GIIE Course: 546U4030 SOFT COMPUTING METHODS AND APPLICATIONS

Assignment 07 (2021) Generic GA Solver and a Binary GA Implementation for Job Assignment Problems

Since the value type of a gene in encoding a chromosome might be integer, 0-1 binary value, and real number, the generic GA solver can be designed as a template class that takes a type parameter for specifying a particular type of the gene value. Normally, we set this type parameter with variable name T. Therefore, the class name of the generic GA solver might be `GenericGASolver<T>.` Although a 2-dimensional rectangular matrix can be defined for the set of chromosomes, a list of one-dimensional chromosome array is preferred for objective evaluation of an individual chromosome. Therefore a jagged 2-D array is defined for the chromosomes; i.e.,

```
T[][] chromosomes;
```

To record the objective values and fitness values of all of the chromosomes, double arrays

`double[] objectiveValues; double[] fitnessValues;` should be defined. In addition, to record so-far-the-best solution and objective, define

```
T[] soFarTheBestSolution; double soFarTheBestObjectiveValue;
```

Note that the constructor of the `GenericGASolver<T>` class should ask for the number of genes (variables), optimization type, and the delegate that can be called to evaluate a solution of type `T[]` and return its objective value. A delegate variable is defined to link customer-supplied objective evaluation function. The objective evaluation function should take a solution (an array of type `T; T[]`) as its argument and return its objective value in `double` type. This delegate is defined as a template delegate; e.g.,

```
public delegate double ObjectiveFunction<T>( T[] solution );
```

In the generic GA class if the delegate object is

`ObjectiveFunction<T> theObjectiveFunction;` to evaluate the objective of the i-th parent chromosome will be

```
objectiveValues[i] = theObjectiveFunction( chromosomes[i] );
```

To efficiently use memory in GA evolutional computation, required data structure should be effectively designed. This assignment asks you to design a generic GA solver implementing required generic data and utility functions. Specific calculations will therefore be overridden in the derived classes.

In genetic evolution, there are three sets of chromosomes participate: the population (parents), crossovered children, and mutated children. Three integer numbers are therefore defined to record numbers of chromosomes in these sets: `populationSize`, `numberOfCrossoveredChildren`, and `numberOfMutatedChildrenbe`. The `populationSize` is specified by the user and its value is thereafter fixed throughout the evolution while `numberOfCrossoveredChildren` and `numberOfMutatedChildrenbe` are calculated and changed in run-time subject to user-specified “rates”.

Moreover, we prefer to allocate a large chunk of memory to accommodate these sets of chromosomes as a group and not to repeatedly re-allocate memory (newing) for them during the run-time. Therefore, only one set of chromosomes is allocated once, whose size is 3 times `populationSize`. The first `populationSize` chromosomes are for the population (parent). Then the following `numberOfCrossoveredChildren` chromosomes are crossovered offspring and followed by `numberOfMutatedChildrenbe` mutated children. Similarly, `double` type arrays for `objectiveValues` and `fitnessValues` are allocated with the same size of 3 times `populationSize` as the aggregated chromosomes.

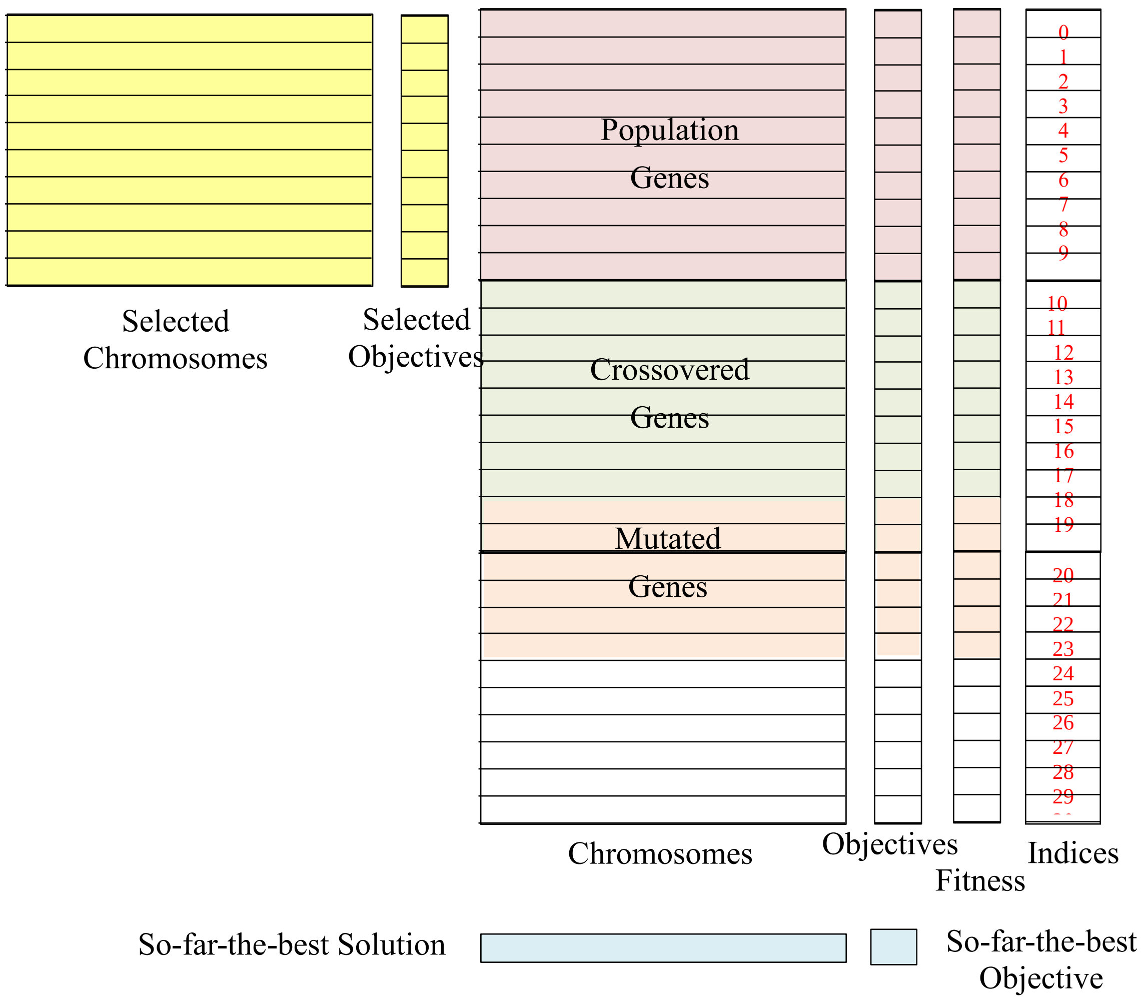


Fig. A. Suggested Data Structure for Implementation

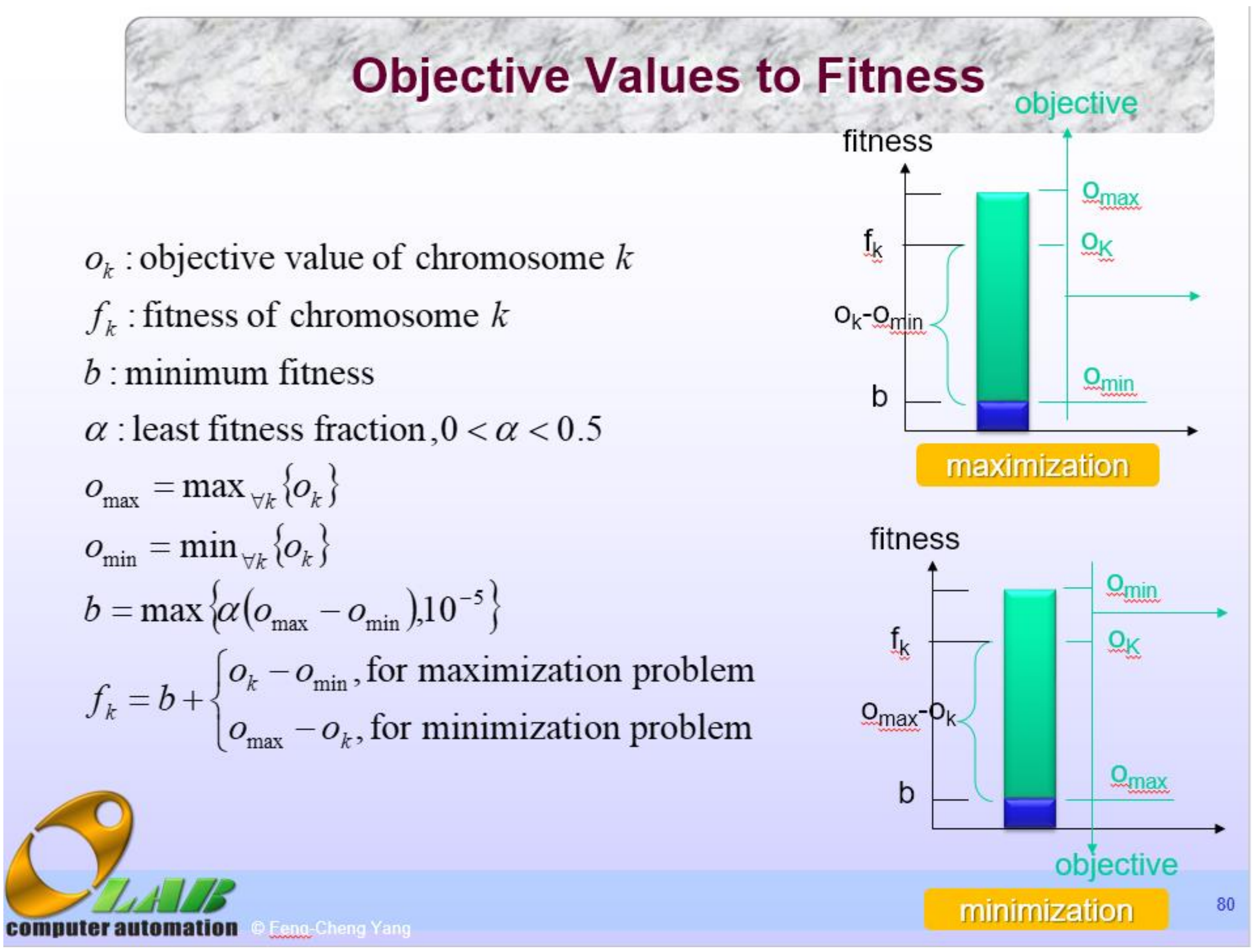
In the Selection Operation we need to pick `populationSize` better chromosomes from these three sets and overwrite the first population set to construct the new generation. In order not to obscure the candidates in the first set during the overwriting and for easier implementation, the selected chromosomes as well as their objective values will be assigned to another chunk of memory. When the selection is done, they are gene-by-gene cloned back to overwrite the original population (the first set). Therefore, a 1-D array of 1-D chromosome array (a 2-D jagged array) is defined for storing the selected chromosomes, whose size is `populationSize`; i.e.,

```
    T[][] selectedChromosomes;
```

In addition, a double array `double[] selectedObjectives;` is defined to store the objective values of the selected chromosome. The dimensions of them are exact the `populationSize`.

In your GA solver, the optimization goal is known to be either minimization or maximization. Prepare a function to transfer objective values to fitness values. Note that the fitness values of chromosomes will serve as the surviving probabilities in genetic selection. Which means the values must be positive and the worst chromosome should receive a minimal amount of probability. Fitness values are transformed from objective values, suppose o_k is the objective value of chromosome *k* and f_k is the fitness of chromosome *k*. (1) Design your transformation operations to calculate f_k from o_k and then (2) implement this function:

```
void setFitnessFromObjectives();
```



Objective Values to Fitness

o_k : objective value of chromosome k

f_k : fitness of chromosome k

b : minimum fitness

α : least fitness fraction, $0 < \alpha < 0.5$

$$o_{\max} = \max_{\forall k}\{o_k\}$$

$$o_{\min} = \min_{\forall k}\{o_k\}$$

$$b = \max\{\alpha(o_{\max} - o_{\min}), 10^{-5}\}$$

$$f_k = b + \begin{cases} o_k - o_{\min}, \text{for maximization problem} \\ o_{\max} - o_k, \text{for minimization problem} \end{cases}$$

objective
fitness
omax
fk
oK
ok-omin
b
omin
maximization
fitness
omin
fk
oK
omax-ok
b
omax
objective
minimization
computer automation
© Feng-Cheng Yang
80

In GA, *Deterministic* Selection is to choose the best (based on fitness) `populationSize` chromosomes from the parent and offspring. In contrast, *Stochastic* Selection generates a random number to perform each probability-proportional selection, where the fitness values are the selection probabilities of chromosomes.

Note that a chromosome might be selected multiple times in stochastic selection. To record the indices of the selected chromosomes, we prepare an integer array, namely `indices`, with the same size of the objective and fitness arrays. The indices of the selected `populationSize` chromosomes will be placed on the first `populationSize` locations of the `indices` array. Your mission is to (3) implement functions:

`void DeterministicSelection();` and `void StochasticSelection();`

which conduct genetic selections to set indices of selected chromosomes to the first `populationSize` positions of the `indices` array. After that (4) implement the

`void PerformSelectionOperation();`

function that calls either function `DeterministicSelection()` or `StochasticSelection()` to determine and set selected indices. Refer to the indices, gene-wise copy the genes of the selected chromosomes to `selectedChromosomes` array. Note that, their objective values are also copied to `selectedObjectives.` After selection is done, these selected gene values are gene-by-gene copied back to the first `populationSize` chromosomes in `chromosomes` as well as their objectives to `objectiveValues.`

From the user's perspective (main form) three public functions (methods) of the `GeneticGASolver <T>` should provide: `Reset(); RunOneIteration(); RunToEnd(). Reset()`is called to allocate necessary amount of memory for the user-modified model. Then virtual initialization operation `initializePopulation`() is called to randomly assign values to the variables and calculate their objectives. In `RunOneIteration()`function, genetic operations `performCrossoverOperation`(), `performMutationOperation`(), `performSelectionOperation` (), and `updateSoFarTheBestObjectiveAndSolution`() are sequentially executed.

In `updateSoFarTheBestObjectiveAndSolution()` you need to call `setFitnessFromObjectives`(). Define public properties of a GA solver that can be displayed on a PropertyGrid UI control and their values can be modified by the user.

Several termination conditions can be implemented to stop the solution evolution; for example, limit of iterations (generations), limit of objective function calls, limit of CPU execution time, etc. Define properties IterationCount, IterationObjectiveAverage, IterationBestObjective for progress display.

In particular, we will implement a derived class `BinaryGA`. Star implementing a binary encoded GA solver that inherits `GenericGASolver<byte>`; e.g., `BinaryGA: GenericGASolver< byte >` Take the advantages of the polymorphism mechanism of O-O techniques to implement virtual and override functions to complete the functionalities of the derived class. In this assignment, create a .NET Framework library project to host the related classes of GA. Then add a WinForm application project for solving the Job Assignment Problem to test the implemented functions and classes. The job assignment problem is modeled and solved as a 0-1 variable encoded mathematical programming model.

Job Assignment Problems Solved by the Binary-encoded GA:

Company A has n machines and n jobs to be processed. Each machine must be assigned with exactly one job. The time required to set up each machine for processing each job is different and given in a time matrix C. Company A wants to minimize the total setup time needed to complete the jobs. Find the solution using a binary-encoded GA model. The benchmarks of job assignment problems are defined files with file extension “aop”. File format of a sample is listed below.

$$C = \left[c_{ij}\right]_{nxn} = \begin{array}{cc} & \textit{Machines} \\ \textit{Jobs} & \left[c_{ij}\right] \end{array}$$

$$\min \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} c_{ij} x_{ij}$$

$s.t.$

$$\sum_{i=0}^{n-1} x_{ij} = 1;\ j = 0,1,\ldots,n-1$$

$$\sum_{j=0}^{n-1} x_{ij} = 1; i = 0,1,\ldots,n-1$$

$$x_{ij} = \begin{cases} 0 \\ 1 \end{cases}$$

Text File Format:

7	n
14 5 8 7 6 10 8	c_{00}~$c_{0(n-1)}$
2 12 6 5 9 4 3	c_{10}~$c_{1(n-1)}$
7 8 3 9 8 11 4	c_{20}~$c_{2(n-1)}$
2 4 6 10 7 9 3	c_{30}~$c_{3(n-1)}$
5 8 10 14 3 5 9	c_{40}~$c_{4(n-1)}$
4 8 7 6 10 8 7	c_{50}~$c_{5(n-1)}$
3 5 7 8 6 4 7	c_{60}~$c_{6(n-1)}$

There will be n^2 0-1 genes subject to n machine-wise constraints and n job-wise constraints. Evaluate the violation amount of these hard constraints as penalties on the objective function. The objective function is then

$$\min f(\mathbf{x}) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} c_{ij} x_{ij} + M \cdot \left(\sum_{j=0}^{n-1} \left| \left(\sum_{i=0}^{n-1} x_{ij} \right) - 1 \right| + \sum_{i=0}^{n-1} \left| \left(\sum_{j=0}^{n-1} x_{ij} \right) - 1 \right| \right),$$

where M is a user-specified positive penalty multiplier. In your solving system, provide a user interface control to let user specify the weight M of penalty. Note that the variables defined in the above equations are in matrix format, while our variables should be a list of binary variables stored in a segment of chromosomes. Therefore, you need to transform the two-dimensional variables into a one-dimensional binary variable array. For example,

$$\mathbf{x}' = \left[x'_0 x'_1 \cdots x'_{n^2-1} \right]$$

$$\min f'(\mathbf{x}') = \sum_{k=0}^{n^2-1} c_{\left(\left\lfloor \frac{k}{n} \right\rfloor\right)(k \bmod n)} x'_k + M \cdot \left(\sum_{j=0}^{n-1} \left| \left(\sum_{k=jn}^{jn+n-1} x'_k \right) - 1 \right| + \sum_{j=0}^{n-1} \left| \left(\sum_{k=0}^{n-1} x'_{kn+j} \right) - 1 \right| \right)$$

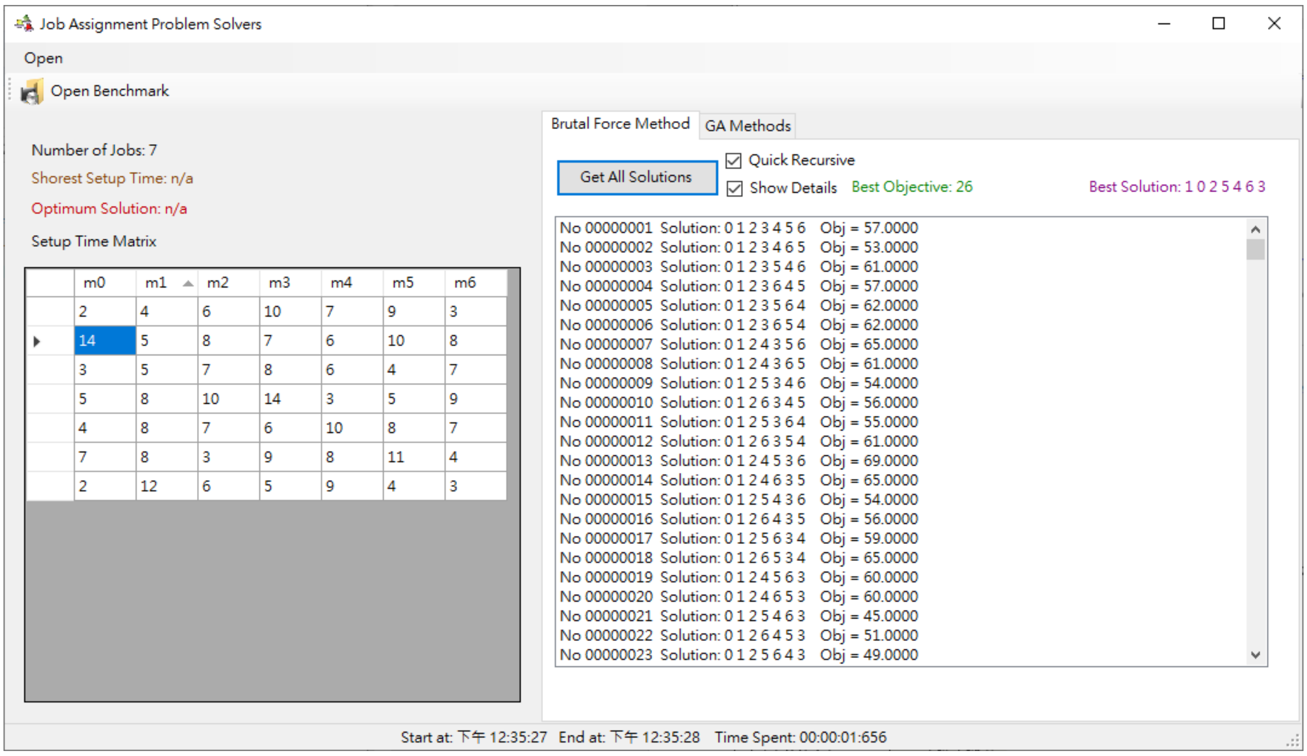
or

$$\min f'(\mathbf{x}') = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} c_{i,j} x'_{(i \cdot n+j)} + M \cdot \left(\sum_{j=0}^{n-1} \left| \left(\sum_{i=0}^{n-1} x'_{(i \cdot n+j)} \right) - 1 \right| + \sum_{i=0}^{n-1} \left| \left(\sum_{j=0}^{n-1} x'_{(i \cdot n+j)} \right) - 1 \right| \right)$$

You need to define the objective evaluation function correctly to return the objective value that has aggregated penalties of the hard constraint violations.

Bonus:

In addition to solving the problem via BinaryGA, develop a brutal force method to enumerate all possible solutions and compute their objective values. Report the best solution and objective value for comparison. Usually, a recursive function is required in your code to systematically constitute all compositions of the jobs in difference sequences.



Number of Jobs: 7
Shorest Setup Time: n/a
Optimum Solution: n/a
Setup Time Matrix

m0	m1	m2	m3	m4	m5	m6
2	4	6	10	7	9	3
14	5	8	7	6	10	8
3	5	7	8	6	4	7
5	8	10	14	3	5	9
4	8	7	6	10	8	7
7	8	3	9	8	11	4
2	12	6	5	9	4	3

[✓] Quick Recursive
[✓] Show Details
Best Objective: 26
Best Solution: 1 0 2 5 4 6 3

```
No 00000001  Solution: 0 1 2 3 4 5 6   Obj = 57.0000
No 00000002  Solution: 0 1 2 3 4 6 5   Obj = 53.0000
No 00000003  Solution: 0 1 2 3 5 4 6   Obj = 61.0000
No 00000004  Solution: 0 1 2 3 6 4 5   Obj = 57.0000
No 00000005  Solution: 0 1 2 3 5 6 4   Obj = 62.0000
No 00000006  Solution: 0 1 2 3 6 5 4   Obj = 62.0000
No 00000007  Solution: 0 1 2 4 3 5 6   Obj = 65.0000
No 00000008  Solution: 0 1 2 4 3 6 5   Obj = 61.0000
No 00000009  Solution: 0 1 2 5 3 4 6   Obj = 54.0000
No 00000010  Solution: 0 1 2 6 3 4 5   Obj = 56.0000
No 00000011  Solution: 0 1 2 5 3 6 4   Obj = 55.0000
No 00000012  Solution: 0 1 2 6 3 5 4   Obj = 61.0000
No 00000013  Solution: 0 1 2 4 5 3 6   Obj = 69.0000
No 00000014  Solution: 0 1 2 4 6 3 5   Obj = 65.0000
No 00000015  Solution: 0 1 2 5 4 3 6   Obj = 54.0000
No 00000016  Solution: 0 1 2 6 4 3 5   Obj = 56.0000
No 00000017  Solution: 0 1 2 5 6 3 4   Obj = 59.0000
No 00000018  Solution: 0 1 2 6 5 3 4   Obj = 65.0000
No 00000019  Solution: 0 1 2 4 5 6 3   Obj = 60.0000
No 00000020  Solution: 0 1 2 4 6 5 3   Obj = 60.0000
No 00000021  Solution: 0 1 2 5 4 6 3   Obj = 45.0000
No 00000022  Solution: 0 1 2 6 4 5 3   Obj = 51.0000
No 00000023  Solution: 0 1 2 5 6 4 3   Obj = 49.0000
```

Start at: 下午 12:35:27 End at: 下午 12:35:28 Time Spent: 00:00:01:656