ECGAAlgorithm

Algorithm

eda, binary

Gevol.evolution.algorithm.eda.binary

Description

Extended Compact Genetic Algorithm (ECGA) is an extended version of Compact Genetic Algorithm (CGA). It uses linkage learning. It means that it looks for correlated genes and put them into groups called blocks. Every gene belongs to only one group, so the groups are independent.

The algorithm works as long as it can improve the marginal probability model (MPM). The model is being searched by greedy algorithm. Simply, it tries to add every gene to every group. If he can't add anything to any group (it can't improve the model) the algorithm stops. Evaluation of the model is based on the formula.

$$CC(MPM) = Model\ Complexity + Compressed\ Population\ Complexity$$

Where:

- Model Complexity – it favorites models with the smallest complexity
- Compressed Population Complexity – the population shouldn't be too complex, it shows the cost of representing the model itself

$$Model\ Complexity = \log_2(N) \sum_{i=1}^{M} 2^{k_i}$$

Where:

- N – population size
- M – number of blocks
- k_i – number of genes in i-th block

Georges Harik in his work "*Linkage Learning via Probabilistic Modeling in the ECGA*" publicized the formula for the Compressed Population Complexity:

$$Compressed\ Population\ Complexity = N \sum E(M_I)$$

Where:

- N – population size
- E – entropy
- M_i – marginal distribution for i-th block

We can understand this formula as: population size x sum of entropies of marginal distributions in every block.

Entropy – how strong the model is unpredictable, for example the biggest entropy is when we have probability 50%, entropy doesn't exist if the probability is 100% or 0%.

$$Binary\ entropy = -\sum_i p(x_i)\log_2(p(x_i))$$

Marginal distribution is probability to get every value of X in the set. Dependency of other variables are omitted.

In the result we can get the following formula to be implemented:

$$Compressed\ Population\ Complexity = N \sum_{i}^{M} \sum_{j}^{2^{k_i}} -p_j \log_2(p_j)$$

Where:

- N - population size
- M - number of blocks
- 2^{k_i} - all possible variation of values for all genes in the block
- p_j - probability to get specific value for every gene in the block

Example.

From the whole population was selected 5 individuals:

1. 0000
2. 0010
3. 0011
4. 1010
5. 1101

The MPM model looks like: $[x_1,x_2][x_3][x_4]$. It means that genes x1 and x2 are correlated, x_3 and x_4 are independent. The marginal distribution would look like:

1. For $[x_1,x_2]$

	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	3/5	1/5
$x_1 = 1$	0	1/5

2. For $[x_3]$

$x_3 = 0$	2/5
$x_3 = 1$	3/5

3. For $[x_4]$

$x_4 = 0$	3/5
$x_4 = 1$	2/5

$$Model\ Complexity = log_2(5)\left((2^2) + (2^1) + (2^1)\right) = 18{,}58$$

$$Compressed\ Population\ Complexity$$
$$= 5\left(abs\left(\frac{3}{5}log_2\left(\frac{3}{5}\right)\right) + abs\left(\frac{0}{5}log_2\left(\frac{0}{5}\right)\right) + abs\left(\frac{1}{5}log_2\left(\frac{1}{5}\right)\right)\right.$$
$$+ abs\left(\frac{1}{5}log_2\left(\frac{1}{5}\right)\right) + abs\left(\frac{2}{5}log_2\left(\frac{2}{5}\right)\right) + abs\left(\frac{3}{5}log_2\left(\frac{3}{5}\right)\right) + abs\left(\frac{3}{5}log_2\left(\frac{3}{5}\right)\right)$$
$$\left. + abs\left(\frac{2}{5}log_2\left(\frac{2}{5}\right)\right)\right) = 16{,}56$$

$$CC(MPM) = 18{,}58 + 16{,}56$$

The most important part – how to evaluate the model is known. The ECGA algorithm looks like:

1. Generate random population
2. Tournament selection (in this implementation it is changed to block selection as it gives much better results)
3. Create the MPM model based on greedy search
4. If it's not possible to find better model then stop otherwise continue
5. Generate new population based on the model
6. Return to step 2.

The greedy search for new model is very simple:

1. Every gene is in independent group
2. Concatenate each group with every other group
3. If concatenated group has better CC, save it
4. If any of the create group have better CC, update the best group and return to step 2, otherwise stop
5. If new the best group is better than current model, update current model with the new group, otherwise stop algorithm.

Observations:

1. At the beginning it builds blocks. Later as the probabilities convergence to one value, the blocks are dismantled, because it will better score and the result will remain.
2. Usually it finds very good solution at the very beginning. Later these individuals often are forgotten. The model also doesn't follow the best individuals.
3. It's important to set proper selection size. If the population is too less, it will not create more complicated model, because it will get too bad score. The smaller population the more important model is.
4. Tested on composed trap function. It may find few traps, but not all. Usually at the end it loses most of the found traps.
5. It doesn't want to create blocks built from more than 2 genes. Tested on composed trap function with 3, 4 and 5 block size.
6. Performance depends on the number of genes. One more gene in chromosome have big impact in performance. The root cause is the greedy MPM model search.

The tournament selection has been changed to block selection. Observations in reference to the previous ones:

1. At the beginning it builds blocks. Later as the probabilities convergence to one value, the blocks are dismantled, because it will better score and the result will remain.
2. It's not true anymore.
3. It's still valid.
4. It finds all of the traps in few iterations.
5. It creates more complicated model where the blocks are built from more than 2 genes.
6. Performance is much better, as only few iterations is required to find the global optimum for smaller traps. For bigger traps it finds some traps.

Parameters

1. Population size – size of the population used in the algorithm
2. Selection size – size of subpopulation used to define MPM model
3. Chromosome length – number of genes in chromosome
4. Blocks iteration limit – it indicates that model is convergence if the model has not changed for given number of iterations

5. Accepted probability difference - it indicates that model is convergence if probabilities for each block in the model has not changed more than given number of iterations. The model is convergence if both convergence parameters are satisfied.

Pseudocode

```
ECGA algorithm:
P = generate random population
While true
      P' = block selection (P)
      M' = greedy search for MPM model (P')
      If( CC(M') < CC(M) )
            M = M'
      Else
            Stop

Greedy search for MPM model (P)
Every gene is in independent group
While true
      For each group g in all groups G
            For each group g' in all groups G
                  If g != g'
                        g = g + g'
                        if CC(g) < CC(the best g')
                              the best g' = g
      If CC(the best g') < CC(the best g)
            The best g = the best g'
      Otherwise
            Stop
```

Implementation details

Roulette selection has been changed to block selection in the `ECGAAlgorithm` class as it gives better results.

```
//RouletteSelection rs = new RouletteSelection(selectionSize);  //block selection
gives better results
BlockSelection rs = new BlockSelection(selectionSize);
```

In the `Apply` method, the greedy MPM search is implemented. At the very beginning it calculates the model where all genes are independent.

```
CalculateModel(population, ref newModel);
```

For now it is the best model for the loop.

```
ECGAModel concatenatedModel = newModel;
```

Try to concatenate every block with every block.

```
for(int i = 0; i < newModel.Blocks.Count - 1; i++)
{
      for (int j = i + 1; j < newModel.Blocks.Count; j++)
```

Check if concatenated model is better:

```
if (newConcatenatedModel.CC < concatenatedModel.CC)
```

If yes, current the best model for the loop `concatenatedModel` is `newConcatenatedModel`.

Continue the search as long as better model is found. So, the best ever model is remembered and the loop starts over again. Try to concatenate every block together starting from the current the best model for the loop.

```
if (concatenatedModel.CC < newModel.CC)
{
	newModel = concatenatedModel;
	continue;
}
```

Method `ConcatenateGroups` does not change existing model, it creates new one with new blocks. It adds genes ids to the first group and it sort it. It is important to have the genes numbers in proper order. Otherwise the probabilities would be understood wrongly. Second group is removed, so there are no duplicates.

```
((List<int>)newModel.Blocks[groupA].Genes).AddRange(newModel.Blocks[groupB].Genes);
((List<int>)newModel.Blocks[groupA].Genes).Sort();
newModel.Blocks.RemoveAt(groupB);
```

Probabilities are calculated in `CalculateModel(Population population, ref ECGAModel model)` method. It changes the input model (`ref ECGAModel model`).

It creates additional temporary list to store the genes' values for each block. For example:

Individual = 1010 1011 1101, the block is built from genes 2,6,7,9 (first gene is 0). So the `genesValues` = 1111. Then it saves how many times that values appeared in the population for that genes.

```
block.Probabilities[model.GenesSetToIndex(genesValues)]++;
```

Method `GenesSetToIndex` converts binary code to integer numbers. In this example 1111 = 15, so `block.Probabilities[15]` will be increased. Let's check how the probabilities is stored for the whole block. In this example, the block is built from four genes 2,6,7,9. So it can have 2^4 possible values like 0011, 1010, 1100, etc. Every value may be converted into the integer number: 0011 = 3, 1010 = 10, 1100 = 12. Every integer number corresponds to the index in probability list of the model. In the end every index (possible set of genes) have the probability calculated, for example `Probabilities[15] = 0.21`.

Generate new individual. Function `Randomizer.NextIndex` returns index from the list. The input list has probabilities saying how probable is to select its index. The index is transformed into binary code using the method `IndexToGenesSet` from `ECGAModel` class. We know the genes number for that block and the rand values. Now it is saved in the individual chromosome on the right positions.

```
chromosome[block.Genes[g]] = genesValues[g];
```

Special termination condition `ECGAModelConvergence` has been introduced to calculate if the model is convergence. It can't be general as model for different algorithms may differ. Additional termination condition `ECGAModelImprovement` has been also implemented. It checks if new ECGA model is better (has lower CC value) than previous one. Usually it gives worse results than checking the convergence.

During the first execution there is no reference model. So it takes the one from the input. In the every next iteration, two checks are performed:

```
if (!checkBlocksStructure((ECGAModel)model) || !checkProbabilities((ECGAModel)model))
```

If both of them fails, input model becomes the referential one. If both succeeded, it means the model is convergence and it stops the algorithm. In the method `checkBlocksStructure` it compares every genes list in every block. In `checkProbabilities` it compares the probabilities.

References

1. Linkage Learning via Probabilistic Modeling in the ECGA, Georges Harik, IlliGAL Technical Report 99010, 1999
2. Exploitation of Linkage Learning in Evolutionary Algorithms, Ying-ping Chen, Natural Computing Laboratory Department of Computer Science National Chiao Tung University, Taiwan, Springer-Verlag Berlin Heidelberg, 2010, ISBN: 978-3-642-12833-2
3. Evolutionary Optimization Algorithms, Dan Simon, Cleveland State University, USA, John Wiley & Sons. Inc. 2013, ISBN: 978-0-470-93741-9
4. Genetic and Evolutionary Computation – GECCO 2004, Kalyanmoy Deb, Riccardo Poli, Woflgang Banzhaf, Hans-Georg Beyer, Edmund Burke, Paul Darwen, Dipankar Dasgupta, Dario Floreano, James Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, Andy Tyrrels (Eds.), Genetic and Evolutionary Computation Conference Seattle, WA, USA, June 26-30, 2004, Proceedings, Part II, Springer-Verlag Berlin Heidelberg New York, ISBN: 3-540-22343-6