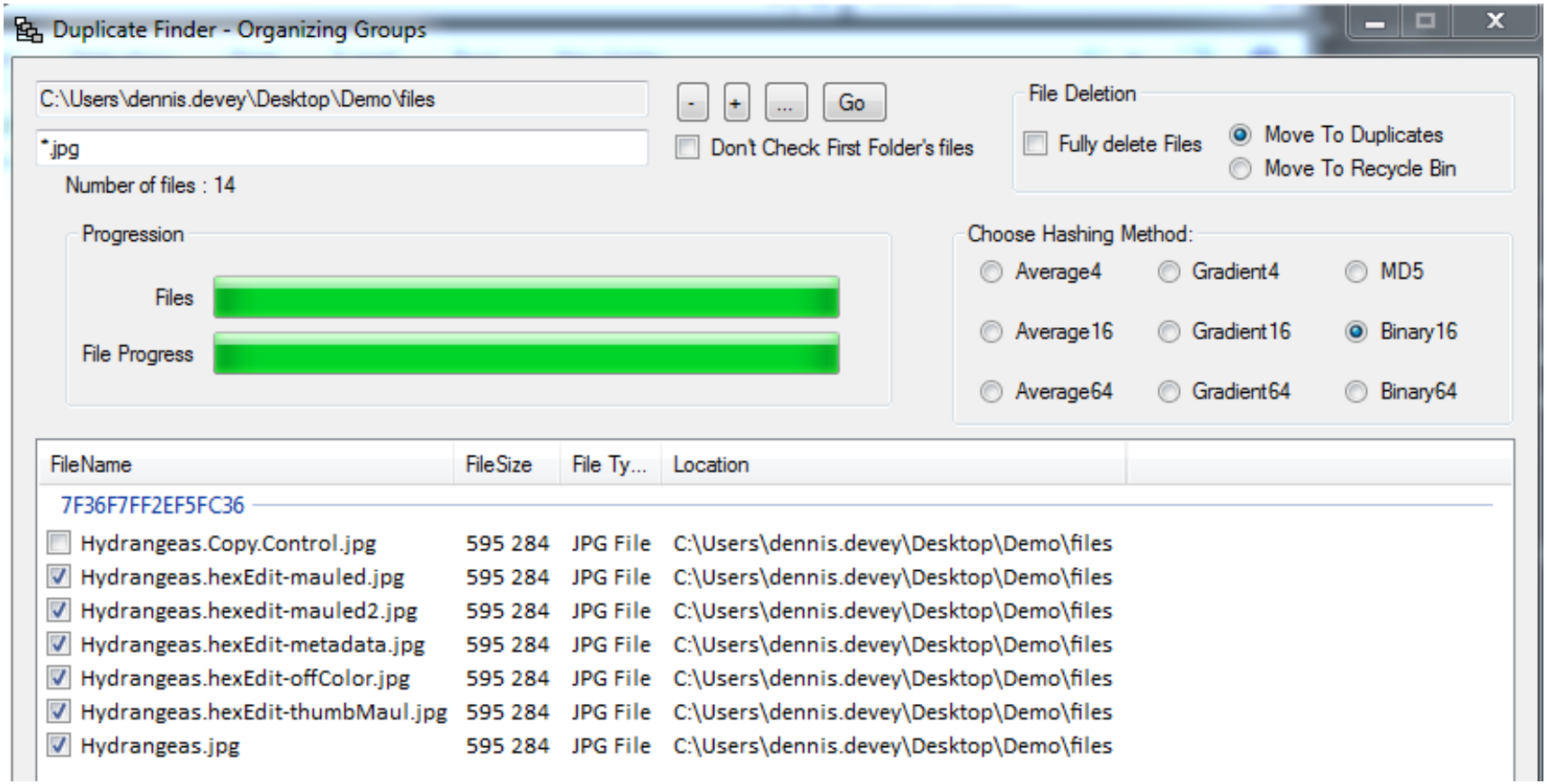
Gradient Hashes

4, 16, 64 Byte Variants

Fewer False Positives, More False Negatives

I was unable to implement a few variations of gradient hash which *nearly* eliminate false positives entirely.

Binary16



16 Byte Average Hash, Med. Resolution
Ignores Bit Flips, Minor Corruption.

Questions?

d.m.devey@gmail.com

(914) 299-7537