

ПРОФИЛИРАНЕ НА ПРОГРАМЕН КОД

Research described in this tutorial was partially supported by the National Scientific Program "Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICTinSES)", financed by the Ministry of Education and Science.

Developed by Velbazhd Software LLC

Стъпките за работоспособност на софтуера са подредени в следния ред – компилация без грешки (по възможност и без warnings), режим на работа без аварийно прекъсване, вярно изпълнение на заложените изчисления и най-накрая оптимално бързо пресмятане.



По отношение на полезността и при програмния код важи емпиричното правило 20:80. Това означава, че 20% от кода изпълнява 80% от съществените задачи за които се използва софтуерът. В този контекст, не е важно всеки ред код да бъде оптимално бърз, а е важно съществените редове, които най-много участват в употребата да бъдат оптимално бързи.



Тъй като времето за разработка е компонентът с най-висока цена в софтуерното производство, изключително важно е правилно да се идентифицират фрагментите от софтуера, които да бъдат подложени на оптимизация, така че инвестираният ресурс да се изплати.



В практиката оптимизацията не се прави на случаен принцип или по интуиция, а се използват инструменти за профилиране, които ясно да покажат „тесните“ места в системата (bottlenecks). Профилирането включва колко често се използват отделните функции и колко от общото време на програмата е прекарано в конкретна функция.



Не рядко програмистите изпадат в неосъзната спирала за микро оптимизация на програмния код. Подобно нещо отнема изключително много време и увеличава значително вероятността за внасяне на дефекти в софтуера.



Генетичните алгоритми са глобална оптимизационна евристика. При тях множество решения се проверяват до колко доближават оптимална стойност. Всички тези междинни изчисления не се използват, но същевременно отнемат много време за пресмятане.





Search or jump to...

[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



TodorBalabanov / Ellipses-Image-Approximator

Unwatch 1

Unstar 3

Fork 0

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security

Insights

Settings

It is an attempt to approximate full color image with limited color ellipses.

Edit

color-reduction

image-vectorization

genetic-algorithm

ant-colony-optimization

g-codes

Manage topics

75 commits

1 branch

0 packages

0 releases

1 contributor

GPL-3.0

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download



TodorBalabanov Bug was found and it was fixed.

Latest commit 22341b0 19 hours ago

| | | |
|-----------------|---|--------------|
| doc | Sorting by colors was wrong and it was fixed. | 12 days ago |
| gradle/wrapper | Gradle build folders reorganization done. | 21 days ago |
| input | Sorting by colors was wrong and it was fixed. | 12 days ago |
| output | Read me file was updated. | 11 days ago |
| src | Bug was found and It was fixed. | 19 hours ago |
| .gitignore | G Code template was added. | 20 days ago |
| .travis.yml | Add file for automated building. | 4 years ago |
| LICENSE | Initial commit | 4 years ago |
| README.md | Some bugs were found and were fixed. | 6 days ago |
| build.gradle | Gradle build folders reorganization done. | 21 days ago |
| gradlew | Gradle build folders reorganization done. | 21 days ago |
| gradlew.bat | Gradle build folders reorganization done. | 21 days ago |
| settings.gradle | Source code formatting was done. | 21 days ago |

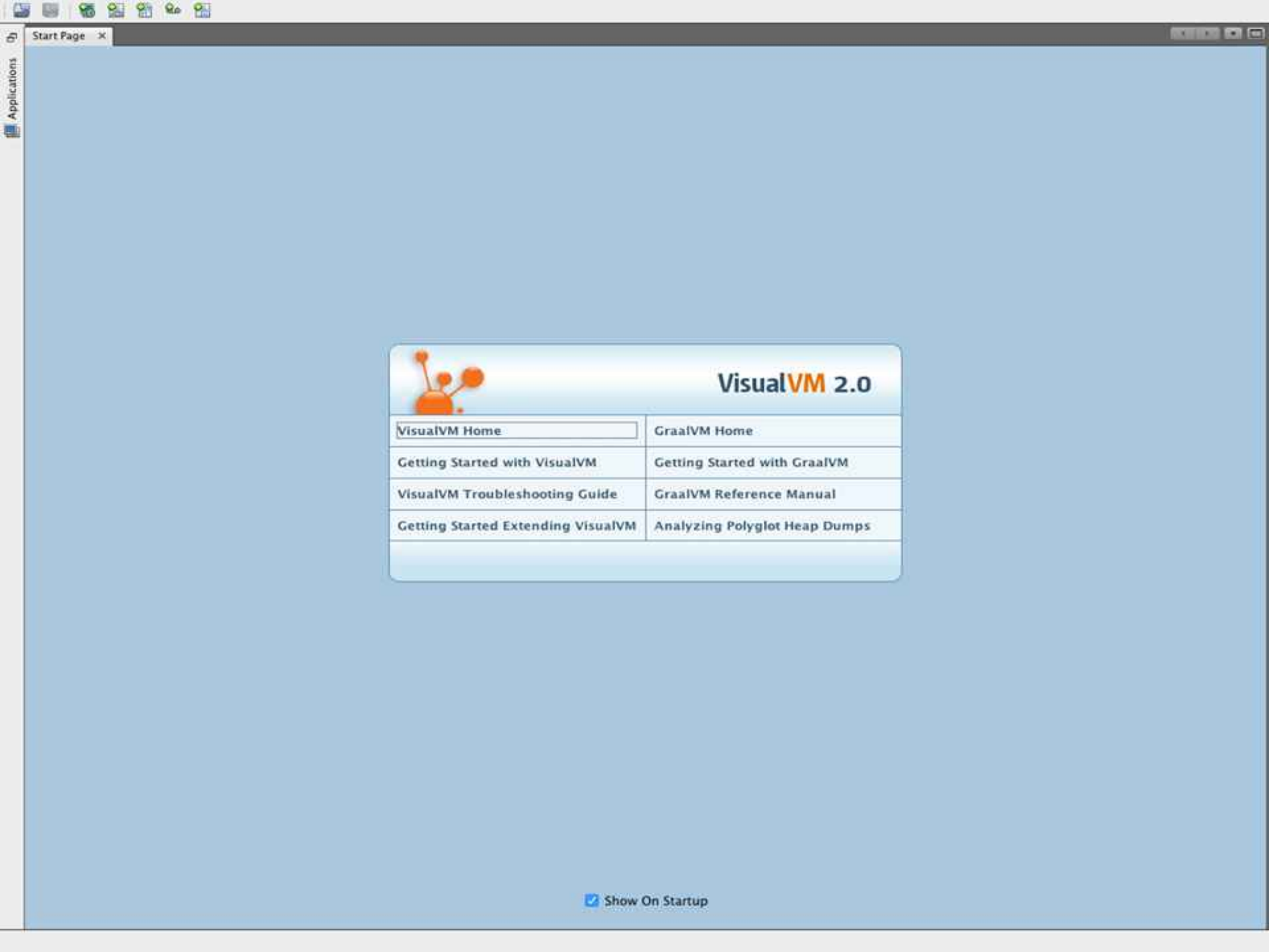
README.md



Ellipses Image Approximator

VisualVM е един от най-разпространените инструменти за профилиране на Java програмен код. С негова помощ ще бъдат определени „тесните“ места в проекта и ще бъде потърсена възможност за оптимизация.





VisualVM 2.0

| | |
|--|---|
| VisualVM Home | GraalVM Home |
| Getting Started with VisualVM | Getting Started with GraalVM |
| VisualVM Troubleshooting Guide | GraalVM Reference Manual |
| Getting Started Extending VisualVM | Analyzing Polyglot Heap Dumps |

Същината за бързодействието на целевата функция е добре известно от документацията на библиотеката Genetic Algorithms - Apache Commons - Apache Software. Дори без да е извършено профилиране е от значение да се определи колко бързо работи тази функция в конкретната реализация. За тези цел може да се преброят извикванията за фиксиран интервал от време.



```

106     this.image = image;
107     this.colors = colors;
108
109     checkValidity(getRepresentation());
110 }
111
112 int counter = 0;
113 @Override
114 public double fitness() {
115     counter++;
116     System.err.println(counter);
117
118     /* Sort ellipses by color with the most used color first. */
119     List<Ellipse> list = sort();
120
121     /*
122      * Draw ellipses.
123      */
124     BufferedImage experimental = new BufferedImage(image.getWidth(),
125                                                    image.getHeight(), BufferedImage.TYPE_INT_ARGB);
126     Util.drawEllipses(experimental, list);
127
128     /* Multiple-criteria for fitness value estimation. */
129     double size = list.size();
130     double distance = Util.distance(image, experimental);
131     double alpha = Util.alphaLevel(experimental, colors);
132
133     // TODO Better handling of multiple criteria should be implemented. At
134     // least coefficients should be parameterized from outside.
135
136     return -(1D * size + 100D * distance + 10D * alpha);
137
138     // return -(Math.pow(1D*size, 1)
139     // + Math.pow(100D*distance, 3)
140     // + Math.pow(10D*alpha, 2));
141 }
142

```


В случая са зададени 7 секунди за извършване на оптимизация с генетичния алгоритъм. Това е доста кратък времеви интервал, но може да бъде показателен за броя извиквания в които ще се бъде изчислена целевата функция.



```
MACMINI:libs todorbalabanov$ java -jar Ellipses-Image-Approximator-all.jar -input ../../input/0009.jpg -output ~/Desktop/ -g_code_print -g_code_x_offset 30 -g_code_y_offset 30 -g_code_z_up 35 -g_code_z_down 15 -g_code_scaling 0.7 -g_code_refill 0.5 -g_code_color_change 600 -ellipse_width 19 -ellipse_height 5 -ellipse_alpha 216 -colors 000000,808080,C0C0C0,FFFFFF,800000,FF0000,808000,FFFF00,008000,00FF00,008080,00FFFF,000080,0000FF,800080,FF00FF -pixel_closest_color -ga -ga_population_size 11 -ga_chromosome_size 100 -ga_tournament_arity 2 -ga_crossover_rate 0.9 -ga_mutation_rate 0.1 -ga_elitism_rate 0.1 -ga_optimization_time 7
```

В една многозадачна операционна система множество фактори влияят върху времето, което е нужно на една програма да извърши своите пресмятания. Поради тази причина, при извършването на измерванията е нужно програмата да се стартира многократно и резултатите да се усреднят.



392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434

Optimization end ...

Fitness: -11264.649634501724

MACMINI:libs todorbalabanov\$

Въпреки, че броят извиквания варира, то този индикатор е достатъчно приемлив ориентир с какъв порядък предстоящите оптимизации ще подобрят времето за изпълнение на целевата функция.



400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442

Optimization end ...

Fitness: -11299.57511108848

MACMINI:libs todorbalabanov\$ █

Още при първото стартиране профилиращия инструмент показва, че повече от 80% от времето използвано от програмата преминава точно в изчисляване на целевата функция.



Applications

Ellipses-Image-Approximator-all.jar (pid 2611) x

OverviewMonitorThreadsSamplerProfiler

Ellipses-Image-Approximator-all.jar (pid 2611)

Profiler

Settings

Profile: CPUMemoryJDBCStop

Status: profiling running (40 methods instrumented)

Profiling results

Results:View:Collected data: Snapshot

| Name | Total Time | Total Time (CPU) | Invocations |
|--|-------------------|-------------------|-------------|
| main | 27,768 ms (100%) | 23,325 ms (100%) | 68,561 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.fitness () | 22,332 ms (80.4%) | 19,086 ms (81.8%) | 5 |
| eu.veldsoft.ellipses.image.approximator.Util.closestColor (java.awt.Color, java.util.Vector) | 5,266 ms (19%) | 4,114 ms (17.6%) | 63,314 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 226 ms (0.8%) | 183 ms (0.8%) | 1 |
| eu.veldsoft.ellipses.image.approximator.ColorCoordinatesComparator.compare (Object, Object) | 42.9 ms (0.2%) | 41.3 ms (0.2%) | 5,239 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 35.3 ms (0.1%) | 35.0 ms (0.2%) | 1 |
| eu.veldsoft.ellipses.image.approximator.Util.drawEllipses (java.awt.image.BufferedImage, java.util.List) | 33.1 ms (0.1%) | 32.9 ms (0.1%) | 1 |

На свой ред, целевата функция е съставена от група извиквания на други функции, където функция за сортиране отнема значителна част при това пресмятане.



Applications

Ellipses-Image-Approximator-all.jar (pid 2611) x

OverviewMonitorThreadsSamplerProfiler

Ellipses-Image-Approximator-all.jar (pid 2611)

Profiler

Settings

Profile:CPUMemoryJDBCStop

Status: profiling running (40 methods instrumented)

Profiling results

Results:View:Collected data:Snapshot

| Name | Total Time | Total Time (CPU) | Invocations |
|--|-------------------|-------------------|-------------|
| main | 54,472 ms (100%) | 46,400 ms (100%) | 68,566 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.fitness () | 49,035 ms (90%) | 42,161 ms (90.9%) | 10 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.sort () | 47,316 ms (86.9%) | 40,622 ms (87.5%) | 10 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 1,278 ms (2.3%) | 1,121 ms (2.4%) | 9 |
| eu.veldsoft.ellipses.image.approximator.Util.drawEllipses (java.awt.image.BufferedImage, java.util.List) | 355 ms (0.7%) | 332 ms (0.7%) | 9 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 84.1 ms (0.2%) | 83.4 ms (0.2%) | 9 |
| Self time | 1.33 ms (0%) | 1.28 ms (0%) | 10 |
| eu.veldsoft.ellipses.image.approximator.Util.closestColor (java.awt.Color, java.util.Vector) | 5,266 ms (9.7%) | 4,114 ms (8.9%) | 63,314 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 226 ms (0.4%) | 183 ms (0.4%) | 1 |
| eu.veldsoft.ellipses.image.approximator.ColorCoordinatesComparator.compare (Object, Object) | 42.9 ms (0.1%) | 41.3 ms (0.1%) | 5,239 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 35.3 ms (0.1%) | 35.0 ms (0.1%) | 1 |
| eu.veldsoft.ellipses.image.approximator.Util.drawEllipses (java.awt.image.BufferedImage, java.util.List) | 33.1 ms (0.1%) | 32.9 ms (0.1%) | 1 |

Сортирането се извършва на база на близостта между цветовете от 16М изображение и 16 базови цвята. Тази близост се пресмята пиксел по пиксел и това обяснява защо толкова много време бива изразходвано при това пресмятане.



Applications

Ellipses-Image-Approximator-all.jar (pid 2611) x

OverviewMonitorThreadsSamplerProfiler

Ellipses-Image-Approximator-all.jar (pid 2611)

Profiler

Settings

Profile: CPUMemoryJDBCStop

Status: profiling running (40 methods instrumented)

Profiling results

Results:View:Collected data: Snapshot

| Name | Total Time | Total Time (CPU) | Invocations |
|--|-------------------|-------------------|-------------|
| main | 81,434 ms (100%) | 69,132 ms (100%) | 68,571 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.fitness () | 75,998 ms (93.3%) | 64,892 ms (93.9%) | 15 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.sort () | 73,342 ms (90.1%) | 62,528 ms (90.4%) | 15 |
| eu.veldsoft.ellipses.image.approximator.Util.closestColor (java.awt.Color, java.util.Vector) | 70,384 ms (86.4%) | 59,785 ms (86.5%) | 1,046,503 |
| Self time | 2,631 ms (3.2%) | 2,461 ms (3.6%) | 15 |
| eu.veldsoft.ellipses.image.approximator.ColorCoordinatesComparator.compare (Object, Object) | 326 ms (0.4%) | 280 ms (0.4%) | 73,176 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 2,019 ms (2.5%) | 1,752 ms (2.5%) | 14 |
| eu.veldsoft.ellipses.image.approximator.Util.drawEllipses (java.awt.image.BufferedImage, java.util.List) | 518 ms (0.6%) | 495 ms (0.7%) | 14 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 116 ms (0.1%) | 115 ms (0.2%) | 14 |
| Self time | 1.85 ms (0%) | 1.75 ms (0%) | 15 |
| eu.veldsoft.ellipses.image.approximator.Util.closestColor (java.awt.Color, java.util.Vector) | 5,266 ms (6.5%) | 4,114 ms (6%) | 63,314 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 226 ms (0.3%) | 183 ms (0.3%) | 1 |
| eu.veldsoft.ellipses.image.approximator.ColorCoordinatesComparator.compare (Object, Object) | 42.9 ms (0.1%) | 41.3 ms (0.1%) | 5,239 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 35.3 ms (0%) | 35.0 ms (0.1%) | 1 |
| eu.veldsoft.ellipses.image.approximator.Util.drawEllipses (java.awt.image.BufferedImage, java.util.List) | 33.1 ms (0%) | 32.9 ms (0%) | 1 |

Разстоянието между два пиксела се определя по евклидова метрика за трите основни компонента при цветообразуването (червен, зелен, син).



Applications

Ellipses-Image-Approximator-all.jar (pid 2611) x

OverviewMonitorThreadsSamplerProfiler

Ellipses-Image-Approximator-all.jar (pid 2611)

ProfilerSettings

Profile:CPUMemoryJDBCStop

Status: profiling running (40 methods instrumented)

Profiling results

Results:View:Collected data:Snapshot

| Name | Total Time | Total Time (CPU) | Invocations |
|--|--------------------|--------------------|-------------|
| main | 128,924 ms (100%) | 109,338 ms (100%) | 68,579 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.fitness () | 123,487 ms (95.8%) | 105,098 ms (96.1%) | 23 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.sort () | 119,313 ms (92.5%) | 101,346 ms (92.7%) | 23 |
| eu.veldsoft.ellipses.image.approximator.Util.closestColor (java.awt.Color, java.util.Vector) | 114,613 ms (88.9%) | 97,128 ms (88.8%) | 1,689,721 |
| Self time | 59,458 ms (46.1%) | 50,365 ms (46.1%) | 1,689,721 |
| eu.veldsoft.ellipses.image.approximator.EuclideanColorComparator.distance (int, int) | 55,155 ms (42.8%) | 46,762 ms (42.8%) | 54,071,042 |
| Self time | 4,159 ms (3.2%) | 3,770 ms (3.4%) | 23 |
| eu.veldsoft.ellipses.image.approximator.ColorCoordinatesComparator.compare (Object, Object) | 540 ms (0.4%) | 447 ms (0.4%) | 115,830 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 3,192 ms (2.5%) | 2,800 ms (2.6%) | 22 |
| eu.veldsoft.ellipses.image.approximator.Util.drawEllipses (java.awt.image.BufferedImage, java.util.List) | 806 ms (0.6%) | 777 ms (0.7%) | 22 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 172 ms (0.1%) | 171 ms (0.2%) | 22 |
| Self time | 2.60 ms (0%) | 2.47 ms (0%) | 23 |
| eu.veldsoft.ellipses.image.approximator.Util.closestColor (java.awt.Color, java.util.Vector) | 5,266 ms (4.1%) | 4,114 ms (3.8%) | 63,314 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 226 ms (0.2%) | 183 ms (0.2%) | 1 |
| eu.veldsoft.ellipses.image.approximator.ColorCoordinatesComparator.compare (Object, Object) | 42.9 ms (0%) | 41.3 ms (0%) | 5,239 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 35.3 ms (0%) | 35.0 ms (0%) | 1 |
| eu.veldsoft.ellipses.image.approximator.Util.drawEllipses (java.awt.image.BufferedImage, java.util.List) | 33.1 ms (0%) | 32.9 ms (0%) | 1 |

По отношение на евклидовото разстояние по цвят, няма кой знае каква оптимизация, която да се направи на този етап. Следователно, ще се разглеждат някои от функциите, които са по-високо в дървото на извикванията.



```
EuclideanColorComparator.java
1 package eu.veldsoft.ellipses.image.approximator;
2
3 class EuclideanColorComparator implements ColorComparator {
4     private double deltaRed;
5     private double deltaGreen;
6     private double deltaBlue;
7
8     public double distance(int a, int b) {
9         deltaRed = ((a >> 16) & 0xFF) - ((b >> 16) & 0xFF);
10        deltaGreen = ((a >> 8) & 0xFF) - ((b >> 8) & 0xFF);
11        deltaBlue = (a & 0xFF) - (b & 0xFF);
12
13        return Math.sqrt(deltaRed * deltaRed + deltaGreen * deltaGreen
14            + deltaBlue * deltaBlue);
15    }
16 }
17
```


Най-близък цвят се търси след извъртане на цикъл, но също не предлага кой знае какви възможности за оптимизация. Това е причина да се потърси още по-нагоре в йерархията от извикване на функции.




```

40     for (size = 0; size < aPixels.length && size < bPixels.length; size++) {
41         sum += euclidean.distance(aPixels[size] & 0xFFFFFF,
42             bPixels[size] & 0xFFFFFF);
43     }
44
45     return sum / size;
46 }
47
48 static Color closestColor(Color color, Vector<Color> colors) {
49     if (colors.size() <= 0) {
50         return color;
51     }
52
53     Color best = colors.get(0);
54     for (Color candidate : colors) {
55         if (euclidean.distance(color.getRGB() & 0xFFFFFF,
56             candidate.getRGB() & 0xFFFFFF) < euclidean.distance(
57                 color.getRGB() & 0xFFFFFF,
58                 best.getRGB() & 0xFFFFFF)) {
59             best = candidate;
60         }
61     }
62
63     return best;
64 }
65
66 static double alphaLevel(BufferedImage image, Vector<Color> colors) {
67     double level = 0;
68
69     int pixels[] = image.getRGB(0, 0, image.getWidth(), image.getHeight(),
70         null, 0, image.getWidth());
71     for (int i = 0; i < pixels.length; i++) {
72         for (Color color : colors) {
73             if (pixels[i] == color.getRGB()) {
74                 level++;
75                 break;
76             }

```

При функцията за сортиране първо се забелязва, че хистограмата за използваните цветове се изчислява многократно. В това има логика, когато на оптимизация подлежат и 16-те основни цвята, но в настоящата реализация тази функционалност още не е реализирана. Поради тази причина е удачно хистограмата да се изчисли само веднъж, при зареждане на изображението.



```
46
47 private List<Ellipse> sort() {
48     /*
49      * Calculate the most used colors from the original picture.
50      */
51     Map<String, Integer> histogram = new HashMap<String, Integer>();
52     int pixels[] = image.getRGB(0, 0, image.getWidth(), image.getHeight(),
53         null, 0, image.getWidth());
54     for (int i = 0; i < pixels.length; i++) {
55         Color color = new Color(pixels[i]);
56         color = Util.closestColor(color, colors);
57         // TODO It is not clear that mapping to the closest color is the
58         // correct way of histogram calculation.
59
60         if (histogram.containsKey(color.toString()) == false) {
61             histogram.put(color.toString(), 1);
62         } else {
63             histogram.put(color.toString(),
64                 histogram.get(color.toString()) + 1);
65         }
66     }
67
68     /*
69      * Sort according color usage and x-y coordinates. The most used colors
70      * should be drawn first.
71      */
72     Util.usage.setHistogram(histogram);
73     List<Ellipse> list = new ArrayList<Ellipse>(getRepresentation());
74
75     /*
76      * Sorting have sense if only there is a histogram calculated.
77      */
78     Collections.sort(list, Util.usage);
79
80     return list;
81 }
82
```


Второто, което се забелязва е, че цветовете се обработват като обекти, а базовите цветове се съхраняват във Vector, вместо в масив. Работата с обекти значително забавя изчисленията, особено при факта, че обекта за изображението дава достъп до пикселите под формата на масив от цели числа.



```
46
47 private List<Ellipse> sort() {
48     /*
49      * Calculate the most used colors from the original picture.
50      */
51     Map<String, Integer> histogram = new HashMap<String, Integer>();
52     int pixels[] = image.getRGB(0, 0, image.getWidth(), image.getHeight(),
53         null, 0, image.getWidth());
54     for (int i = 0; i < pixels.length; i++) {
55         Color color = new Color(pixels[i]);
56         color = Util.closestColor(color, colors);
57         // TODO It is not clear that mapping to the closest color is the
58         // correct way of histogram calculation.
59
60         if (histogram.containsKey(color.toString()) == false) {
61             histogram.put(color.toString(), 1);
62         } else {
63             histogram.put(color.toString(),
64                 histogram.get(color.toString()) + 1);
65         }
66     }
67
68     /*
69      * Sort according color usage and x-y coordinates. The most used colors
70      * should be drawn first.
71      */
72     Util.usage.setHistogram(histogram);
73     List<Ellipse> list = new ArrayList<Ellipse>(getRepresentation());
74
75     /*
76      * Sorting have sense if only there is a histogram calculated.
77      */
78     Collections.sort(list, Util.usage);
79
80     return list;
81 }
82
```


Като начало в този обект няма да се пресмята всеки път хистограмата, а тя ще бъде изнесена за съхранение там където се съхранява обектът на изображението. В този обект ще се пази само референция на вече пресметнатата хистограма.



```

1 package eu.veldsoft.ellipses.image.approximator;
2
3 import java.awt.Color;
4
14
15 class EllipseListChromosome extends AbstractListChromosome<Ellipse>
16     implements
17         GCode {
18
19     /** The amount of simple primitives can float, but it has average size. */
20     private static int AVERAGE_LENGTH = 0;
21
22     /** Reference to the original image. */
23     private BufferedImage image = null;
24
25     /** Reference to the original image histogram. */
26     Map<String, Integer> histogram = null;
27
28     /** Reference to the set of reduced colors. */
29     private Vector<Color> colors = null;
30
31     /**
32      * @return The average length of the chromosome.
33      */
34     public static int AVERAGE_LENGTH() {
35         return AVERAGE_LENGTH;
36     }
37
38     /**
39      * @param averageLength
40      *         The average length of the chromosome.
41      */
42     public static void AVERAGE_LENGTH(int averageLength) {
43         if (averageLength < 0) {
44             averageLength = 0;
45         }
46
47         AVERAGE_LENGTH = averageLength;

```

Използването на референция към хистограмата се отразява в конструкторите на класа.




```
86         return list;
87     }
88
89     public EllipseListChromosome(Ellipse[] representation, BufferedImage image,
90         Map<String, Integer> histogram, Vector<Color> colors)
91         throws InvalidRepresentationException {
92         super(representation);
93         this.image = image;
94         this.histogram = histogram;
95         this.colors = colors;
96
97         checkValidity(getRepresentation());
98     }
99
100    public EllipseListChromosome(List<Ellipse> representation,
101        BufferedImage image, Map<String, Integer> histogram,
102        Vector<Color> colors) throws InvalidRepresentationException {
103        super(representation);
104        this.image = image;
105        this.histogram = histogram;
106        this.colors = colors;
107
108        checkValidity(getRepresentation());
109    }
110
111    public EllipseListChromosome(List<Ellipse> representation, boolean copy,
112        BufferedImage image, Map<String, Integer> histogram,
113        Vector<Color> colors) throws InvalidRepresentationException {
114        super(representation, copy);
115        this.image = image;
116        this.histogram = histogram;
117        this.colors = colors;
118
119        checkValidity(getRepresentation());
120    }
121
122    // static int counter = 0;
```

На този етап от проектирането на модулите е логично хистограмата да бъде съхранявана като обект там където се съхранява и оригиналното пълноцветно изображение.




```

1 package eu.veldsoft.ellipses.image.approximator;
2
3 import java.awt.Color;
4
29
30 public class Main {
31     private static BufferedImage original = null;
32     private static Map<String, Integer> histogram = new HashMap<String, Integer>();
33     private static Vector<Color> colors = new Vector<Color>();
34
35     private static Population doGeneticAlgorithmOptimization(Population initial,
36         double crossoverRate, double mutationRate, int tournamentArity,
37         int time) {
38         GeneticAlgorithm algorithm = new GeneticAlgorithm(
39             new InstructionsCrossover(), crossoverRate,
40             new RandomEllipsesMutation(original, histogram, colors),
41             mutationRate, new TournamentSelection(tournamentArity));
42
43         Population optimized = algorithm.evolve(initial,
44             new FixedElapsedTime(time));
45
46         return optimized;
47     }
48
49     private static void doAntColonyOptimization(List<Ellipse> ellipses,
50         int antsAmount, int repetitions, int iterations) {
51         /*
52          * For ant colony graph.
53          */
54         double neighbours[][] = new double[ellipses.size()][ellipses.size()];
55         for (int i = 0; i < ellipses.size(); i++) {
56             for (int j = 0; j < ellipses.size(); j++) {
57                 /*
58                  * Node will not be connected to itself.
59                  */
60                 if (i == j) {
61                     neighbours[i][j] = 0;
62                     continue;

```

Веднага след зареждане на входното изображение може да се изпълни изчисляване на хистограмата и това действие да бъде еднократно за цялото време през което програмата работи. Ако се добави оптимизация на 16-те базови цвята, хистограмата ще трябва да се преизчислява.



```
EllipseListChromosome.java  Main.java 33
318     system.out.println();
319     (new HelpFormatter()).printHelp(
320         "java -jar Ellipses-Image-Approximator-all.jar", options,
321         true);
322     System.out.println();
323     System.exit(0);
324 }
325
326     /* Read input image. */
327     original = ImageIO.read(input);
328
329     /*
330      * Calculate the most used colors from the original picture.
331      */
332     int pixels[] = original.getRGB(0, 0, original.getWidth(),
333         original.getHeight(), null, 0, original.getWidth());
334     for (int i = 0; i < pixels.length; i++) {
335         Color color = new Color(pixels[i]);
336         color = Util.closestColor(color, colors);
337         // TODO It is not clear that mapping to the closest color is the
338         // correct way of histogram calculation.
339
340         if (histogram.containsKey(color.toString()) == false) {
341             histogram.put(color.toString(), 1);
342         } else {
343             histogram.put(color.toString(),
344                 histogram.get(color.toString()) + 1);
345         }
346     }
347
348     /* Associate output folder. */
349     File output = null;
350     if (commands.hasOption("output") == true) {
351         output = new File(commands.getOptionValue("output"));
352     } else {
353         output = new File(".");
354     }
355
```


След така направените промени, времето за сортиране драстично намалява. Съвсем логично, следва промяна на функцията консумираща най-много време и в този случай това е пресмятането на евклидово разстояние между две растерни изображения.



Applications

Ellipses-Image-Approximator-all.jar (pid 3827) x

OverviewMonitorThreadsSamplerProfiler

Ellipses-Image-Approximator-all.jar (pid 3827)

Profiler

Settings

Profile:

CPU

Memory

JDBC

Stop

Status: application terminated

Profiling results

Results:

View:

Collected data:

Snapshot

| Name | Total Time | Total Time (CPU) | Invocations |
|--|-------------------|-------------------|-------------|
| main | 19,781 ms (100%) | 16,715 ms (100%) | 88 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.fitness () | 17,878 ms (90.4%) | 15,191 ms (90.9%) | 83 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 12,843 ms (64.9%) | 10,758 ms (64.4%) | 83 |
| Self time | 6,722 ms (34%) | 5,760 ms (34.5%) | 83 |
| eu.veldsoft.ellipses.image.approximator.EuclideanColorComparator.distance (int, int) | 6,120 ms (30.9%) | 4,998 ms (29.9%) | 6,150,632 |
| eu.veldsoft.ellipses.image.approximator.Util.drawEllipses (java.awt.image.BufferedImage, java.util.List) | 3,094 ms (15.6%) | 2,803 ms (16.8%) | 83 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.sort () | 1,300 ms (6.6%) | 1,040 ms (6.2%) | 83 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 606 ms (3.1%) | 580 ms (3.5%) | 83 |
| Self time | 32.9 ms (0.2%) | 8.27 ms (0%) | 83 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.toGCode (eu.veldsoft.ellipses.image.approximator.GC) | 1,220 ms (6.2%) | 1,040 ms (6.2%) | 1 |
| eu.veldsoft.ellipses.image.approximator.Util.distance (java.awt.image.BufferedImage, java.awt.image.BufferedImage) | 491 ms (2.5%) | 313 ms (1.9%) | 1 |
| eu.veldsoft.ellipses.image.approximator.Util.writeSolution (java.awt.image.BufferedImage, java.util.List, String) | 126 ms (0.6%) | 108 ms (0.6%) | 1 |
| eu.veldsoft.ellipses.image.approximator.Util.alphaLevel (java.awt.image.BufferedImage, java.util.Vector) | 48.1 ms (0.2%) | 46.1 ms (0.3%) | 1 |
| eu.veldsoft.ellipses.image.approximator.EllipseListChromosome.getSortedEllipses () | 16.9 ms (0.1%) | 14.9 ms (0.1%) | 1 |

Оптимизацията на програмния код е с последен приоритет в работния процес, тъй като вярното пресмятане е много по-важно от бързото пресмятане. Безотказната работа е много по-важна от грешното пресмятане. А компилиращият се софтуер е неотменно условие въобще за извършване на доставка до крайния клиент.



