COCO for Large-Scale Benchmarking

Duc Manh Nguyen[1] Ouassim Ait ElHara[2]

[1]TAO Team, INRIA Saclay.
[2]Univ. Paris-Sud, LRI, Université Paris-Saclay, TAO Team, INRIA.

8th NumBBO Meeting
May 26, 2016
Paris, France

Outline

Core Change

Problem: Orthogonal (rotation) matrices ($\mathbf{R}$ and $\mathbf{Q}$ in [Hansen et al., 2009]) have quadratic time and space complexities.

Solution: Permuted Orthogonal Block-Diagonal Matrices [Ait Elhara et al., 2016].

Core Change

Problem: Orthogonal (rotation) matrices (**R** and **Q** in [Hansen et al., 2009]) have quadratic time and space complexities.

Solution: Permuted Orthogonal Block-Diagonal Matrices [Ait Elhara et al., 2016].

Core Change

Problem: Orthogonal (rotation) matrices (**R** and **Q** in [Hansen et al., 2009]) have quadratic time and space complexities.

Solution: Permuted Orthogonal Block-Diagonal Matrices [Ait Elhara et al., 2016].

Model

$$\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}} \quad , \tag{1}$$

where:

- $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$: permutation matrices generated using a series of a number, n_s, of truncated uniform swaps:
  - given i, it is swapped with j sampled uniformly from the set $\{\max(0, i - r_s), ..., \min(d, i + r_s)\} \setminus \{i\}$, where r_s is the *swap range*.
- $\mathbf{B}$: a block-diagonal matrix with n_b blocks. Each block $\mathbf{B}_i$ is a *full* orthogonal matrix of size $s_i \times s_i$.

Model

$$\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}} \quad , \tag{1}$$

where:

- $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$: permutation matrices generated using a series of a number, n_s, of truncated uniform swaps:
  - given i, it is swapped with j sampled uniformly from the set $\{\max(0, i - r_s), ..., \min(d, i + r_s)\} \setminus \{i\}$, where r_s is the *swap range*.
- $\mathbf{B}$: a block-diagonal matrix with n_b blocks. Each block $\mathbf{B}_i$ is a *full* orthogonal matrix of size $s_i \times s_i$.

Model

$$\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}} \quad , \tag{1}$$

where:

- $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$: permutation matrices generated using a series of a number, n_s, of truncated uniform swaps:
  - given i, it is swapped with j sampled uniformly from the set $\{\max(0, i - r_s), ..., \min(d, i + r_s)\} \setminus \{i\}$, where r_s is the *swap range*.
- $\mathbf{B}$: a block-diagonal matrix with n_b blocks. Each block $\mathbf{B}_i$ is a *full* orthogonal matrix of size $s_i \times s_i$.

Model

$$\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}} \quad , \tag{1}$$

where:

- $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$: permutation matrices generated using a series of a number, n_s, of truncated uniform swaps:
  - given i, it is swapped with j sampled uniformly from the set $\{\max(0, i - r_s), ..., \min(d, i + r_s)\} \setminus \{i\}$, where r_s is the *swap range*.
- $\mathbf{B}$: a block-diagonal matrix with n_b blocks. Each block $\mathbf{B}_i$ is a *full* orthogonal matrix of size $s_i \times s_i$.

Implementation 1/2

f_ellipsoid_permblockdiag_bbob_problem_allocate(function, dimension, instance, seed,...)

coco_compute_blockrotation($\mathbf{B}$, seed1, d, $\mathbf{s}$, n_b);
coco_compute_truncated_uniform_swap_permutation($\mathbf{P}_{\text{right}}$, seed2, d, n_s, r_s);
coco_compute_truncated_uniform_swap_permutation($\mathbf{P}_{\text{left}}$, seed3, d, n_s, r_s);

d: problem dimension, $\mathbf{s}$: block-sizes and n_b: number of blocks, n_s: number of swaps, r_s: swap range.

Implementation 1/2

f_ellipsoid_permblockdiag_bbob_problem_allocate(function, dimension, instance, seed,...)

coco_compute_blockrotation($\mathbf{B}$, seed1, d, $\mathbf{s}$, n_b);
coco_compute_truncated_uniform_swap_permutation($\mathbf{P}_{\text{right}}$, seed2, d, n_s, r_s);
coco_compute_truncated_uniform_swap_permutation($\mathbf{P}_{\text{left}}$, seed3, d, n_s, r_s);

d: problem dimension, $\mathbf{s}$: block-sizes and n_b: number of blocks, n_s: number of swaps, r_s: swap range.

Implementation 2/2

f_ellipsoid_rotated_bbob_problem_allocate(function, dimension, instance, seed,...)

problem = transform_vars_affine(problem, $\mathbf{M}$, $\mathbf{b}$, d);

f_ellipsoid_permblockdiag_bbob_problem_allocate(function, dimension, instance, seed,...)

problem = transform_vars_permutation(problem, $\mathbf{P}_{\text{left}}$, d);
problem = transform_vars_blockrotation(problem, $\mathbf{B}$, d, $\mathbf{s}$, n_b);
problem = transform_vars_permutation(problem, $\mathbf{P}_{\text{right}}$, d);

Implementation 2/2

f_ellipsoid_rotated_bbob_problem_allocate(function, dimension, instance, seed,...)

problem = transform_vars_affine(problem, $\mathbf{M}$, $\mathbf{b}$, d);

f_ellipsoid_permblockdiag_bbob_problem_allocate(function, dimension, instance, seed,...)

problem = transform_vars_permutation(problem, $\mathbf{P}_{\text{left}}$, d);
problem = transform_vars_blockrotation(problem, $\mathbf{B}$, d, $\mathbf{s}$, n_b);
problem = transform_vars_permutation(problem, $\mathbf{P}_{\text{right}}$, d);

Implementation 2/2

f_ellipsoid_rotated_bbob_problem_allocate(function, dimension, instance, seed,...)

problem = transform_vars_affine(problem, $\mathbf{M}$, $\mathbf{b}$, d);

f_ellipsoid_permblockdiag_bbob_problem_allocate(function, dimension, instance, seed,...)

problem = transform_vars_permutation(problem, $\mathbf{P}_{\text{left}}$, d);
problem = transform_vars_blockrotation(problem, $\mathbf{B}$, d, $\mathbf{s}$, n_b);
problem = transform_vars_permutation(problem, $\mathbf{P}_{\text{right}}$, d);

Implementation 2/2

f_ellipsoid_rotated_bbob_problem_allocate(function, dimension, instance, seed,...)

problem = transform_vars_affine(problem, $\mathbf{M}$, $\mathbf{b}$, d);

$\Downarrow$

f_ellipsoid_permblockdiag_bbob_problem_allocate(function, dimension, instance, seed,...)

problem = transform_vars_permutation(problem, $\mathbf{P}_{\text{left}}$, d);
problem = transform_vars_blockrotation(problem, $\mathbf{B}$, d, $\mathbf{s}$, n_b);
problem = transform_vars_permutation(problem, $\mathbf{P}_{\text{right}}$, d);

Additional Changes

Normalization by Dimensions

problem = transform_obj_norm_by_dim(problem);

$$f^{\text{new}}(\mathbf{x}) = \gamma(d) f(\mathbf{x}), \tag{2}$$

with $\gamma(d) = \min(1, 40/d)$.

Generalized Discuss (f_{11}) and Bent-Cigar (f_{12}) Functions

The number of short (resp. long) axes is set to $\lceil d/40 \rceil$ (it remains 1 for $d \leq 40$).

Additional Changes

Normalization by Dimensions

problem = transform_obj_norm_by_dim(problem);

$$f^{\text{new}}(\mathbf{x}) = \gamma(d) f(\mathbf{x}), \tag{2}$$

with $\gamma(d) = \min(1, 40/d)$.

Generalized Discuss (f_{11}) and Bent-Cigar (f_{12}) Functions

The number of short (resp. long) axes is set to $\lceil d/40 \rceil$ (it remains 1 for $d \leq 40$).

Additional Changes

Normalization by Dimensions

problem = transform_obj_norm_by_dim(problem);

$$f^{\text{new}}(\mathbf{x}) = \gamma(d) f(\mathbf{x}), \tag{2}$$

with $\gamma(d) = \min(1, 40/d)$.

Generalized Discuss (f_{11}) and Bent-Cigar (f_{12}) Functions

The number of short (resp. long) axes is set to $\lceil d/40 \rceil$ (it remains 1 for $d \leq 40$).

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

General Purpose Changes

void *versatile_data {tentative name}

- New member of coco_problem_s (and consequently of coco_problem_t).
- Helps to clearly distinguish the raw problems from the transformed problems.
- Points to data that can be accessible from any *level/layer* of the problem transformation-process:
  - in coco_problem_duplicate(const coco_problem_t *other): problem→versatile_data = other→versatile_data;
  - Use examples:
    - Sets and saves the value of $\hat{z}_1$ (obtained from an intermediate transformation) in the step-ellipsoid function (f_7).
    - Contains the sub-problems of the Gallagher (f_{21} and f_{22}) and Lunacek bi-Rastrigin (f_{24}) functions.

Parameter Setting

Parameter Setting [Ait Elhara et al., 2016]:

- swap-range: $r_s = \lfloor d/3 \rfloor$.
- number of swaps: $n_s = d$.
- block-sizes: $s_i = \min(d, 40), \forall i, 1 \leq i \leq n_b$.
  - linear complexity in d.

Parameter Setting

Parameter Setting [Ait Elhara et al., 2016]:

- swap-range: $r_s = \lfloor d/3 \rfloor$.
- number of swaps: $n_s = d$.
- block-sizes: $s_i = \min(d, 40), \forall i, 1 \leq i \leq n_b$.
  - linear complexity in d.

Parameter Setting

Parameter Setting [Ait Elhara et al., 2016]:

- swap-range: $r_s = \lfloor d/3 \rfloor$.
- number of swaps: $n_s = d$.
- block-sizes: $s_i = \min(d, 40), \forall i, 1 \leq i \leq n_b$.
  - linear complexity in d.

Parameter Setting

Parameter Setting [Ait Elhara et al., 2016]:

- swap-range: $r_s = \lfloor d/3 \rfloor$.
- number of swaps: $n_s = d$.
- block-sizes: $s_i = \min(d, 40), \forall i, 1 \leq i \leq n_b$.
  - linear complexity in d.

Parameter Setting

Parameter Setting [Ait Elhara et al., 2016]:

- swap-range: $r_s = \lfloor d/3 \rfloor$.
- number of swaps: $n_s = d$.
- block-sizes: $s_i = \min(d, 40), \forall i, 1 \leq i \leq n_b$.
  - linear complexity in d.

General Description

- **Functions**: the 24 functions from the noiseless test-suite of [Hansen et al., 2009] normalized by $\gamma(d)$. **R** and **Q** are replaced with (different) realizations of $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$.
- **Dimensions**: 20, 40, **80**, 160, **320**, 640 (1280, 2560... excluded from post-processing for now).
- **Instance**: same instance structure as BBOB-2009, new elements ($\mathbf{B}, \mathbf{P}_{\text{left}}$, $\mathbf{P}_{\text{right}}$...) are generated with seeds: rseed + 1e6, rseed + 2e6 and rseed + 3e6...

General Description

- **Functions**: the 24 functions from the noiseless test-suite of [Hansen et al., 2009] normalized by $\gamma(d)$. **R** and **Q** are replaced with (different) realizations of $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$.
- **Dimensions**: 20, 40, **80**, 160, **320**, 640 (1280, 2560... excluded from post-processing for now).
- **Instance**: same instance structure as BBOB-2009, new elements ($\mathbf{B}, \mathbf{P}_{\text{left}}, \mathbf{P}_{\text{right}}$...) are generated with seeds: rseed + 1e6, rseed + 2e6 and rseed + 3e6...

General Description

- **Functions**: the 24 functions from the noiseless test-suite of [Hansen et al., 2009] normalized by $\gamma(d)$. **R** and **Q** are replaced with (different) realizations of $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$.
- **Dimensions**: 20, 40, **80**, 160, **320**, 640 (1280, 2560... excluded from post-processing for now).
- **Instance**: same instance structure as BBOB-2009, new elements ($\mathbf{B}, \mathbf{P}_{\text{left}}, \mathbf{P}_{\text{right}}$...) are generated with seeds: rseed + 1e6, rseed + 2e6 and rseed + 3e6...

General Description

- **Functions**: the 24 functions from the noiseless test-suite of [Hansen et al., 2009] normalized by $\gamma(d)$. **R** and **Q** are replaced with (different) realizations of $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$.
- **Dimensions**: 20, 40, **80**, 160, **320**, 640 (1280, 2560... excluded from post-processing for now).
- **Instance**: same instance structure as BBOB-2009, new elements ($\mathbf{B}, \mathbf{P}_{\text{left}}$, $\mathbf{P}_{\text{right}}$...) are generated with seeds: rseed + 1e6, rseed + 2e6 and rseed + 3e6...

Usage

After building the code for the desired language...:

- use "bbob-largescale" as suite-name (instead of "bbob" or "bbob-biobj")
- uses the single objective "bbob" observer
- make sure to have the correct dimensions (e.g., "dimensions: 20,40,80,160,320")
- post-processing changes are transparent to the user (same commands as for the other suites)

Run in parallel (when possible), example_experiment.py provides a nice interface to run multiple batches, and the post-processing handles it well.

Usage

After building the code for the desired language...:

- use "bbob-largescale" as suite-name (instead of "bbob" or "bbob-biobj"
- uses the single objective "bbob" observer
- make sure to have the correct dimensions (e.g., "dimensions: 20,40,80,160,320")
- post-processing changes are transparent to the user (same commands as for the other suites)

Run in parallel (when possible), example_experiment.py provides a nice interface to run multiple batches, and the post-processing handles it well.

Usage

After building the code for the desired language...:

- use "bbob-largescale" as suite-name (instead of "bbob" or "bbob-biobj"
- uses the single objective "bbob" observer
- make sure to have the correct dimensions (e.g., "dimensions: 20,40,80,160,320")
- post-processing changes are transparent to the user (same commands as for the other suites)

Run in parallel (when possible), example_experiment.py provides a nice interface to run multiple batches, and the post-processing handles it well.

Usage

After building the code for the desired language...:

- use "bbob-largescale" as suite-name (instead of "bbob" or "bbob-biobj"
- uses the single objective "bbob" observer
- make sure to have the correct dimensions (e.g., "dimensions: 20,40,80,160,320")
- post-processing changes are transparent to the user (same commands as for the other suites)

Run in parallel (when possible), example_experiment.py provides a nice interface to run multiple batches, and the post-processing handles it well.

Usage

After building the code for the desired language...:

- use "bbob-largescale" as suite-name (instead of "bbob" or "bbob-biobj"
- uses the single objective "bbob" observer
- make sure to have the correct dimensions (e.g., "dimensions: 20,40,80,160,320")
- post-processing changes are transparent to the user (same commands as for the other suites)

Run in parallel (when possible), example_experiment.py provides a nice interface to run multiple batches, and the post-processing handles it well.

Usage

After building the code for the desired language...:

- use "bbob-largescale" as suite-name (instead of "bbob" or "bbob-biobj"
- uses the single objective "bbob" observer
- make sure to have the correct dimensions (e.g., "dimensions: 20,40,80,160,320")
- post-processing changes are transparent to the user (same commands as for the other suites)

Run in parallel (when possible), example_experiment.py provides a nice interface to run multiple batches, and the post-processing handles it well.

testbedsetting.py

- Addition of large-scale testbed (scenario_largescalefixed, testbed_name_largescale,...)
- New parent class: "class SingleObjectiveTestbed(Testbed)"
  - GECCOBBOBTestbed(SingleObjectiveTestbed)
  - LargeScaleTestbed(SingleObjectiveTestbed)
- Suite-dependent parameters (dimensions_to_display, htmlDimsOfInterest...) moved from genericsettings.py to become attributes of the child classes GECCOBBOBTestbed and LargeScaleTestbed.
- The large-scale testbed (LargeScaleTestbed) is used when at least one dataSet has $d > 40$.
- Use of the testbed attributes (dimensions_to_display, rldDimsOfInterest... in place of the static values (5, 20,...) in various places

testbedsetting.py

- Addition of large-scale testbed (scenario_largescalefixed, testbed_name_largescale,...)
- New parent class: "class SingleObjectiveTestbed(Testbed)"
  - GECCOBBOBTestbed(SingleObjectiveTestbed)
  - LargeScaleTestbed(SingleObjectiveTestbed)
- Suite-dependent parameters (dimensions_to_display, htmlDimsOfInterest...) moved from genericsettings.py to become attributes of the child classes GECCOBBOBTestbed and LargeScaleTestbed.
- The large-scale testbed (LargeScaleTestbed) is used when at least one dataSet has $d > 40$.
- Use of the testbed attributes (dimensions_to_display, rldDimsOfInterest... in place of the static values (5, 20,...) in various places

testbedsetting.py

- Addition of large-scale testbed (scenario_largescalefixed, testbed_name_largescale,...)
- New parent class: "class SingleObjectiveTestbed(Testbed)"
  - GECCOBBOBTestbed(SingleObjectiveTestbed)
  - LargeScaleTestbed(SingleObjectiveTestbed)
- Suite-dependent parameters (dimensions_to_display, htmlDimsOfInterest...) moved from genericsettings.py to become attributes of the child classes GECCOBBOBTestbed and LargeScaleTestbed.
- The large-scale testbed (LargeScaleTestbed) is used when at least one dataSet has $d > 40$.
- Use of the testbed attributes (dimensions_to_display, rldDimsOfInterest... in place of the static values (5, 20,...) in various places

testbedsetting.py

- Addition of large-scale testbed (scenario_largescalefixed, testbed_name_largescale,...)
- New parent class: "class SingleObjectiveTestbed(Testbed)"
  - GECCOBBOBTestbed(SingleObjectiveTestbed)
  - LargeScaleTestbed(SingleObjectiveTestbed)
- Suite-dependent parameters (dimensions_to_display, htmlDimsOfInterest...) moved from genericsettings.py to become attributes of the child classes GECCOBBOBTestbed and LargeScaleTestbed.
- The large-scale testbed (LargeScaleTestbed) is used when at least one dataSet has $d > 40$.
- Use of the testbed attributes (dimensions_to_display, rldDimsOfInterest... in place of the static values (5, 20,...) in various places

testbedsetting.py

- Addition of large-scale testbed (scenario_largescalefixed, testbed_name_largescale,...)
- New parent class: "class SingleObjectiveTestbed(Testbed)"
  - GECCOBBOBTestbed(SingleObjectiveTestbed)
  - LargeScaleTestbed(SingleObjectiveTestbed)
- Suite-dependent parameters (dimensions_to_display, htmlDimsOfInterest...) moved from genericsettings.py to become attributes of the child classes GECCOBBOBTestbed and LargeScaleTestbed.
- The large-scale testbed (LargeScaleTestbed) is used when at least one dataSet has $d > 40$.
- Use of the testbed attributes (dimensions_to_display, rldDimsOfInterest... in place of the static values (5, 20,...) in various places

testbedsetting.py

- Addition of large-scale testbed (scenario_largescalefixed, testbed_name_largescale,...)
- New parent class: "class SingleObjectiveTestbed(Testbed)"
  - GECCOBBOBTestbed(SingleObjectiveTestbed)
  - LargeScaleTestbed(SingleObjectiveTestbed)
- Suite-dependent parameters (dimensions_to_display, htmlDimsOfInterest...) moved from genericsettings.py to become attributes of the child classes GECCOBBOBTestbed and LargeScaleTestbed.
- The large-scale testbed (LargeScaleTestbed) is used when at least one dataSet has $d > 40$.
- Use of the testbed attributes (dimensions_to_display, rldDimsOfInterest... in place of the static values (5, 20,...) in various places

testbedsetting.py

- Addition of large-scale testbed (scenario_largescalefixed, testbed_name_largescale,...)
- New parent class: "class SingleObjectiveTestbed(Testbed)"
  - GECCOBBOBTestbed(SingleObjectiveTestbed)
  - LargeScaleTestbed(SingleObjectiveTestbed)
- Suite-dependent parameters (dimensions_to_display, htmlDimsOfInterest...) moved from genericsettings.py to become attributes of the child classes GECCOBBOBTestbed and LargeScaleTestbed.
- The large-scale testbed (LargeScaleTestbed) is used when at least one dataSet has $d > 40$.
- Use of the testbed attributes (dimensions_to_display, rldDimsOfInterest... in place of the static values (5, 20,...) in various places

Post-Processing

- How to use : as in BBOB small scale, for instance
  python -m bbob_pproc ALG1 ALG2 ALG3
- Rungeneric : 1, 2, and many dataset
- Best algorithm : (demo) best 2009 $\rightarrow$ best 2016
- File generated : pdf, svg, ...
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- HTML : Figure and Table with updated caption, legend; Link

Question : The function name should be changed?

Post-Processing

- How to use : as in BBOB small scale, for instance
  python -m bbob_pproc ALG1 ALG2 ALG3
- Rungeneric : 1, 2, and many dataset
- Best algorithm : (demo) best 2009 $\rightarrow$ best 2016
- File generated : pdf, svg, ...
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- HTML : Figure and Table with updated caption, legend; Link

Question : The function name should be changed?

Post-Processing

- How to use : as in BBOB small scale, for instance
  python -m bbob_pproc ALG1 ALG2 ALG3
- Rungeneric : 1, 2, and many dataset
- Best algorithm : (demo) best 2009 $\rightarrow$ best 2016
- File generated : pdf, svg, ...
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- HTML : Figure and Table with updated caption, legend; Link

Question : The function name should be changed?

Post-Processing

- How to use : as in BBOB small scale, for instance
  python -m bbob_pproc ALG1 ALG2 ALG3
- Rungeneric : 1, 2, and many dataset
- Best algorithm : (demo) best 2009 $\rightarrow$ best 2016
- File generated : pdf, svg, ...
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- HTML : Figure and Table with updated caption, legend; Link

Question : The function name should be changed?

Post-Processing

- How to use : as in BBOB small scale, for instance
  python -m bbob_pproc ALG1 ALG2 ALG3
- Rungeneric : 1, 2, and many dataset
- Best algorithm : (demo) best 2009 $\rightarrow$ best 2016
- File generated : pdf, svg, ...
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- HTML : Figure and Table with updated caption, legend; Link

Question : The function name should be changed?

Post-Processing

- How to use : as in BBOB small scale, for instance
  python -m bbob_pproc ALG1 ALG2 ALG3
- Rungeneric : 1, 2, and many dataset
- Best algorithm : (demo) best 2009 $\rightarrow$ best 2016
- File generated : pdf, svg, ...
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- HTML : Figure and Table with updated caption, legend; Link

Question : The function name should be changed?

Post-Processing

- How to use : as in BBOB small scale, for instance
  python -m bbob_pproc ALG1 ALG2 ALG3
- Rungeneric : 1, 2, and many dataset
- Best algorithm : (demo) best 2009 $\rightarrow$ best 2016
- File generated : pdf, svg, ...
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- HTML : Figure and Table with updated caption, legend; Link

Question : The function name should be changed?

Post-Processing

- How to use : as in BBOB small scale, for instance
  python -m bbob_pproc ALG1 ALG2 ALG3
- Rungeneric : 1, 2, and many dataset
- Best algorithm : (demo) best 2009 $\rightarrow$ best 2016
- File generated : pdf, svg, ...
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- HTML : Figure and Table with updated caption, legend; Link

Question : The function name should be changed?

LaTeX

- New latex files: templateBBOB**LS**article.tex, templateBBOB**LS**cmp.tex, templateBBOB**LS**many.tex
- Best algorithm: (demo) best 2009 $\rightarrow$ best 2016
- Dimension: [5, 20] $\rightarrow$ [80, 320] in large scale
- Figures and Tables: updated caption, legend

Benchmarked Algorithms

sep-CMA-ES [Ros and Hansen, 2008]: a budget of $1e4 \times d$, Python code.

VD-CMA-ES [Akimoto et al., 2014]: a budget of $1e3 \times d$, Octave/Matlab code.

LM-CMA-ES [Loshchilov, 2014]: a budget of $1e4 \times d$, C/C++ code.

Benchmarked Algorithms

sep-CMA-ES [Ros and Hansen, 2008]: a budget of $1e4 \times d$, Python code.

VD-CMA-ES [Akimoto et al., 2014]: a budget of $1e3 \times d$, Octave/Matlab code.

LM-CMA-ES [Loshchilov, 2014]: a budget of $1e4 \times d$, C/C++ code.

Benchmarked Algorithms

sep-CMA-ES [Ros and Hansen, 2008]: a budget of $1e4 \times d$, Python code.

VD-CMA-ES [Akimoto et al., 2014]: a budget of $1e3 \times d$, Octave/Matlab code.

LM-CMA-ES [Loshchilov, 2014]: a budget of $1e4 \times d$, C/C++ code.

Benchmarked Algorithms

sep-CMA-ES [Ros and Hansen, 2008]: a budget of $1e4 \times d$, Python code.

VD-CMA-ES [Akimoto et al., 2014]: a budget of $1e3 \times d$, Octave/Matlab code.

LM-CMA-ES [Loshchilov, 2014]: a budget of $1e4 \times d$, C/C++ code.

Some Plots

??

References I

Ait Elhara, O., Auger, A., and Hansen, N. (2016).
Permuted orthogonal block-diagonal transformation matrices for large scale optimization benchmarking.
Inria.
Preprint with appendix on
https://hal.inria.fr/hal-01308566.

Akimoto, Y., Auger, A., and Hansen, N. (2014).
Comparison-based natural gradient optimization in high dimension.
In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 373–380. ACM.

References II


Hansen, N., Finck, S., Ros, R., and Auger, A. (2009).
Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions.
Research Report RR-6829, INRIA.


Loshchilov, I. (2014).
A computationally efficient limited memory cma-es for large scale optimization.
In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 397–404. ACM.

References III

Ros, R. and Hansen, N. (2008).
A simple modification in cma-es achieving linear time and space complexity.
In *Parallel Problem Solving from Nature–PPSN X*, pages 296–305. Springer.