MATEDA: A modular approach to EDAs in Matlab

Roberto Santana

Computational Intelligence Group
www.ai-research.eu

Technical University of Madrid

Outline

Characteristics of the program

MATEDA features

- Optimization of single and multi-objective problems
- Highly modular implementation where each EDA component is implemented as an independent program.
- User can add new components to the program
- Includes learning and sampling methods of undirected and directed probabilistic graphical models
- It can solve problems with discrete and continuous variables

Characteristics of the program

MATEDA features

- The knowledge extraction and visualization module can extract, process, and display information from the data generated by the EDA
- It has a variety of seeding, local optimization, repairing, selection and replacement methods
- It includes statistical analysis of different measures of the EDA evolution
- It has an extended library of functions and testbed problems

Characteristics of the program

MATEDA modules

- Optimization module: Implements different EDAs for single and multi-objective problems
- Data analysis and visualization module: Extracts and visualizes the structures of the models learned during the evolution
- Function approximation module: Creates and validates models of the functions based on the probabilistic models learned by EDAs

A modular approach to EDAs

1	Set $t \Leftarrow 0$
2	Do
3	If $t = 0$
4	Generate an initial population D_0 using a *seeding method*
5	If required, apply a *repairing method* to D_0
6	Evaluate (all the objectives of) population D_0 using an *evaluation method*
7	If required, apply a *local optimization method* to D_0
8	Else
9	Sample a $D_{Sampled}$ population using a *sampling method*
10	Evaluate (all the objectives of) population $D_{Sampled}$ using an *evaluation method*
11	If required, apply a *repairing method* to $D_{Sampled}$
12	If required, apply a *local optimization method* to $D_{Sampled}$
13	Create D_t from populations D_{t-1} and $D_{Sampled}$ using a *replacement method*
14	Select a set D_t^S of points from D_t according to a *selection method*
15	Compute a probabilistic model of D_t^S using a *learning method*
16	$t \Leftarrow t + 1$
17	Until the evaluation of the *termination criteria method* is true

Table: MATEDA approach to EDAs

Characteristics of the program

MATEDA components

- *Repairing methods* are applied to constrained problems for which sampled solutions are unfeasible
- The *evaluation method* comprises the evaluation of the solutions
- *Local optimization methods* do some local search around the current solution
- *Sampling methods* are used to generate new solutions from the learned probabilistic models

Characteristics of the program

MATEDA components

- *Replacement methods* combine the solutions visited in previous generations with the current set of sampled solutions
- *Selection methods* identify the set of solutions that will be modeled, which are usually the solutions with the best fitness evaluation.
- The *learning method* involves parametric or structural learning, also known as model fitting and model selection, respectively.
- The *termination criteria method* determines the stopping conditions for EDA evolution

Characteristics of the program

Probabilistic models that can be implemented

- General factorizations
- Bayesian networks
- Gaussian networks
- Markov networks
- Mixtures of distributions

Characteristics of the program

EDAs that can be implemented

- Univariate marginal distribution algorithm
- Factorized distribution algorithm
- EDAs based on trees and forests
- EDAs based on Bayesian and Gaussian networks
- Markov chain estimation of distribution algorithm

Characteristics of the program

EDAs that can be implemented

- EDAs based on univariate and multivariate Gaussian distributions
- EDAs based on mixtures of continuous distributions
- Markov optimization algorithm
- Affinity propagation EDA

Program installation

Steps to install Mateda-2.0

1. Create a working directory TestMateda
2. Download Mateda program from http://www.sc.ehu.es/ccwbayes/members/rsantana/software/matlab/MATEDA.html
3. Unzip and untar file IntEDA.tar.gz and put the Mateda2.0 directory within the directory TestMateda
4. Download Bayes net toolbox for matlab http://code.google.com/p/bnt/
5. Unzip FullBNT-1.0.4.zip and put the directory FullBNT-1.0.4 within TestMateda
6. Download Structure Learning Package for Bayes Net Toolbox at http://www.mathworks.com/matlabcentral/fileexchange/13562-structure-learning-package-for-bayes-net-toolbox
7. Unzip and untar the BNT-SL.tar.zip file, extract the directory BNT-SLP and put it within the directory TestMateda
8. Set the path to the current BNT structure learning tool directory. This is done by modifying file $add\_SLP.m$, $SLP\_HOME = D:\TestMateda\BNT\_SLP$;
9. Open file InitEnvironment.m which is within Mateda2.0 and set $path\_mateda = D:\TestMateda\Mateda2.0$, $path\_FullBNT = D:\TestMateda\FullBNT1.0.4$ and $path\_BNT\_SLP = D:\TestMateda\BNT\_SLP$
10. *cd TestMateda\Mateda* and run *InitEnvironment*

Characteristics of the program

Steps to run an EDA in Mateda-2.0

1. Create or define the file of the function to be optimized
2. Define the type of representation to be used
3. Create the vector with the range of values (for the continuous case) or the cardinality of the variables (for the discrete case)
4. Choose each EDA component, identify its corresponding MATEDA implementation and determine the parameters to be passed to each method
5. Execute RunEDA.m

Characteristics of the program

How to define the EDA components?

- Type of variable representation (discrete or continuous)
- Domain of definition for each variable (discrete problems with high cardinality may not be treated using complex models)
- Existence of prior information about the problem (e.g., when a feasible factorization is known it can be employed)
- Computational cost of the fitness function (should be taken into account when setting the population size)

Input parameters

Main parameters

- *PopSize*: Population size
- *n*: Number of variables
- *F*: Name of the Matlab file that implements the (possibly multiobjective) function
- *Card*: Cardinalities of the variables for the discrete problems or range of each variable for the continuous problem
- *cache*: A vector specifying which components of the algorithm will be stored

Cache parameters

1. Entire population
2. Selected population
3. Probabilistic model
4. Fitness values of the entire population
5. Fitness values of the selected population

Input parameters

How to define the EDA components?

- *edaparams*: An array of cells specifying all the components and parameters used by the EDA
- Each row has the form {*type_of_method*, *name_of_implementation*, *implementation_parameters*}
- *type_of_method* defines an EDA component:
  - 'seeding_pop_method'
  - 'sampling_method',
  - 'repairing_method'
  - 'local_opt_method'
  - 'replacement_method'
  - 'selection_method'
  - 'learning_method'
  - 'statistics_method'
  - 'verbose_method'
  - 'stop_cond_method'

EDA implementation

EDA with parameters by default

- Model: Tree
- Learning method: LearnBN
- Sampling method: SampleBN
- Selection method: Truncation selection ($T = 0.5$)
- Replacement method: Best elitism
- Stopping criterion: Max. number of generations (50)

EDA implementation

EDA with parameters by default: Onemax function

- Problem: Define an EDA with default parameters for solving the discrete OneMax function $f(\mathbf{x}) = \sum_{i=1}^{n} x_i$ where $n = 30$ and $x_i \in \{0, 1\}$
- Open file DefaultEDA_OneMax.m in OptimizationScripts directory

```
  PopSize = 300;
1 n = 30;
2 F = 'sum';
3 Card(1,:) = zeros(1,n);
4 Card(2,:) = 2*ones(1,n);
5 cache = [1,1,1,1,1];
6 edaparams = {};
7 [AllStat,Cache] = RunEDA(PopSize,n,F,Card,cache,edaparams);
```

EDA implementation output

EDA with parameters by default:Onemax function

********** Generation 19 **********
Max objective values: 10
Sum of max objective values: 10
Mean objective values: 10
Sum of mean objective values: 10
Median objective values: 10
Sum of median objective values: 10
Min objective values: 10
Sum of min objective values: 10
Variance of the objective values: 0
Best individual: 1 1 1 1 1 1 1 1 1 1
Number of different individuals: 1
Max values of the variables: 1 1 1 1 1 1 1 1 1 1
Mean values of the variables: 1 1 1 1 1 1 1 1 1 1
Median values of the variables: 1 1 1 1 1 1 1 1 1 1
Min values of the variables: 1 1 1 1 1 1 1 1 1 1
Variance of the variables: 0 0 0 0 0 0 0 0 0 0
number of evaluations: 15000
Time: sampling 3.000000e-01, repairing 0, evaluation 0, local-opt 0, replacement 0, selection 0, learning 4.000000e-02, and total 3.400000e-01

EDA implementation output

EDA with parameters by default:Onemax function

- Populations of the first and tenth generations $image(Cache\{1,1\} * 50)$, $image(Cache\{1,10\} * 50)$,
- Probabilistic model of the first and tenth generations: $Cache\{3,1\}$, $Cache\{3,10\}$

Mean fitness value at each generation

```
for i=1:50,
meanval(i) = AllStat{i}(2)
end
plot(meanval)
```

EDA implementation

EDA with parameters by default:Trapn function

- Problem: Define an EDA with default parameters for solving the discrete deceptive *Trapn* function $f(\mathbf{x}) = \sum_{i=1}^{n} x_i$ where $n = 45$ and $x_i \in \{0, 1\}$

```
  PopSize = 300;
1 n = 45;
2 F = 'evalfunctrapn';
3 global ntrapparam; ntrapparam = 5; Card(1,:) = zeros(1,n);
4 Card(2,:) = 2*ones(1,n);
5 cache = [1,1,1,1,1];
6 edaparams = {}; MaxGen = 10;
7 stop_cond_params = {MaxGen};
8 edaparams{1} = {'stop_cond_method','maxgen',stop_cond_params};
9 [AllStat,Cache] = RunEDA(PopSize,n,F,Card,cache,edaparams);
```

EDA implementation

EDA with parameters by default:Trapn function

```
function[val]=evalfunctrapn(vector)
global ntrapparam
NumbVar = size(vector,2);
val = 0;
for i=1:ntrapparam:NumbVar
val = val+Trapn(vector(i:i+ntrapparam-1),ntrapparam);
end
```

EDA implementation

Implementation of a continuous Gaussian UMDA

- Open file GaussianUMDA_ContSumFunction.m in OptimizationScripts directory

```
  PopSize = 500;
1 n = 30;
2 F = 'sum';
3 Card(1,:) = zeros(1,n);
4 Card(2,:) = 5*ones(1,n);
5 cache = [0,0,0,0,0];
6 edaparams{1} = {'learning_method','LearnGaussianUnivModel',{}};
7 edaparams{2} = {'sampling_method','SampleGaussianUnivModel',{PopSize,1
8 edaparams{3} = {'replacement_method','elitism',{1,'fitness_ordering'}}
9 edaparams{4} = {'selection_method','prop_selection',{}};
10 [AllStat,Cache] = RunEDA(PopSize,n,F,Card,cache,edaparams);
```

EDA implementation

Tree-EDA for the optimization of the Ising

- Open file BayesianTree_IsingModel.m in OptimizationScripts directory

```
  global lattice; % The lattice and the interactions
1 global inter; % are defined as global variables
2 Pop-Size = 500;
3 n = 64;
4 cache = [0,0,0,0,0];
5 Card = 2*ones(1,n);
6 F = 'EvalIsing'; % Ising function
7 MaxGen = 150;
8 MaxVal = 86;
9 lattice, inter= LoadIsing(n, 1); % The Ising instances is read
10 stop_cond_params = {MaxGen,MaxVal}; % The algorithm stops when
  % optimum is found or maxgen is reached
11 edaparams{1} = {'stop_cond_method','maxgen_maxval',stop_cond_params};
12 edaparams{2} = {'sampling_method',' SampleMPE_BN',{PopSize}};
13 [AllStat,Cache]=RunEDA(PopSize,n,F,Card,cache,edaparams);
```

EDA implementation

Implementation of a discrete HP protein model

```
  global InitConf; % This is the HP protein instance
1 % defined as a sequence of zeros and ones
2 InitConf = [zeros(1,12),1,0,1,0,1,1,0,0,1,1,0,0,1,1,0,1,1,0,0,1,1
3  ,0,0,1,1,0,1,1,0,0,1,1,0,0,1,1,0,1,0,1,zeros(1,12)];
  PopSize = 500;
4 NumbVar = size(InitConf,2);
5 cache = [0,0,0,0,0];
6 Card = 3*ones(1,NumbVar);
7 maxgen = 100;
8 % The Markov chain model(Cliques) is constructed specifying the number of
9 % conditioned (previous) variables. In the example below this number is
10 % 1., i.e. p(x) = p(x0)p(x1|x0)...p(xn|xn − 1)
11 Cliques = CreateMarkovModel(NumbVar,1);
12 F = 'EvaluateEnergy'; % HP protein evaluation function
13 edaparams{1} = {'learning_method','LearnFDA',Cliques};
14 edaparams{2} = {'sampling_method','SampleFDA',PopSize};
15 % Repairing method used to guarantee that solutions do not self-intersect
16 edaparams{3} = {'repairing_method','HP_repairing',{}};
17 edaparams{4} = {'stop_cond_method','max_gen',{maxgen}};
18 [AllStat,Cache]=RunEDA(PopSize,NumbVar,F,Card,cache,edaparams)
```

EDA implementation

Bayesian network based EDA for multi-objective 3-SAT

```
  global Formulas; % Global variable for SAT function
1 load TypeForm_1_Form_1.mat; % Multimodal SAT instance
2 n = 20; % Number of variables
3 m = 10; % Number of objectives
4 c = 20; % Number of clauses
5 PopSize = 500;
6 NumbVar = n;
7 cache = [1,1,1,1,1];
8 Card = 2*ones(1,NumbVar);
9 maxgen = 50;
10 F = 'EvaluateSAT'; % 3-SAT function
11 selparams(1:2) = {0.5,'ParetoRank_ordering'};
12 sampling_params(1:3) = {PopSize,m,Obj_Card};
13 BN_params(1:7) = {'k2',5,0.05,'pearson','bayesian','no'};
14 edaparams{1} = {'learning_method','LearnBN',BN_params};
15 edaparams{2} = {'sampling_method','SampleBN',sampling_params};
16 edaparams{3} = {'selection_method','truncation_selection',selparams};
17 edaparams{4} = {'replacement_method','best_elitism',{'ParetoRank_ordering'}};
18 edaparams{5} = {'stop_cond_method','max_gen',{maxgen}};
19 [AllStat,Cache]=RunEDA(PopSize,NumbVar,F,Card,cache,edaparams);
```

EDA implementation

Mixture of Gaussian distributions for a trajectory problem

```
  global MGADSMproblem % Global variable for the space trajectory problem
1 load EdEdJ; % The instance is read
2 NumbVar = 12;
3 PopSize = 5000;
4 F = 'EvalSaga';
5 Card(1,:) = [7000,0,0,0,50,300,0.01,0.01,1.05,8,-1*pi,-1*pi];
6 Card(2,:) = [9100,7,1,1,2000,2000,0.90,0.90,7.00,500,pi,pi];
7 cache = [0,0,1,0,1];
8 learning_params(1:5) = {'vars','ClusterPointsKmeans',10,'sqEuclidean',1};
9 edaparams{1} = {'learning_method','LearnMixtureofFullGaussianModels',learning_params};
10 edaparams{2} = {'sampling_method','SampleMixtureofFullGaussianModels',{PopSize,3}};
11 edaparams{3} = {'replacement_method','best_elitism',{'fitness_ordering'}};
12 selparams(1:2) = {0.1,'fitness_ordering'};
13 edaparams{4} = {'selection_method','truncation_selection',selparams};
14 edaparams{5} = {'repairing_method','SetWithinBounds_repairing',{}};
15 edaparams{6} = {'stop_cond_method','max_gen',{5000}};
16 [AllStat,Cache]=RunEDA(PopSize,NumbVar,F,Card,cache,edaparams)
```

EDA implementation

Optimization of multi-objective decomposable functions

```
  PopSize = 1000;
1 n = 50;
2 cache = [1,1,1,1,1];
3 Card = 2*ones(1,n);
4 MaxGen = 30;
5 global FunctionTables;
6 global FunctionStructure;
7 global FunctionAccCard;
8 global SelectedObjectives;
9 % The circular structure is created
10 FunctionStructure = CreateListFactorsCircularNK(n,4);
11 % Values are read from a file
12 FunctionTables = ReadFunctionsFromData('testNK_fnt_N50_k4Inst_1.txt',
13 FunctionStructure,Card);
14 FunctionAccCard = FindListCard(FunctionStructure,Card);
15 SelectedObjectives = [1:4:48]; % Only some of the objectives are evaluated
16 F = 'PartialEvaluateGeneralFunction';
17 selparams(1:2) ={0.5,'ParetoRank_ordering'};
18 edaparams{1} = {'selection_method','truncation_selection',selparams};
19 edaparams{2} = {'replacement_method','best_elitism',{'ParetoRank_ordering'}};
20 edaparams{3} = {'stop_cond_method','max_gen',{MaxGen}};
21 [AllStat,Cache]=RunEDA(PopSize,n,F,Card,cache,edaparams);
```