G53IDS - User Manual

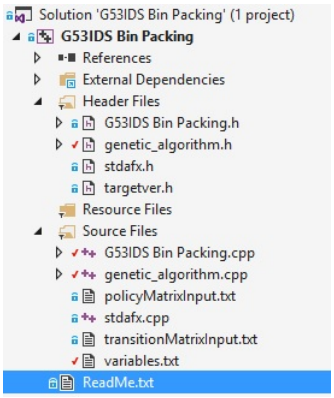
Akbar Mohammed - 4260137

How To Run The Software

This software is designed to work on Visual Studio 2015 and 2017. To run it, open the solution file from Visual Studio and click run without debugging. This program is designed to run in the console.

Program File Structure

The program has the the following file structure. This is relevant as it shows the locations of the files that have to be edited in order to run this software. It also gives insight into the structure of this program.



The file `genetic_algorithm.cpp` runs the genetic algorithm. The file `G53IDS Bin Packing.cpp` is the bin-packer which is called from the `genetic_algorithm.cpp` file from the evaluate function.

The following files need to be edited in order to use the program effectively. The way to edit these files are discussed below.

- `variables.txt`
- `genetic_algorithm.h`
- `policyMatrixInput.txt`
- `transitionMatrixInput.txt`

Changing Variables

The variables that can be edited include the MCBP problem instance, the precision level which represents the scale of the probabilities, minimum and maximum and policy matrix, number of items in the item stream and policy matrix to compare against.

To describe instances of the bin packing problem, the notation MCBP(C, S_{min}, S_{max}, N) is used, such that:

- Bin capacity is represented by C where bin capacity is an integer greater than or equal to S_{max}.
- Minimum and maximum item sizes present in the item stream and that will be packed is S_{min} and S_{max} respectively and are integers. Also, $(S_{min}, S_{max}) > 0$.
- Number of items in the item stream is represented by N where $N > 0$ and N is an integer.

The variables can be edited in the `variables.txt` file as seen below.

```
//to adjust the precision level of the probabilities listed in the Markov Chain
100
//set the starting state of the Markov Chain
10
//bin capacity, note: bincapacity must be greater than or equal to the largest item size within the stream of items
20
//minimum item size
5
//maximum item size
10
//number of items to place
1000
//minimum policy matrix score
1
//maximum policy matrix score
5
//choose whether to evolve policy matrix only (set value to 1) or evolve both the policy and transition matrices (set value to 0)
1
//select algorithm to generate the policy matrix to compare against, if above has been set to 0
//input 1 to compare against first fit
//input 2 to compare against best fit
//input 3 to compare against worst fit
//input 4 to read policy matrix from a file
//otherwise leave blank
```

Comments have been added in the above file for clarity and to make it user friendly.

If the value 0 has been set for the second last variable in the variables file, a policy matrix has to input. This is because the policy matrix input is used to compare against the policy matrix generated through the extended algorithm in the program. The transition and policy matrix pair are evolved by the program in this situation, if the variable is set to 0.

To input the policy matrix, input it in the file, `policyMatrixInput.txt`. This is an example of the policy matrix input:

```
5   3 0 0 0 0 0
6   1 5 0 0 0 0
7   1 1 4 0 0 0
8   1 1 1 2 0 0
9   1 1 1 1 3 0
10  1 1 1 1 2 2
11  2 3 1 1 3 2
12  4 4 2 1 1 1
13  1 2 4 2 1 1
14  2 4 2 2 1 1
15  1 5 3 2 2 1
16  0 0 0 0 0 0
17  0 0 0 0 0 0
18  0 0 0 0 0 0
19  0 0 0 0 0 0
20  2 2 2 2 2 2
    5 6 7 8 9 10
```

This is the policy matrix for UBP(20,5,10) which was obtained from the research paper "Policy Matrix Evolution for Generation of Heuristics" for comparison. Note that the axes have to be labelled. The vertical, y-axis represents the residual capacity and the horizontal, x-axis represents item size.

Otherwise if the value 1 has been set for the second last variables in the variables file, a transition matrix has to be input. The transition matrix is the program's internal representation of the Markov Chain. When producing the item stream, the transition matrix is used to traverse through the Markov Chain and determine the subsequent item. The transition matrix is used to produce the item stream that the extended algorithm would adapt to and generate a policy matrix when the variable is set to 1.

To input the transition matrix, input in the file `transitionMatrixInput.txt`. This is an example of the policy matrix input:

```
17 17 25 10 25 6
17 17 17 17 16 16
25 10 10 5 30 20
15 15 20 20 15 15
9 10 11 55 10 3 2
33 0 33 0 34 0
```

This is the transition matrix for MCBP(20,5,10) where each value represents the probability of transitioning between states. Each row corresponds to the current state and each column corresponds to the new state. The first row and column corresponds to item size 5, second row corresponds to item size 6 and so forth. Unlike the previous matrix, there is no need to label the axes.

Note that each row needs to sum up to the precision level. In the transition matrix above, each row summed up to 100 which was the precision level set.

Changing Genetic Algorithm Parameters

The program has 4 parameters available to change. These can be changed in the `genetic_algorithm.h` file.

- Number of generations
- Population size
- Crossover
- Mutation

These variables are listed in the header file as:

```
18  # define POPSIZE 24
19  # define MAXGENS 200
20  # define PXOVER 1.0
21  # define PMUTATION 0.0175
```

The number of generations represents the number of iterations of the genetic algorithm. It has been set to 200, although this can be changed to any number greater than or equal to 1. The population size is the population size at each generation. This has been set to 24.

The crossover rate has been set to 1.0, the optimal value. However, if needed this can be changed. The mutation rate is any real number between $0 \leq mutationRate \leq 1.0$. It represents the probability that a gene is mutated. The mutation rate printed out below at the start of the program is the optimal mutation rate. This is calculated by $1/x$ is the number of active variables of the policy matrix or the number of active variables in the transition and policy matrices. The value of x depends on whether the policy matrix is being evolved or the policy and transition matrix pair is being evolved.

```
Number of variables in chromosome 176
Optimal mutation rate is 1/187 = 0.00535
```

This probability means that one gene on average will be mutated for each member of the population at each generation.

Program Output

When running the program using the variables and genetic algorithm parameters, the following is obtained. Note that 5 generations have been used. Also, the instance MCBP(20,5,10,100).

When choosing to evolve the policy matrix only, the following output was produced.

```
Number of variables in chromosome 51
Optimal mutation rate is 1/57 = 0.01754
23 April 2018 05:39:33 PM

G53IDS - Policy Matrix Bin Packing
  C++ version
  Optimisation of policy matrices and transition matrix

  Generation       Best            Average       Standard       f1       f2
  number           value           fitness       deviation      value    value

        0       0.918293        0.872308       0.0258564              0              0
        1       0.918293        0.880915       0.0233458              0              0
        2       0.918293        0.874285       0.0171278              0              0
        3       0.918293        0.884163       0.0230955              0              0
        4       0.918293        0.880329        0.018312              0              0
```

```
Remaining Capacity 5    4 0 0 0 0 0
Remaining Capacity 6    3 2 0 0 0 0
Remaining Capacity 7    4 3 4 0 0 0
Remaining Capacity 8    1 3 3 2 0 0
Remaining Capacity 9    3 3 3 3 2 0
Remaining Capacity 10   4 2 1 5 3 2
Remaining Capacity 11   5 3 5 2 3 4
Remaining Capacity 12   4 5 5 2 2 1
Remaining Capacity 13   3 2 3 3 3 1
Remaining Capacity 14   2 2 2 3 2 5
Remaining Capacity 15   2 1 2 4 1 3
Remaining Capacity 16   0 0 0 0 0 0
Remaining Capacity 17   0 0 0 0 0 0
Remaining Capacity 18   0 0 0 0 0 0
Remaining Capacity 19   0 0 0 0 0 0
Remaining Capacity 20   2 2 2 2 2 2
Items in Policy Matrix: 5 6 7 8 9 10


  Best fitness = 0.918293

SIMPLE_GA:
  Normal end of execution.

23 April 2018 05:40:08 PM
Press any key to continue . . .
```

When choosing to evolve the policy and transition matrix pair, the following output was obtained.

```
Number of variables in chromosome 87
Optimal mutation rate is 1/93 = 0.01075
23 April 2018 05:49:32 PM

G53IDS - Policy Matrix Bin Packing
  C++ version
  Optimisation of policy matrices and transition matrix

  Generation       Best            Average       Standard       f1       f2
  number           value           fitness       deviation      value    value

        0       0.0840036      -0.0680356       0.0214788       0.919231       0.835227
        1       0.0840036      -0.0611059       0.0246994       0.919231       0.835227
        2       0.0840036      -0.0680906       0.0251362       0.919231       0.835227
        3       0.0840036      -0.0704697       0.0233293       0.919231       0.835227
        4       0.0840036      -0.0803188       0.0179895       0.919231       0.835227
```

```
23 2 19 18 21 17
35 13 7 25 13 7
3 17 28 30 4 18
22 14 13 8 33 10
21 11 20 8 23 17
21 18 11 4 12 34

Remaining Capacity 5    4 0 0 0 0 0
Remaining Capacity 6    2 1 0 0 0 0
Remaining Capacity 7    5 4 2 0 0 0
Remaining Capacity 8    5 1 2 5 0 0
Remaining Capacity 9    2 4 4 2 4 0
Remaining Capacity 10   3 3 2 3 4 2
Remaining Capacity 11   5 2 3 3 1 4
Remaining Capacity 12   2 4 4 2 3 4
Remaining Capacity 13   5 4 4 3 3 5
Remaining Capacity 14   2 2 4 2 3 3
Remaining Capacity 15   5 2 2 2 2 1
Remaining Capacity 16   0 0 0 0 0 0
Remaining Capacity 17   0 0 0 0 0 0
Remaining Capacity 18   0 0 0 0 0 0
Remaining Capacity 19   0 0 0 0 0 0
Remaining Capacity 20   2 2 2 2 2 2
Items in Policy Matrix: 5 6 7 8 9 10


  Best fitness = 0.0840036

SIMPLE_GA:
  Normal end of execution.

23 April 2018 05:50:39 PM
Press any key to continue . . .
```

The values f1 and f2 are shown only if both the policy and transition matrix is being evolved. The value f1 corresponds to the fitness of the evolved matrix from the program and the value f2 corresponds to the fitness

of a policy matrix to compare against such as first-fit or best-fit or one input from a file. This is for a given transition matrix being evolved by the program. The values f1 and f2 are gained from the best member of the population, the one with the highest fitness value which is calculated through $f = f1 - f2$. The values f1 and f2 are stored in each member of the population with the genes.

There is also functionality to write the policy matrix to a file. To do this, un-comment lines 283 and 360. The file will be saved in the source directory of the program along with the MCBP instance the policy matrix is for.

Runtime

The performance of the program can be slow, depending on the instance used. When using a larger item stream, larger number of generations or a larger population size, it can take longer to evolve a heuristic. In addition, the MCBP instance used can influence the time taken for the program to run. When this program was tested and run, it took 4 hours to evolve a solution for MCBP(20,5,10) and 11 hours for MCBP(40,10,20). The length of the item stream was 500, the population size was 24 and 200 generations were used. This was to evolve the policy matrix.