Evolving Mean-Update Selection Methods for CMA-ES

Samuel Richter (snr359@mst.edu)
Daniel Tauritz (dtauritz@acm.org)
Michael Schoen (ms778@mst.edu)

Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri 65409

ECADA @ GECCO, July 2019

Introduction

- Covariance-Matrix Adaptation Evolution Strategies (CMA-ES) use a sample-update cycle to optimize functions

Introduction

- Covariance-Matrix Adaptation Evolution Strategies (CMA-ES) use a sample-update cycle to optimize functions
- CMA-ES keep several state variables, including search space mean, evolution path, and covariance matrix

Introduction

- Covariance-Matrix Adaptation Evolution Strategies (CMA-ES) use a sample-update cycle to optimize functions
- CMA-ES keep several state variables, including search space mean, evolution path, and covariance matrix
- Search space mean is updated every generation using sampled points

Overview

- We wish to tune CMA-ES to a particular problem class

Overview

- We wish to tune CMA-ES to a particular problem class
- We do this by tuning the mean-update method

Overview

- We wish to tune CMA-ES to a particular problem class
- We do this by tuning the mean-update method
- We evolve a new method of selecting the points used to update the mean

Objective

- Objective: use a Hyper-Heuristic to generate a new mean-update selection function for CMA-ES to tune it to a problem class

Objective

- Objective: use a Hyper-Heuristic to generate a new mean-update selection function for CMA-ES to tune it to a problem class
- Step 1: define a representation for selection functions to form a search space

Objective

- Objective: use a Hyper-Heuristic to generate a new mean-update selection function for CMA-ES to tune it to a problem class
- Step 1: define a representation for selection functions to form a search space
- Step 2: explore this space and determine the quality of the selection functions to find the best one

Selection Function Representation

- We use a two-part structure to represent a selection function

Selection Function Representation

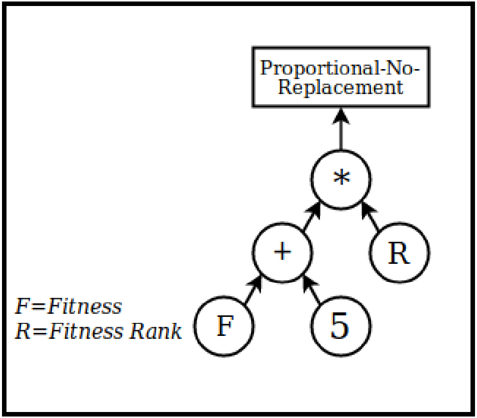
- We use a two-part structure to represent a selection function
- The first part is a GP-Tree encoding a real-valued function of each point's fitness, fitness ranking, and other metrics, returning a "desirability" score for each point

Selection Function Representation

- We use a two-part structure to represent a selection function
- The first part is a GP-Tree encoding a real-valued function of each point's fitness, fitness ranking, and other metrics, returning a "desirability" score for each point
- The second part is a final selection step that selects points based on their desirability scores

Representation

A selection function is represented by a mathematical function (encoded in a parse tree) and a selection method.



GP-Tree Operators

Operator	Operands	Description
$+$	2	Adds the left and right operands.
$-$	2	Subtracts the right operand from the left operand.
$\times$	2	Multiplies the left and right operands.
$/$	2	Divides the left operand by the right operand. If the right operand is 0, the left operand is instead divided by a very small number, returning a large number while preserving the sign of the left operand.
Min	2	Returns the minimum of the left and right operands.
Max	2	Returns the maximum of the left and right operands.

GP-Tree Operators

Operator	Operands	Description
Step	2	Returns 1 if the left operand is greater than or equal to the right operand, and 0 otherwise.
Absolute Value	1	Returns the absolute value of the operand.

GP-Tree Terminals

Terminal	Description
Fitness	The individual's fitness value.
Fitness Rank	The individual's index in a list of the population members sorted by fitness, increasing.
Relative Fitness	The individual's fitness value divided by the sum of all fitness values in the population.
Birth Generation	The generation number that the individual first appeared in the population.
Relative Uniqueness	The Cartesian distance between the individual's genome and the centroid of all genomes in the population.

GP-Tree Terminals

Terminal	Description
Population Size	The number of individuals in the population.
Min Fitness	The smallest fitness value in the population.
Max Fitness	The largest fitness value in the population.
Sum Fitness	The sum of all fitness values in the population.
Generation Number	The number of generations of individuals that have been evaluated since the beginning of evolution.

GP-Tree Terminals

Terminal	Description
Constant	A constant number, which is generated from a uniform selection within a configured range when the selection function is generated and held constant for the entire lifetime of the selection function.
Random	A random number, which is generated from a uniform selection within a configured range every time selection is performed.

Selection Methods

Method	Description
Proportional-Replacement	A weighted random selection, with each individual's weight equal to its desirability score.
Proportional-No-Replacement	As with Proportional-Replacement, but an individual is removed from the selection pool after being selected.
k-Tournament-Replacement	A random subset of k individuals is considered, and the individual with the highest desirability score in the subset is selected.

Selection Methods

Method	Description
k-Tournament-No-Replacement	As with k-Tournament-Replacement, but an individual is removed from the selection pool after being selected.
Truncation	Individuals with the highest desirability score are selected, with no individual being selected more than once.
Stochastic-Universal-Sampling	Individuals are chosen at evenly spaced intervals of their desirability scores.

Meta-EA

- We use a meta-EA to search through the space of possible mean-update selection functions

Meta-EA

- We use a meta-EA to search through the space of possible mean-update selection functions
- Koza-style GP is used to evolve the trees, with an extra gene encoding the final selection step and any parameters to it

Meta-EA

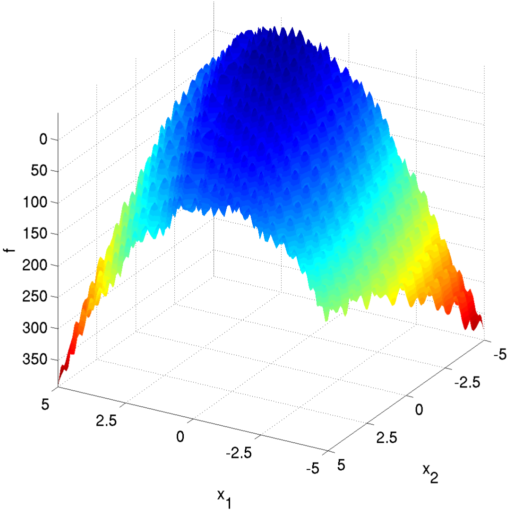
- We use a meta-EA to search through the space of possible mean-update selection functions
- Koza-style GP is used to evolve the trees, with an extra gene encoding the final selection step and any parameters to it
- Each run of the meta-EA targets one of the 24 noiseless test function classes in the Comparing Continuous Optimizers (COCO) function set

Meta-EA

- We use a meta-EA to search through the space of possible mean-update selection functions
- Koza-style GP is used to evolve the trees, with an extra gene encoding the final selection step and any parameters to it
- Each run of the meta-EA targets one of the 24 noiseless test function classes in the Comparing Continuous Optimizers (COCO) function set
- Dimensions of 2, 3, 5, and 10 are used, each with their own meta-EA

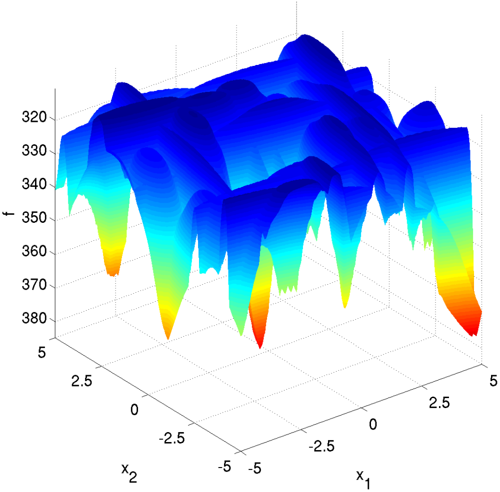
Meta-EA

COCO Function Class #15: Rastrigin Function



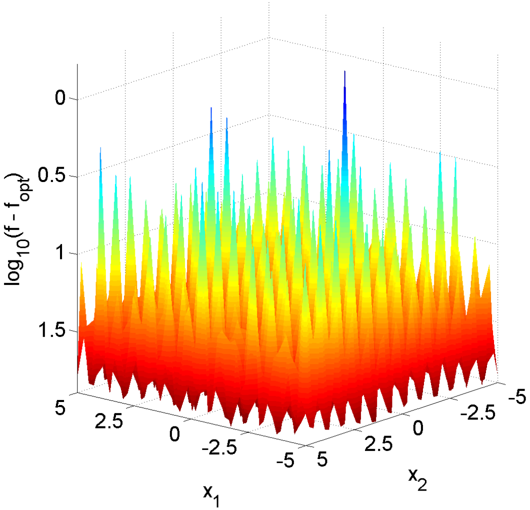
Meta-EA

COCO F. C. #21: Gallagher's Gaussian 101-me Peaks Function



Meta-EA

COCO F. C. #23: Katsuura Function



Meta-EA

- Within the meta-EA, a mean-update selection function is rated by running CMA-ES with it

Meta-EA

- Within the meta-EA, a mean-update selection function is rated by running CMA-ES with it
- CMA-ES is run on a number of instances from the problem class. The solution rate is used as the fitness of the selection function

Meta-EA

- Within the meta-EA, a mean-update selection function is rated by running CMA-ES with it
- CMA-ES is run on a number of instances from the problem class. The solution rate is used as the fitness of the selection function
- After the meta-EA concludes, CMA-ES is run with the best selection function on new instances, to test for generalization

Meta-EA Parameters

Parameter	Value
Population Size	40
Offspring Size	40
Evaluation Count	4000
Max GP-Tree Initialization Depth	4
Parent Selection	k-tournament, k=4
Survival Selection	Truncation
Mutation	Subtree Regeneration
Crossover	Subtree Crossover
Parsimony Pressure Coefficient	0.0005
Mutation Rate	0.25
Range for Constant Terminals	[-100, 100]
Range for Random Terminals	[-100, 100]
Number of Runs (Training)	5
Number of Runs (Testing)	200

Results

- On problem classes 4, 6, 12, 17, 18, 19, 20, and 21, the tuned CMA-ES achieved a 20% greater solution rate than base CMA-ES for at least one dimensionality

Results

- On problem classes 4, 6, 12, 17, 18, 19, 20, and 21, the tuned CMA-ES achieved a 20% greater solution rate than base CMA-ES for at least one dimensionality
- For function class 6 and $D = 10$, success rate increased from 0% to 96%

Results

- On problem classes 4, 6, 12, 17, 18, 19, 20, and 21, the tuned CMA-ES achieved a 20% greater solution rate than base CMA-ES for at least one dimensionality
- For function class 6 and $D = 10$, success rate increased from 0% to 96%
- For function class 12, success rate increased from 44% to 100%

Results

- On problem classes 4, 6, 12, 17, 18, 19, 20, and 21, the tuned CMA-ES achieved a 20% greater solution rate than base CMA-ES for at least one dimensionality
- For function class 6 and $D = 10$, success rate increased from 0% to 96%
- For function class 12, success rate increased from 44% to 100%
- Very few cases where the tuned CMA-ES performs worse, and only 7.4% worse at worst

Results: Selected Examples

Percentage of Runs Solved By Unmodified CMA-ES/Modified CMA-ES, averaged over all instances (selected examples)

F. C.	D=2	D=3
4	3.3% → 32.15%	0.25% → 0.15%
6	100.0% → 100.0%	100.0% → 100.0%
12	100.0% → 99.55%	100.0% → 99.5%
14	100.0% → 100.0%	100.0% → 100.0%
21	27.65% → 56.8%	28.7% → 55.4%
24	1.45% → 1.6%	0.3% → 0.1%

F. C.	D=5	D=10
4	0.0% → 0.0%	0.0% → 0.0%
6	100.0% → 99.1%	0.0% → 96.0%
12	100.0% → 99.85%	44.05% → 44.05%
14	100.0% → 100.0%	100.0% → 100.0%
21	3.3% → 36.05%	36.0% → 28.6%
24	0.0% → 0.0%	0.0% → 0.0%

Future Work

- Tune selection strategies for other problems, including real-world problems

Future Work

- Tune selection strategies for other problems, including real-world problems
- Reduce *a priori* computation required for tuning

Future Work

- Tune selection strategies for other problems, including real-world problems
- Reduce *a priori* computation required for tuning
- Tune more CMA-ES variants

“Take Home Message”

Performance of CMA-ES can be improved on a particular problem class by using a meta-EA to tune the method by which sampled points are selected and used to update the search space mean