cs380su21-meta.sg

<u>Dashboard</u> / My courses / <u>cs380su21-meta.sg</u> / <u>26 July - 1 August</u> / <u>Assignment 12 (Genetic Algorithm)</u>

Grade

Reviewed on Thursday, 5 August 2021, 3:43 AM by Automatic grade

grade: 100.00 / 100.00

Assessment report **%** [-]

[±]Summary of tests

Submitted on Thursday, 5 August 2021, 3:43 AM (Download)

functions.cpp

```
\file functions.cpp
  \author Vadim Surov, Goh Wei Zhe
4 \par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
   \par Course: CS380
  \par Section: B
   \par Programming Assignment 12
8 \date 4-8-2021
   \brief
10 This file has declarations and definitions that are required for submission
12 #include "functions.h"
13
14 namespace AI
15 - {
16
18 } // end namespace
```

functions.h

```
2 \file functions.h
    \author Vadim Surov, Goh Wei Zhe
 4
     \par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
    \par Course: CS380
    \par Section: B
 6
    \par Programming Assignment 12
 8
    \date 4-8-2021
 9
    This file has declarations and definitions that are required for submission
 10
 11
 12
    #ifndef FUNCTIONS H
 13
    #define FUNCTIONS_H
 14
 15
    #include <iostream>
    #include <sstream>
 16
 17
    #include <vector>
 18
    #include <string>
 19
    #include <cmath>
 20
 21
    #include "data.h"
 22
 23
    #define UNUSED(x) (void)x;
 24
 25
    template<typename Gene>
    struct Fitness_Accumulate
 26
 27 ₹ {
         *************************
 28 -
 29
        Calculates the sum of all genes in a chromosome
 30
 31
        \param genes
        A vector of genes of type Gene
 32
 33
 34
 35
        Returns the sum of all genes in a chromosome
 36
 37
        int operator()(const std::vector<Gene>& genes) const
 38 *
 39
           int s = 0;
 40
 41
           for (auto g : genes)
 42 -
 43
              s += g.getValue();
 44
 45
 46
           return s;
 47
 48
    };
 49
    template<typename Gene>
 50
    struct Fitness_Nbits
 52 - {
        53 🕶
 54
 55
        Calculates a percentage that indicates the fitness of a particular
        \hbox{chromosome into a particular solution. Fittest chromosome has all genes}\\
 56
 57
        equal to 1.
 58
 59
        \param genes
 60
        A vector of genes of type Gene
 61
 62
        \return
 63
        Returns the percentage.
                    64
 65
        int operator()(const std::vector<Gene>& genes) const
 66 🔻
 67
           if (genes.size())
 68
 69
               int s = 0;
 70
 71
 72
               for (auto g : genes)
 73
                  s += g.getValue();
 74
 75
               return (100 * s) / genes.size();
 76
 77
 78
           return 0;
 79
 80
    };
 81
    template<typename Gene>
 82
     struct Fitness_8queens
 84 🔻 {
 85 🔻
         \brief
 86
 87
        Calculates a measure in percentage that indicates the fitness of a
 88
        particular chromosome into a particular solution of 8 queens problem.
 89
 90
        \param genes
 91
        A vector of genes of type Gene
 92
 93
        \return
 94
        Returns the percentage.
        95
 96
        int operator()(const std::vector<Gene>& genes) const
 97 -
 98
            int size = genes.size();
 99
           int max = (size - 1) * size / 2;
100
101
           //Check horizontal threats
102
           int counter = 0;
103
           for (int j = 0; j < size - 1; ++j)
104
105
               for (int i = j + 1; i < size; ++i)
106
107 -
                  int dy = genes[j].getValue() - genes[i].getValue();
108
```

```
109
                    if (dy == 0)
110
                        counter++;
111
112
113
114
115
             //Check diagronal threats
116
             for (int j = 0; j < size - 1; ++j)
117 🔻
                 for (int i = j + 1; i < size; ++i)
118
119 🔻
120
                     int dx = i - j;
121
                     int dy = std::abs(genes[j].getValue() - genes[i].getValue());
122
123
                     if (dy == dx)
124
                        counter++;
125
126
127
             return std::abs(100*(max - counter) / max);
128
129
     };
130
131
132
    namespace AI
133 - {
134
         // Crossover methods for the genetic algorithm
         enum CrossoverMethod { Middle, Random };
135
136
137
         // Simplest gene seeding class/function
138
         struct Seed
139 🔻
             int operator()(int p = 0) const
140
141
142
                 return p;
143
144
145
         // Gene seeding class/function with a fixed value
146
         template<int Val = 0>
147
148
         struct Seed_Value
149 ¬
             int operator()(int /* p */ = 0) const
150
151
             .{
152
                 return Val;
153
154
         };
155
156
         template<int Max>
157
         struct Seed_Random
158 🔻
             /************************
159 🕶
160
             \brief
161
             Gene random seeding function
162
163
             \return
164
             Returns random seed value
             165
             int operator()(int /* p */ = 0) const
166
167 ¬
             }
                 return static_cast<int>(std::floor(std::rand() % Max));
168
169
170
         };
171
172
         // Gene class
173
         template<typename T = int, typename S = Seed>
174
         class Gene
175 🔻
176
             T value;
177
178
179
             Gene(int p = 0): value{ S()(p) }{}
180
181
             T getValue() const
182 🤻
                 return value;
183
184
185
186
             void setValue(T v)
187
                 value = v;
188
189
190
             friend std::ostream& operator<<(std::ostream& os, const Gene& rhs)
191
192 🔻
             {
193
                 os << rhs.value;
194
                 return os;
195
196
         };
197
198
         // Chromosome class
199
         template<typename Gene, typename Fitness, size_t Size>
         class Chromosome
200
201 -
202
             std::vector<Gene> genes;
203
             int fitness;
204
205
         public:
206
207
             using gene_type = Gene;
208
209
             static const size_t size = Size;
210
211
             Chromosome()
                 : genes(Size), fitness{ Fitness()(genes) }{}
212
213
214
             std::vector<Gene>& getGenes()
215 -
             {
216
                 return genes:
```

```
217
              }
218
219
              void setGenes(const std::vector<Gene>& v)
220 -
221
                  genes = v;
222
                  fitness = Fitness()(genes);
223
224
225
              Gene getGene(size_t i) const
226 🔻
227
                  return genes[i];
228
229
230
              void setGene(size_t i, const Gene& v)
231 7
232
                  genes[i] = v;
                  fitness = Fitness()(genes);
233
234
235
236
              int getFitness() const
237 ¬
                  return fitness;
238
239
240
241
              // Select a random mutation point and change
242
              // gene at the mutation point
243
              void randomMutation()
244 🔻
              {
245
                  setGene(std::rand() % Chromosome::size, Gene());
246
247
248
              // Copy genes from a source
249
              void copyGenesFrom(Chromosome& src)
250
              {
                  std::copy(src.genes.begin(), src.genes.end(), genes.begin());
251
252
                  fitness = Fitness()(genes);
253
254
              friend std::ostream& operator<<(std::ostream& os,</pre>
255 -
256
                  const Chromosome& rhs)
257 -
258 🔻
                  for (auto it = rhs.genes.begin(); it != rhs.genes.end(); ++it)
259
                   os << *it << (it + 1 != rhs.genes.end() ? "," : "");
260
                  os << "]=" << rhs.fitness;
261
262
                  return os;
263
264
          };
265
266
          // Individual class
267
          template<typename Chromosome>
          class Individual
268
269
270
              Chromosome chromosome;
271
272
          public:
273
274
              using chromosome_type = Chromosome;
275
              using gene_type = typename Chromosome::gene_type;
276
277
              Individual()
278
                  : chromosome{ }{}
279
280
              Chromosome& getChromosome()
281 *
282
                  return chromosome;
283
284
285
              std::vector<gene_type>& getGenes()
286
287
                  return chromosome.getGenes();
288
289
290
              void setGenes(const std::vector<gene_type>& v)
291
              {
292
                  chromosome.setGenes(v);
293
294
              gene_type getGene(size_t i) const
295
296 *
297
                  return chromosome.getGene(i);
298
              }
299
300
              void copyGenesFrom(Individual& individual)
301 -
302
                  chromosome.copyGenesFrom(individual.chromosome);
303
304
              void setGene(size_t i, gene_type gene)
305
306
                  chromosome.setGene(i, gene);
307
308
309
310
              int getFitness() const
311 -
312
                  return chromosome.getFitness();
313
314
              friend std::ostream& operator<<(std::ostream& os, Individual& rhs)</pre>
315
316 -
                  os << rhs.chromosome;</pre>
317
                  return os;
318
319
320
          };
321
322
          // Population class
323
          template<typename Individual>
324
          class Population
```

```
325 🔻
        {
326
            std::vector<Individual> individuals;
           Individual* fittest;
327
328
        public:
329
330
           Population(size_t size = 0)
331
               : individuals{ }, fittest{ nullptr }
332 🔻
               if (size)
333
334 🔻
               {
                  individuals.resize(size);
335
336
                  updateFittest();
337
338
339
340
           size_t getSize() const
341 🔻
342
               return individuals.size();
343
344
345
           Individual& getIndividual(size_t i)
346 *
347
               return individuals[i];
348
349
           Individual* getFittest() const
350
351 🔻
           {
352
               return fittest;
353
354
            /**********************
355 🔻
356
357
           Update Fittest function to update fitness if there is a higher fitness
358
           value than previous
359
360
            \return
361
           None
            362
363
           void updateFittest()
364 -
               if (this->individuals.size())
365
366 ₹
367
                  this->fittest = &individuals[0];
368
                   for (size_t i = 1; i < this->individuals.size(); ++i)
369
370 -
                      if (individuals[i].getFitness() > fittest->getFitness())
371
372 -
                          this->fittest = &individuals[i];
373
374
375
376
               }
377
378
                  this->fittest = nullptr;
379
380
           friend std::ostream& operator<<(std::ostream& os, Population& rhs)</pre>
381
382 -
               os << " = " << rhs.getFittest()->getFitness() << std::endl;</pre>
383
               for (size_t i = 0; i < rhs.getSize(); ++i)
    os << " " << i << ':' << rhs.getIndividual(i) << std::endl;</pre>
384
385
386
               return os;
387
388
        };
389
390
        // Genetic Algorithm class
391
        template<typename Individual>
392
        class GeneticAlgorithm
393 🔻
394
            Population<Individual>* population;
395
           int generation;
396
397
398
           GeneticAlgorithm(): population{ nullptr }, generation{ 0 }{}
399
            /*******************
400 -
401
402
           Destructor for class GeneticAlgorithm
403
            \return
404
405
406
            ***********************
407
408 -
               if(population)
409
                  delete population;
410
411
412
            413 🔻
414
            \brief
415
           Getter function for getFittest()
416
417
            \return
           Returns a pointer to Class Individual
418
                      419
420
           Individual* getFittest() const
421 🕶
               return this->population->getFittest();
422
423
424
            425 🔻
            \brief
426
427
           Implementation of the Roulette Wheel Selection. The probability of an
428
           individual to be selected is directly propoertional to its fitness.
429
430
            \param sizeOfPopulation
431
           Population size
432
```

```
433
             \return
434
             Returns a pointer to class Population
             435
436
             Population<Individual>* selection(size_t sizeOfPopulation)
437 🕶
438
                if (!this->population)
439 -
                    this->setPopulation(new Population<Individual>
440
                        (sizeOfPopulation));
441
                Population<Individual>* newGeneration =
442
                    new Population<Individual>(sizeOfPopulation);
443
444
445
                //Play roulette
446
                int sum_fitness = 0;
447
448
                for (size_t i = 0; i < sizeOfPopulation; ++i)</pre>
449
                    sum_fitness += this->population->getIndividual(i).getFitness();
450
                for (size_t i = 0; i < sizeOfPopulation; ++i)</pre>
451
452
453
                    int random = std::rand() % sum_fitness;
454
455
                    size_t j = 0;
456
457
                    while(j < sizeOfPopulation)</pre>
458
459
                        random -= this->population->getIndividual(j).getFitness();
460
461
                        if (random <= 0)
462
                            break;
463
464
                        ++j;
465
466
467
                    //Copy genes
                    newGeneration->getIndividual(i).copyGenesFrom
468
469
                     (this->population->getIndividual(j));
470
471
                newGeneration->updateFittest();
472
473
                return newGeneration;
474
475
             /**********************
476 -
477
478
             Crossover parents genes function
479
480
             \param newGeneration
481
             A pointer to class Population
482
483
             \param crossoverMethod
484
             An enumeration
485
486
             \return
487
             **********************************
488
489 -
             void crossover(Population<Individual>* newGeneration,
490
                CrossoverMethod crossoverMethod)
491
492
                int crossOverPoint = 0;
493
494
                if (crossoverMethod == CrossoverMethod::Middle)
495
                    crossOverPoint = Individual::chromosome_type::size / 2;
                else if (crossoverMethod == CrossoverMethod::Random)
496
497
                    crossOverPoint =
                    std::rand() % Individual::chromosome_type::size;
498
499
                for (size_t j = 0; j < newGeneration->getSize() - 1; j+=2)
500
501
                     //Swap values among pairs
502
503
                    for (int i = 0; i < crossOverPoint; ++i)</pre>
504 7
                        auto t = newGeneration->getIndividual(j + 1).getGene(i);
505
506
507
                        newGeneration->getIndividual(j + 1).setGene(i,
508
                            newGeneration->getIndividual(j).getGene(i));
509
                        newGeneration->getIndividual(j).setGene(i, t);
510
511
                    }
512
513
514
515
516
             \brief
             Mutation of genes under a random probability
517
518
519
             \param newGeneration
520
             A pointer to class Population
521
             \param mutationProbability
522
             Percentatage of mutation
523
524
525
             \return
526
             None
             527
             void mutation(Population<Individual>* newGeneration,
528 -
                int mutationProbability)
529
530 -
531
                int sizeOfPopulation = newGeneration ?
                                       newGeneration->getSize() : 0;
532
533
534
                //Select a random mutation point and flip
535
                //gene at the mutation point
536
                for (int j = 0; j < sizeOfPopulation; ++j)</pre>
537 🕶
                    if (std::rand() % 100 < mutationProbability)</pre>
538
539
                        newGeneration->getIndividual(j).getChromosome().
                        randomMutation().
```

```
541
                }
542
543
544 -
545
546
            Replace existing population with a new generation
547
548
             \param newGeneration
549
            A pointer to class Population
550
551
             \return
552
            None
             *******************************
553
554
            void setPopulation(Population<Individual>* newGeneration)
555 7
            {
556
                if (population)
557
                    delete population;
558
559
                this->population = newGeneration;
560
                this->population->updateFittest();
561
562
             563 *
564
             \brief
            Start the search
565
566
567
             \param sizeOfPopulation
568
            Population size
569
570
             \param mutationProbability
571
            Percentage of mutation
572
573
             \param crossoverMethod
574
            An Enumeration
575
576
             \param os
577
            Output stream
578
579
             \return
580
            None
             ******************************
581
            void run(size_t sizeOfPopulation = 100, int mutationProbability = 70,
582 -
                        CrossoverMethod crossoverMethod = CrossoverMethod::Middle,
583
                        std::ostringstream* os = nullptr)
584
585 -
            {
586
                this->generation = 0;
587
                this->setPopulation(new Population<Individual>(sizeOfPopulation));
588
589
590
                //While loop unti the solution is found
591
                while (this->next(mutationProbability, crossoverMethod, os)) {}
592
593
             594 -
595
             \brief
596
            Continue the search
597
             \param mutationProbability
598
599
            Percentage of mutation
600
601
             \param crossoverMethod
602
            An Enumeration
603
604
             \param os
605
            Output stream
606
607
             \return
608
            Return true if solution found, else return false if not found.
609
610 🔻
            bool next(int mutationProbability, CrossoverMethod crossoverMethod,
611
                       std::ostringstream* os)
612 🔻
613
                if(!this->population)
                    this->setPopulation(new Population<Individual>
614 🔻
615
                        (this->population->getSize()));
616
                Individual* fittest = this->population->getFittest();
617
618
619
                if (os)
620
                    *os << *fittest;
621
                //Stop the search when either max fitness of solution or maxi limit
622
623
                 //tor generation achieved
                if (this->population->getFittest()->getFitness() == 100 | |
624
                    this->generation > 10000)
625
                    return false;
626
627
                Population<Individual>* newGeneration =
628
                    this->selection(this->population->getSize());
629
630
631
                //Recombination: creates new individuals by taking the chromosomes
                //from the fittest members of the population and modifying these
632
                //chromosomes using crossover and/or mutation.
633
                this->crossover(newGeneration, crossoverMethod);
634
635
                this->mutation(newGeneration, mutationProbability);
636
                //Set population with new generation
637
638
                this->setPopulation(newGeneration);
639
                this->generation++;
640
641
                return true;
642
643
         };
644
645
     } // end namespace
646
     #endif
647
```

■ Showcase: Genetic Algorithm

Jump to... \$

Example questions for Genetic Algorithm with answers for quiz 4 ►

<u>VPL</u>

You are logged in as Wei Zhe GOH (Log out) cs380su21-meta.sg
Data retention summary
Get the mobile app