

PatternFinder is a tool that finds non-overlapping or overlapping patterns in any input sequence.

Pattern Finder Input Parameters:

USAGE:

PatternDetective.exe [

-help

/?

-f [filename]

-min [minimum pattern length]

-max [maximum pattern length]

-c

-threads [number of threads]

-mem [memory limit in MB]

-ram

-hd

-i [minimum times a pattern has to occur in order to keep track of it]

-v [verbosity level]

-n

-o

-his [hd files]

-lr [low range pattern search]

-hr [high range pattern search]

-pname

-plevel [level to show detailed output]

-ptop [n most common patterns indicated by -plevel]

Options:

-help	Displays this help page
/?	Displays this help page
-f [string]	Sets file name to be processed
-min [unsigned long]	Sets the minimum pattern length to be searched
-max [unsigned long]	Sets the maximum pattern length to be searched
-c	Finds the best threading scheme for computer
-threads [unsigned int]	Sets thread count to be used
-mem [unsigned long]	Sets the maximum RAM memory that can be used in MB
-ram	Forces program to use only RAM
-hd	Forces program to use Hard Disk based on -mem
-i [unsigned long]	Minimum occurrences to consider a pattern (Default occurrences will be 2)
-v [unsigned long]	Verbosity level, turn logging and pattern generation on or off with 1 or 0
-n	Non overlapping pattern search
-o	Overlapping pattern search, this is set by default
-his	HD processing file history keeps or removes files level by level with a 1 or a 0
-lr	Search for patterns that begin with the value lr to 255 if hr isn't set, otherwise lr to hr range
-hr	Search for patterns that begin with the value hr to 0 if lr isn't set, otherwise lr to hr range
-pname	Do not print pattern string data
-plevel	Sets the level the user wants to see detailed output for
-ptop	Display the top N most common patterns in detail for the level indicated by -plevel

How to build PatternFinder:

PREREQS:

cmake version 2.5 or higher

c++11 compatible compiler

python 2.7 to run parallel serial jobs

Visual Studio 2012 or 2015 for building with Windows

download repo using https address <https://github.com/octopusprime314/PatternDetective.git>

or use git and ssh using address `git@github.com:octopusprime314/PatternDetective.git`

BUILD INSTRUCTIONS:

!!!!!!!!!!!!ALWAYS BUILD IN RELEASE UNLESS DEBUGGING CODE!!!!!!!!!!!!

Linux:

create a build folder at root directory

cd into build

`cmake -D CMAKE_BUILD_TYPE=Release -G "Unix Makefiles" ..`

`cmake --build .`

Windows:

create a build folder at root directory

cd into build

`cmake -G "Visual Studio 11 2012 Win64" ..` OR `cmake -G "visual Studio 14 2015 Win64" ..`

`cmake --build . --config Release`

How to run PatternFinder as a standalone executable:

LOCATION OF FILES TO BE PROCESSED:

Place your file to be processed in the Database/Data folder

EXAMPLE USES OF PATTERNFINDER:

1) `./PatternFinder -f Database -threads 4 -ram`

Pattern searches all files recursively in directory using DRAM with 4 threads

2) `./PatternFinder -f TaleOfTwoCities.txt -c -ram`

Finds the most optimal thread usage for processing a file

3) `./PatternFinder -f TaleOfTwoCities.txt -v 1`

Processes file using memory prediction per level for HD or DRAM processing

4) `./PatternFinder -f TaleOfTwoCities.txt -mem 1000`

Processes file using memory prediction per level for HD or DRAM processing with a memory constraint of 1 GB

5) `./PatternFinder -f TaleOfTwoCities.txt -min 5 -max 100`

Finds patterns of length 5 to 100 and then terminates processing

6) `./PatternFinder -f Boosh.avi -n`

Processes file using non overlapping processing

7) `./PatternFinder -f TaleOfTwoCities.txt -hd`

Processes file using the hard disk only.

8) `./PatternFinder -f Boosh.avi -i 10`

Processes patterns that occur at least 10 times or more. Default is 2.

9) `./PatternFinder -f TaleOfTwoCities.txt -plevel 3 -ptop 10`

Processes file using memory prediction per level for HD or DRAM processing and displays detailed information for level 3 patterns for the top 10 patterns found

10) `./PatternFinder -f TaleOfTwoCities.txt -plevel 3 -ptop 10 -pname`

Processes file using memory prediction per level for HD or DRAM processing and displays detailed information for level 3 patterns for the top 10 patterns found but doesn't print pattern string

How to run PatternFinder Python Scripts:

PYTHON RUN EXAMPLES:

1) `python splitFileForProcessing.py [file path] [number of chunks]`

Use `splitFileForProcessing.py` Python script to split files into chunks and run multiple instances of PatternFinder on those chunks

Ex. `python splitFileForProcessing.py ~/Github/PatternDetective/Database/Data/TaleOfTwoCities.txt 4`

equally splits up `TaleOfTwoCities.txt` into 4 files and 4 instances of PatternFinder get dispatched each processing one of the split up files.

2) `python segmentRootProcessing.py [file path] [number of jobs] [threads per job]`

Use `segmentRootProcessing.py` Python script splits up PatternFinder jobs to search for patterns starting with a certain value

Ex. `python segmentedRootProcessing.py ../Database/Data/Boosh.avi 4 4`

Dispatches 4 processes equipped with 4 threads each. Each PatternFinder will only look for patterns starting with the byte representation of 0-63, 64-127, 128-191, 192-255.

PatternFinder Input Files:

PatternFinder accepts any type of input file because it processes at the byte level.

PatternFinder Output Files:

Nine outputs are available. One is a general logger using ascii text format, another is the Output file which generates patterns based on -pname, -plevel, ptop and the remaining seven are Comma Separated Variable files used for post processing in MATLAB.

- 1) Logger file: records general information including the most common patterns, number of times a pattern occurs and the pattern's coverage at every level until the last pattern is found. Simple text file.
- 2) Output file: generates patterns based on -pname, -plevel and -ptop
- 3) Collective Pattern Data file: records each level's most common pattern and number of times the pattern occurs in CSV format.
- 4) File Processing Time: records each file's processing time in CSV format. Used for processing large data sets with many files.
- 5) File Coverage: records the most common pattern's coverage of the file in CSV format.
- 6) File Size Processing Time file: records each file's processing time and corresponding size in CSV format. Used primarily to isolate files in a large dataset that contain large patterns.
- 7) Thread Throughput: records the processing throughput improvement while incrementing the number of processing threads in CSV format. Typically used with -c option which tests threads in multiples of 2 starting at 1 until the number of cores on the machine has been met.
- 8) Thread Speed: records the processing time taken while incrementing the number of processing threads in CSV format. Typically used with -c option which tests threads in multiples of 2 starting at 1 until the number of cores on the machine has been met.

Output file contents is pattern string, number of instances, occurrence, average distance and location:

./PatternFinder -f TaleOfTwoCities.txt -threads 4 -ram -plevel 2 -ptop 10

Level 1

unique patterns = 83, average occurrence frequency = 9490.39, frequency of top pattern: 129157

Level 2

unique patterns = 1401, average occurrence frequency = 562.024, frequency of top pattern: 21032

1. pattern = e , instances = 21032, coverage = 5.34006%, average pattern distance = 6206.12, first occurrence index = 4

2. pattern = t, instances = 18017, coverage = 4.57454%, average pattern distance = 437.065, first occurrence index = 75

3. pattern = he, instances = 16814, coverage = 4.2691%, average pattern distance = 11276.5, first occurrence index = 3

4. pattern = th, instances = 16713, coverage = 4.24346%, average pattern distance = 8971.26, first occurrence index = 91

5. pattern =

, instances = 16366, coverage = 4.15535%, average pattern distance = 48.1297, first occurrence index = 65

6. pattern = d , instances = 14169, coverage = 3.59753%, average pattern distance = 10886.3, first occurrence index = 289

7. pattern = a, instances = 13777, coverage = 3.498%, average pattern distance = 628.75, first occurrence index = 80

8. pattern = , , instances = 11447, coverage = 2.90641%, average pattern distance = 5117.52, first occurrence index = 53

9. pattern = s , instances = 11112, coverage = 2.82135%, average pattern distance = 23865.6, first occurrence index = 123

10. pattern = in, instances = 10750, coverage = 2.72944%, average pattern distance = 9752.35, first occurrence index = 105

Level 3

unique patterns = 7934, average occurrence frequency = 98.9629, frequency of top pattern: 12204

Level 4

unique patterns = 26273, average occurrence frequency = 29.5211, frequency of top pattern: 8907

Level 5

unique patterns = 57380, average occurrence frequency = 13.0067, frequency of top pattern: 6427

Level 6

unique patterns = 86748, average occurrence frequency = 7.87542, frequency of top pattern: 2153

Level 7

unique patterns = 103739, average occurrence frequency = 5.72812, frequency of top pattern: 974

Level 8

unique patterns = 108726, average occurrence frequency = 4.53085, frequency of top pattern: 780

Level 9

unique patterns = 102229, average occurrence frequency = 3.80874, frequency of top pattern: 339

PatternFinder post processing scripts using the seven available CSV outputs with MATLAB:

- 1) DRAM versus HD Processing Speeds->DRAMtoHDPprocessingLiminationSpeeds.m
- 2) DRAM versus HD Performance->DRAMVsHardDiskPerformance.m
- 3) Most Common Pattern versus Coverage->MostCommonPatternLengthVsCoveragePercentage.m
- 4) Overlapping versus Non Overlapping Comparison->Overlapping_NonOverlappingComparison.m
- 5) Overlapping versus Non Overlapping File Speeds->OverlappingVsNonOverlappingFileSpeeds.m
- 6) Process Time versus File Size->ProcessTimeVsFileSize.m