Package 'sezgi'

September 15, 2026

Title Metaheuristic Optimization with a 'Rust' Core

Version 0.1.1

Description Build and run metaheuristic optimization algorithms as serializable component graphs executed by a 'Rust' core, with reproducible, bit-exact trajectories shared across the 'R' and 'Python' frontends. Includes population and local-search algorithm presets, standard benchmark suites (BBOB, CEC 2014/2017/2022, TSP), multi-objective indicators, structural-bias diagnostics and statistical comparison tools.

License MIT + file LICENSE

Copyright sezgi authors, except as documented in inst/COPYRIGHTS (bundled benchmark data and quoted reference excerpts)

Encoding UTF-8

Depends R (>= 4.2)

Imports R6 (>= 2.4.0)

Suggests testthat (>= 3.0.0), jsonlite

SystemRequirements Cargo (Rust's package manager), rustc (>= 1.88)

URL https://github.com/tdelphi1981/sezgi

BugReports https://github.com/tdelphi1981/sezgi/issues

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/sezgi/MSRV 1.88.0

NeedsCompilation yes

Author Tolga Berber [aut, cre] (ORCID: <https://orcid.org/0000-0002-6487-5581>), Beyzanur Siyah [aut] (ORCID: <https://orcid.org/0000-0002-2071-3724>), Emir Karayağız [aut] (ORCID: <https://orcid.org/0009-0005-1673-621X>), The authors of the dependency Rust crates [ctb] (see inst/AUTHORS file for details)

Maintainer Tolga Berber <tolga.berber@fen.ktu.edu.tr>

Repository CRAN

Date/Publication 2026-09-15 10:40:02 UTC

Contents

`Algorithm` *Subclassable R6 base for an engine-hosted algorithm.*

Description

Mirrors ‘sezgi.Algorithm‘ (‘py-sezgi/python/sezgi/algorithm.py‘, M4-1 Task 3) closely: a subclass implements ‘generate(pop, ctx)‘ (REQUIRED) and may optionally override ‘initialize_population(n, space, ctx)‘ (RULING 8: named ‘initialize_population‘, NOT ‘initialize‘ – R6 reserves ‘initialize‘ for the constructor, so Python’s ‘initialize(n, ctx)‘ hook name cannot be reused here; a deliberate, documented divergence) and ‘validate_space(space)‘. ‘run()‘ is the sole entry point, wiring the instance through T3’s ‘sz_solve_r_generator‘/ ‘sz_solve_r_generator_bbob‘ bridge.

Details

‘pop‘ (handed to ‘generate()‘/a ‘PopulationAlgorithm‘’s ‘select()‘/ ‘vary()‘/a ‘LocalSearch‘’s ‘neighbor()‘) is ‘list(x, f)‘: ‘f‘ is a numeric vector, the current population’s fitness values, one entry per individual. ‘x‘ is an n x dim numeric MATRIX (rows = individuals) for a SINGLE-Float-block space (the ergonomic, common case – ‘pop$x[i,]‘, ‘ncol(pop$x)‘); for every OTHER space shape (multi-block, or a single NON-Float block – Int/Categorical/Binary/Permutation), ‘x‘ is instead a plain (unnamed) ‘list‘ of per-individual values, one entry per individual (bare for a single block, a further per-block ‘list‘ for multiple blocks) – the SAME single-vs-multi-block convention ‘Problem$evaluate(x)‘’s own ‘x‘ argument already uses. A ‘generate()‘/‘vary()‘/‘neighbor()‘ return value mirrors this: a matrix (only meaningful when ‘pop$x‘ was itself a matrix) or a plain ‘list‘ of offspring x-values.

‘ctx‘ is an ‘environment‘ with ‘$rng‘ (an already-wrapped RNG handle – ‘ctxrngnext_f64()‘ (a double in ‘[0, 1)‘), ‘ctxrngnext_below(n)‘ (an integer-valued double in ‘[0, n)‘), ‘ctxrngsplit(child_id)‘

(an independent child stream)), ‘$iteration‘ (a double, the engine’s own 0-based generation counter – always ‘0‘ for an ‘initialize_population()‘ call), and ‘$space‘ (the block-descriptor ‘list‘ for the space being solved – one entry per block, each a named ‘list‘ with a ‘type‘ field plus that block’s own fields; the SAME shape ‘validate_space(space)‘’s own ‘space‘ argument uses).

Override-detection caveat: ‘initialize_population‘/‘validate_space‘ are detected as overridden by comparing the resolved method against this class’s own declared default at the CLASS level (body + formals, ignoring environment – see ‘R/algorithm.R‘’s own module doc for the exact spelling) – a subclass method whose body happens to be byte-identical to the base default (e.g. a copy-pasted ‘stop(...)‘ call with the same message) is therefore treated as NOT overridden even though it was technically redeclared; harmless today since both defaults are no-op-equivalent (‘initialize_population‘’s default always errors if ever reached, ‘validate_space‘’s default always accepts), but worth knowing.

See this file’s own module doc (top of ‘R/algorithm.R‘) for the full rationale behind these choices and the ‘log_dir‘ rejection.

Methods:

‘generate(pop, ctx)‘ REQUIRED: produce this generation’s offspring. Default: ‘stop()‘s with "not implemented".

‘initialize_population(n, space, ctx)‘ Optional: produce the initial population of size ‘n‘. Default: NOT overridden – ‘run()‘ never calls this default body; the engine’s own Rust ‘init/uniform‘ initializer runs instead, with NO R call for population initialization at all (byte-identical to a raw ‘sz_solve_r_generator‘ call with ‘initializer = NULL‘).

‘validate_space(space)‘ Optional build-time veto: ‘stop()‘ to reject ‘space‘ BEFORE any ‘generate()‘/‘initialize_population()‘ call. Default: no-op (accepts any space).

‘run(problem, budget, pop_size = 50, seed = 0, run_id = 0, name = NULL, log_dir = NULL)‘ Runs this algorithm’s hooks INSIDE the Rust engine loop. ‘problem‘: a ‘Problem‘ subclass instance (routed through ‘sz_as_problem()‘) OR a built-in problem descriptor (see [sz_builtin_bbob()]). ‘log_dir‘: NOT supported (see module doc) – a non-‘NULL‘ value raises a clear error. Returns an ‘sz_result‘ (see [‘.sz_wrap_result‘]). NOTE – known, INTENTIONAL divergence from ‘sezgi.Algorithm.run(problem, budget, seed=0, pop_size=50, log_dir=None, ...)‘ (pysezgi/python/sezgi/algorithm.py): ‘pop_size‘ and ‘seed‘ are swapped, and ‘log_dir‘ sits last rather than third, kept as-is (not reordered to match) because every anchored test and example already calls ‘run()‘ positionally against THIS order – reshuffling would break them for no benefit in a language where both sides are typically called with named arguments anyway.

Methods

Public methods:

- `Algorithm$generate()`
- `Algorithm$initialize_population()`
- `Algorithm$validate_space()`
- `Algorithm$run()`
- `Algorithm$clone()`

`Algorithm$generate()`:

Usage:

```
Algorithm$generate(pop, ctx)
```

`Algorithm$initialize_population()`:

Usage:

```
Algorithm$initialize_population(n, space, ctx)
```

`Algorithm$validate_space()`:

Usage:

```
Algorithm$validate_space(space)
```

`Algorithm$run()`:

Usage:

```
Algorithm$run(
  problem,
  budget,
  pop_size = 50,
  seed = 0,
  run_id = 0,
  name = NULL,
  log_dir = NULL
)
```

`Algorithm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
Algorithm$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
RandomStep <- R6::R6Class("RandomStep", inherit = Algorithm, public = list(
  generate = function(pop, ctx) {
    step <- vapply(seq_len(ncol(pop$x)), function(i) ctx$rng$next_f64() - 0.5, numeric(1))
     matrix(pop$x[1, ] + step, nrow = 1)
  }
))
res <- RandomStep$new()$run(sz_builtin_bbob(1, 2, 1), budget = 100, pop_size = 1, seed = 1)
res$evals_used
```

`DifferentialEvolution` *Differential Evolution – delegates to ‘sz_preset_de_rand_1‘ (default, DE/rand/1/bin), ‘sz_preset_de_best_1‘ (DE/best/1/bin), or ‘sz_preset_jde‘ (self-adaptive jDE), selected by ‘variant‘ (one of ‘"rand_1"‘, ‘"best_1"‘, ‘"jde"‘) at ‘$new()‘ time. All three presets share the identical ‘(pop_size, budget)‘ signature, so ‘variant‘ is the only dispatch axis – no problem introspection needed (unlike [GeneticAlgorithm]’s space-driven auto-dispatch).*

Description

See [sz_preset_classes]’s own doc for the problem-form support matrix and result shape (identical here).

Methods

Public methods:

- `DifferentialEvolution$new()`
- `DifferentialEvolution$run()`
- `DifferentialEvolution$clone()`

`DifferentialEvolution$new()`:

Usage:

```
DifferentialEvolution$new(pop_size = 20, variant = "rand_1", ...)
```

`DifferentialEvolution$run()`:

Usage:

```
DifferentialEvolution$run(problem, budget, seed = 0, run_id = 0)
```

`DifferentialEvolution$clone()`: The objects of this class are cloneable with this method.

Usage:

```
DifferentialEvolution$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`EvalSession` *The ask/tell evaluation session – see the module doc.*

Description

The ask/tell evaluation session – see the module doc.

Usage

```
EvalSession
```

`FeatureSelection` *Binary feature-selection search over a 2D dataset's columns.*

Description

Mirrors ‘sezgi.recipes.FeatureSelection‘ (‘py-sezgi/python/sezgi/recipes.py‘) exactly.

Details

‘space()‘ is ‘sz_space(sz_binary(n_features))‘: a genotype is a length-‘n_features‘ logical mask (per [Problem]'s own genotype conversion table – a Binary block converts to a logical vector), ‘TRUE‘ selecting a column.

MINIMIZE convention: ‘scorer(X_sub, y)‘ must return a LOWER-is-better number (e.g. an error/loss/residual metric, NOT accuracy/R^2/any higher-is-better score) – ‘evaluate()‘ never negates or inverts it. An sklearn-shaped cross-validated ACCURACY scorer would need to be wrapped to flip its sign before it could be used here (this package has no ML-package dependency of its own – the sketch below is a COMMENT only, never an actual call anywhere in this file):

```
# scorer <- function(X_sub, y) {
#   # a cross-validated accuracy is HIGHER-is-better -- negate it so
#   # lower is better, matching this class's minimize convention.
#   -mean(cross_val_accuracy(X_sub, y))
# }
# FeatureSelection$new(X, y, scorer, penalty = 0.01)
```

‘evaluate(mask)‘ = ‘scorer(X[, mask, drop = FALSE], y) + penalty * (popcount(mask) / n_features)‘ – the penalty term is a fraction of the FULL feature count (‘popcount / n_features‘, not a raw popcount), so it stays on a comparable scale to ‘scorer‘'s own output regardless of ‘n_features‘, and its size is ‘penalty‘ at the all-features mask, ‘0‘ at the empty mask. Larger ‘penalty‘ biases the search toward smaller feature subsets; ‘penalty = 0‘ (default) is a pure ‘scorer‘-value search with no feature-count preference of its own.

‘X[, mask, drop = FALSE]‘: the ‘drop = FALSE‘ is LOAD-BEARING – without it, a single-‘TRUE‘ mask would subset ‘X‘ down to a plain vector (R's single-bracket auto-drop for a one-column result), silently changing ‘X_sub‘'s shape from a matrix to a vector depending on the mask's own popcount and breaking any ‘scorer‘ that assumes a consistent ‘(n_samples, k)‘ matrix shape (e.g. anything calling ‘ncol(X_sub)‘ or doing matrix algebra on it) – the R analogue of M4-1 Task 6's Python bool-indexing trap, pinned here with its own regression test (‘tests/testthat/test-oop-recipes.R‘).

EMPTY MASK (‘popcount == 0‘): ‘scorer‘ is NEVER called – ‘X[, mask, drop = FALSE]‘ would be an ‘(n_samples, 0)‘ matrix, and most real scorers (a model fit, a distance-correlation-like statistic, ...) cannot meaningfully score zero columns; the ‘popcount == 0‘ case is instead handled directly, as a DOCUMENTED SENTINEL: ‘evaluate(rep(FALSE, n_features)) == Inf‘. ‘Inf‘ was chosen over, say, a large-but-finite number because it is unambiguous (no dataset-dependent magnitude to pick or accidentally beat by a legitimately bad but non-empty selection) and it sorts correctly under every consumer's own comparison (‘<‘) with no special-casing needed – the empty mask is simply never the minimizer of any run that has at least one non-empty candidate in its population, which every population-based search here always does.

Methods

‘new(X, y, scorer, penalty = 0)‘ ‘X‘ a 2D matrix/data.frame (coerced via ‘as.matrix()‘), ‘y‘ the target (any vector/type ‘scorer‘ accepts), ‘scorer(X_sub, y) -> numeric(1)‘ a caller-supplied minimize-convention scoring function, ‘penalty‘ a numeric scalar (default ‘0‘, no feature-count preference).

‘space()‘ ‘sz_space(sz_binary(n_features))‘, where ‘n_features = ncol(X)‘.

‘evaluate(mask)‘ See the class doc above.

Super class

`Problem` -> `FeatureSelection`

Methods

Public methods:

- `FeatureSelection$new()`
- `FeatureSelection$space()`
- `FeatureSelection$evaluate()`
- `FeatureSelection$clone()`

`FeatureSelection$new()`:

Usage:

```
FeatureSelection$new(X, y, scorer, penalty = 0)
```

`FeatureSelection$space()`:

Usage:

```
FeatureSelection$space()
```

`FeatureSelection$evaluate()`:

Usage:

```
FeatureSelection$evaluate(mask)
```

`FeatureSelection$clone()`: The objects of this class are cloneable with this method.

Usage:

```
FeatureSelection$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

GeneticAlgorithm *Genetic Algorithm – auto-dispatches to 'sz_preset_ga_real'/'ga_perm'/ 'ga_bin'/'ga_int'/'ga_cat' ('R/000-wrappers.R') based on 'problem''s own search space, read via 'prob$space()'/'.sz_space_to_blocks()':*

Description

all-Float 'sz_preset_ga_real'

all-Permutation 'sz_preset_ga_perm'

all-Binary 'sz_preset_ga_bin'

all-Int 'sz_preset_ga_int'

all-Categorical 'sz_preset_ga_cat'

Details

A space MIXING block kinds (e.g. Float + Int together) has no 'ga_*' preset in this milestone – '$run()' raises a clear error naming the limitation (mirrors py-sezgi's 'GeneticAlgorithm''s own identical 'NotImplementedError', M4-1 Task 5 deferral (d)).

'representation' (a constructor kwarg, one of '"real"'/'"perm"'/ '"bin"'/'"int"'/'"cat"') OVERRIDES auto-dispatch entirely – block-kind introspection is skipped whenever it is given.

After a '$run()' call, '$dispatched_representation' holds the representation actually engaged – an introspectable field proving which preset ran, independent of 'representation' itself (which stays 'NULL' under auto-dispatch).

See [sz_preset_classes]'s own doc for the problem-form support matrix and result shape (identical here).

Methods

Public methods:

- `GeneticAlgorithm$new()`
- `GeneticAlgorithm$run()`
- `GeneticAlgorithm$clone()`

`GeneticAlgorithm$new()`:

Usage:

```
GeneticAlgorithm$new(pop_size = 20, representation = NULL, ...)
```

`GeneticAlgorithm$run()`:

Usage:

```
GeneticAlgorithm$run(problem, budget, seed = 0, run_id = 0)
```

`GeneticAlgorithm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
GeneticAlgorithm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`LocalSearch` *Family base for single-trajectory local search (hill-climbing/ SA-shaped).*

Description

Inherits [Algorithm]. A subclass implements ‘neighbor(x, ctx)‘ (REQUIRED – proposes one perturbed candidate from the current point) and may optionally override ‘accept(f_old, f_new, ctx)‘ (default: greedy, ‘f_new <= f_old‘). Designed for ‘pop_size = 1‘ – ‘run()‘ is overridden here to default AND ENFORCE ‘pop_size = 1‘ (‘stop()‘ otherwise).

Details

Mirrors ‘sezgi.LocalSearch‘ (‘py-sezgi/python/sezgi/algorithm.py‘, M4-1 Task 4) exactly, including its honest structural limitation:

‘pop‘ (handed to ‘generate()‘, which itself calls ‘neighbor()‘/ ‘accept()‘) is ‘list(x, f)‘, at ‘pop_size = 1‘ always a single individual: ‘f‘ is a length-1 numeric vector, the current point’s fitness; ‘x‘ is a length-1 x dim numeric MATRIX (a single row) for a SINGLE-Float-block space (‘x[1,]‘ is the current point), or a length-1 ‘list‘ otherwise (‘x[[1]]‘ is the current point, bare for a single non-Float block, a further per-block ‘list‘ for multiple blocks) – the SAME single-vs-multi-block convention ‘Problem$ evaluate(x)‘’s own ‘x‘ uses (‘.sz_pop_at(pop, 1)‘, used internally by ‘generate()‘ below, abstracts over both shapes). ‘neighbor()‘’s return value is a SINGLE x-value in that same convention (NOT a list – ‘generate()‘ wraps it). ‘ctx‘ is an ‘environment‘ with ‘$rng‘ (a wrapped RNG handle – ‘rngnext_f64()‘/‘rngnext_below(n)‘/‘rng split(child_id)‘), ‘$iteration‘ (a double), and ‘$space‘ (the block- descriptor ‘list‘ for the space being solved). See [Algorithm]’s own doc for the full detail on both, and for the override-detection caveat that applies to any ‘initialize_population‘/‘validate_space‘ override this subclass adds (a body byte-identical to ‘Algorithm‘’s own default is treated as NOT overridden, even if redeclared).

ACCEPT() DESIGN (read carefully – this is not the naive reading of "accept decides whether the engine keeps the point"): this base runs over the engine’s default ‘replace/mu-plus-lambda‘ replacer. At ‘pop_size = 1‘ (mu = 1), with ‘generate()‘ here always returning exactly ONE offspring (lambda = 1), that replacer ALWAYS keeps the strictly-better (or, on an exact tie, the OLD) of {current, neighbor} as the NEXT call’s ‘pop$x‘/‘pop$f‘’s own first entry – a structural, UNCONDITIONAL guarantee of ‘replace/mu-plus-lambda‘ itself, not a decision ‘accept()‘ makes. Two direct consequences: (1) the fitness this base reads on the call following a ‘neighbor()‘ proposal is ALWAYS ‘<= f_old‘ – ‘accept()‘ can therefore NEVER actually be called with an ‘f_new‘ that is objectively WORSE than ‘f_old‘; a worse neighbor’s exact fitness is never even surfaced back to R by this bridge. **This base’s ‘accept()‘ therefore CANNOT implement true simulated-annealing-style "sometimes accept a worse move" semantics – that would require intercepting the replacer’s own decision, which this design deliberately does NOT attempt.** (2) Given (1), the DEFAULT

'accept' ('f_new <= f_old') is a TAUTOLOGY over every pair this base can ever actually present to it – it is ALWAYS true, faithfully mirroring (not fighting) what 'replace/mu-plus-lambda' already unconditionally enforces at 'pop_size = 1'. What 'accept()' DOES meaningfully control: whether this base's OWN internally-tracked anchor ('self$current_x'/'self$current_f', the point the NEXT 'neighbor()' call is proposed from) ADVANCES to match the engine's already-decided outcome, or stays where it was – i.e. 'accept()' governs this base's own SEARCH TRAJECTORY, never population membership (which the replacer alone decides, unconditionally, at 'pop_size = 1').

Super class

`Algorithm` -> `LocalSearch`

Methods

Public methods:

- `LocalSearch$neighbor()`
- `LocalSearch$accept()`
- `LocalSearch$run()`
- `LocalSearch$generate()`
- `LocalSearch$clone()`

`LocalSearch$neighbor()`:

Usage:

```
LocalSearch$neighbor(x, ctx)
```

`LocalSearch$accept()`:

Usage:

```
LocalSearch$accept(f_old, f_new, ctx)
```

`LocalSearch$run()`:

Usage:

```
LocalSearch$run(
  problem,
  budget,
  pop_size = 1,
  seed = 0,
  run_id = 0,
  name = NULL,
  log_dir = NULL
)
```

`LocalSearch$generate()`:

Usage:

```
LocalSearch$generate(pop, ctx)
```

`LocalSearch$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LocalSearch$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`MixedTuning` *The general "tune anything" door: binds a caller-supplied ‘objective(x) -> numeric(1)‘ over ANY declared ‘sz_space(...)‘ – a single block (‘sz_float(...)‘, ‘sz_categorical(...)‘, ...) or a multi-block space. ‘evaluate(x)‘ delegates to ‘objective‘ UNCHANGED – ‘x‘’s exact shape follows [Problem]’s own genotype conversion table (a bare converted value for a single-block space, an unnamed ‘list‘ of per-block converted values in ‘space()‘’s own block order for a multi-block one).*

Description

Mirrors ‘sezgi.recipes.MixedTuning‘ (‘py-sezgi/python/sezgi/recipes.py‘) exactly.

Details

Note [GeneticAlgorithm] auto-dispatches only over a SINGLE-kind space (all-Float, all-Binary, ...) – it raises a clear error naming the limitation on a genuinely Mixed space (see its own doc); a mixed-space ‘MixedTuning‘ instance is instead run via a hand-built ‘gen/compound‘ algorithm spec passed to ‘sezgi:::sz_solve_r_problem()‘ directly (see ‘tests/testthat/test-oop-recipes.R‘’s own worked example), or ‘evaluate()‘d directly for a non-engine use (grid search, a script sanity check, ...). A single-kind ‘MixedTuning‘ space (e.g. Float-only, tuning several continuous hyperparameters at once) runs through [GeneticAlgorithm] exactly like any other single-kind [Problem].

Methods

‘new(space, objective)‘ ‘space‘ an ‘sz_space(...)‘ object, ‘objective(x) -> numeric(1)‘ a caller-supplied minimize-convention function.

‘space()‘ Returns the constructor’s own ‘space‘, unchanged.

‘evaluate(x)‘ Returns ‘objective(x)‘, unchanged.

Super class

`Problem` -> `MixedTuning`

Methods

Public methods:

- `MixedTuning$new()`
- `MixedTuning$space()`
- `MixedTuning$evaluate()`
- `MixedTuning$clone()`

`MixedTuning$new()`:

Usage:

```
MixedTuning$new(space, objective)
```

`MixedTuning$space()`:

Usage:

```
MixedTuning$space()
```

`MixedTuning$evaluate()`:

Usage:

```
MixedTuning$evaluate(x)
```

`MixedTuning$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MixedTuning$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`NSGA2` *Class skin over [sz_nsga2()] (‘R/mo.R‘) – NOT one of the 34 ‘sz_preset_*‘ builders (NSGA-II is not built on the scalar Engine/Registry/Generator/AlgorithmSpec machinery every ‘sz_preset_*‘ targets, see this file’s own module doc). ZERO new MO capability: ‘$run()‘ delegates to ‘sz_nsga2()‘ VERBATIM, same positional/keyword arguments, same return value shape.*

Description

Constructor kwargs mirror ‘sz_nsga2()‘’s own "algorithm configuration" parameters – ‘pop_size‘ plus every variation-operator knob (‘eta_c‘, ‘eta_m‘, ‘p_c‘, ‘p_m‘, ‘p_c_bin‘, ‘p_m_bin‘, ‘p_c_cat‘, ‘p_m_cat‘), the ones that describe the algorithm instance itself, independent of which problem it is pointed at. ‘$run()‘’s own arguments mirror ‘sz_nsga2()‘’s remaining "this particular problem/run" parameters (‘problem‘, ‘dim‘, ‘budget‘, ‘m‘, ‘k‘, ‘l‘, ‘seed‘, ‘log_dir‘, ‘label‘) – ‘m‘/‘k‘/‘l‘ describe the PROBLEM being solved (‘m‘ = objective count for dtlz/wfg; ‘k‘/‘l‘ = WFG’s own shape parameters), not the algorithm, so they belong with ‘$run()‘ rather than ‘$new()‘, exactly mirroring ‘sz_nsga2()‘’s own per-problem-family validation.

Details

‘NSGA2$new(pop_size = P, ...)$run(problem, dim, budget, m = M, seed = S, ...)‘ is IDENTICAL to calling ‘sz_nsga2(problem, dim, P, budget, m = M, seed = S, ...)‘ directly (see ‘test-oop-builtins.R‘’s own anchored equivalence tests, a ZDT anchor and a WFG anchor where ‘k‘/‘l‘ matter).

‘$run()‘’s return value is ‘sz_nsga2()‘’s OWN list shape (‘individuals‘, ‘objectives‘, ‘front0‘, ‘evals_used‘, ‘violations‘ when constrained) – NOT an ‘sz_result‘ (T4’s ‘.sz_wrap_result()‘ shape assumes a single-objective run with one best point, which does not fit NSGA-II’s multi-objective Pareto-front result).

Methods

Public methods:

- NSGA2$new()
- NSGA2$run()
- NSGA2$clone()

NSGA2$new():

Usage:

```
NSGA2$new(
  pop_size,
  eta_c = 20,
  eta_m = 20,
  p_c = 0.9,
  p_m = NULL,
  p_c_bin = 0.9,
  p_m_bin = NULL,
  p_c_cat = 0.9,
  p_m_cat = NULL
)
```

NSGA2$run():

Usage:

```
NSGA2$run(
  problem,
  dim,
  budget,
  m = NULL,
  seed = 0,
  k = NULL,
  l = NULL,
  log_dir = NULL,
  label = NULL
)
```

NSGA2$clone(): The objects of this class are cloneable with this method.

Usage:

```
NSGA2$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`PopulationAlgorithm` *Family base for population-style algorithms (GA/DE/ES-shaped).*

Description

Inherits [Algorithm]. A subclass implements ‘vary(parents, ctx)‘ (REQUIRED – turns a parent pool into offspring) and may optionally override ‘select(pop, k, ctx)‘ (default: k-fold binary tournament, tournament size 2, minimization). ‘generate(pop, ctx)‘ is NOT meant to be overridden (override ‘select‘/‘vary‘ instead) – it is composed here from ‘select‘ + ‘vary‘.

Details

Mirrors ‘sezgi.PopulationAlgorithm‘ (‘py-sezgi/python/sezgi/ algorithm.py‘, M4-1 Task 4) exactly, including the PROVEN-uniform two-fixed-draw index-shift tournament formula (ported verbatim below, with its own determinism comment).

‘pop‘ (handed to ‘select()‘/‘generate()‘) is ‘list(x, f)‘: ‘f‘ is a numeric vector of per-individual fitness values; ‘x‘ is an n x dim numeric MATRIX (rows = individuals) for a SINGLE-Float-block space, or a plain (unnamed) ‘list‘ of per-individual values otherwise (bare for a single non-Float block, a further per-block ‘list‘ for multiple blocks) – the SAME single-vs-multi-block convention ‘Problem$ evaluate(x)‘’s own ‘x‘ uses. ‘vary()‘’s return value mirrors this (a matrix or a ‘list‘). ‘ctx‘ is an ‘environment‘ with ‘$rng‘ (a wrapped RNG handle – ‘rngnext_f64()‘/‘rngnext_below(n)‘/‘rngsplit (child_id)‘), ‘$iteration‘ (a double), and ‘$space‘ (the block- descriptor ‘list‘ for the space being solved). See [Algorithm]’s own doc for the full detail on both, and for the override-detection caveat that applies to any ‘initialize_population‘/‘validate_space‘ override this subclass adds (a body byte-identical to ‘Algorithm‘’s own default is treated as NOT overridden, even if redeclared).

Methods:

‘select(pop, k, ctx)‘ Default: k-fold binary tournament selection (tournament size 2; minimization – lower fitness wins). For each of the ‘k‘ requested parent slots, INDEPENDENTLY draws exactly two values from ‘ctx$rng‘ (with ‘n = ‘ the current population size): ‘i <- ctxrngnext_below(n)‘; ‘j <- ctxrngnext_below(n - 1); if (j >= i) j <- j + 1‘ (for ‘n > 1‘). This is the standard "index-shift" trick for drawing two DISTINCT indices from ‘[0, n)‘ using exactly two draws and no rejection/ retry loop – the draw count per slot is always EXACTLY 2 for ‘n > 1‘, so the whole sequence is hand-traceable given a fixed RNG seed and a known ‘n‘. The individuals at ‘i‘ and ‘j‘ then "fight": ‘pop$f[i+1] <= pop$f[j+1]‘ wins as ‘i‘ – this is also the EXACT tie-break rule: on an exact tie, the first-drawn contestant (‘i‘) always wins, never ‘j‘. ‘n == 1‘ (a population of one) is a degenerate edge case with no second index to draw: only ‘i <- ctxrngnext_below(1)‘ (always ‘0‘) is drawn, and ‘i‘ wins trivially, costing 1 draw instead of 2. Returns a ‘list‘ of exactly ‘k‘ parents (individual x-values, one per slot, in slot order) – ties/duplicates across slots are possible and expected.

‘vary(parents, ctx)‘ REQUIRED: turn a parent pool (a ‘list‘ of ‘length(parents)‘ x-values) into this generation’s offspring (a matrix or a ‘list‘, see the module doc). Offspring COUNT is entirely this method’s own choice, independent of ‘length(parents)‘.

‘generate(pop, ctx)‘ Composes ‘select()‘ then ‘vary()‘ – NOT meant to be overridden. Arity contract (PINNED): always calls ‘self$select(pop, n, ctx)‘ where ‘n‘ is the CURRENT population’s own size (the standard "mating pool" convention: a full-size parent pool). The resulting

'parents' list is passed to 'self$vary(parents, ctx)' UNCHANGED, and 'vary()''s return value is returned UNCHANGED as this generation's offspring.

Super class

`Algorithm` -> `PopulationAlgorithm`

Methods

Public methods:

- `PopulationAlgorithm$select()`
- `PopulationAlgorithm$vary()`
- `PopulationAlgorithm$generate()`
- `PopulationAlgorithm$clone()`

`PopulationAlgorithm$select()`:

Usage:

```
PopulationAlgorithm$select(pop, k, ctx)
```

`PopulationAlgorithm$vary()`:

Usage:

```
PopulationAlgorithm$vary(parents, ctx)
```

`PopulationAlgorithm$generate()`:

Usage:

```
PopulationAlgorithm$generate(pop, ctx)
```

`PopulationAlgorithm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PopulationAlgorithm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`print.sz_block` *Prints an 'sz_block' object (any of the five block kinds).*

Description

Prints an 'sz_block' object (any of the five block kinds).

Usage

```
## S3 method for class 'sz_block'
print(x, ...)
```

Arguments

x	An ‘sz_block‘ object (from [sz_float()], [sz_int()], [sz_categorical()], [sz_binary()], or [sz_permutation()]).
...	Ignored.

Value

‘x‘, invisibly.

---

print.sz_result *Prints an ‘sz_result‘ object.*

Description

Prints an ‘sz_result‘ object.

Usage

```
## S3 method for class 'sz_result'
print(x, ...)
```

Arguments

x	An ‘sz_result‘ object (from [.sz_wrap_result]/[Algorithm]’s own ‘run()‘).
...	Ignored.

Value

‘x‘, invisibly.

---

print.sz_space *Prints an ‘sz_space‘ object.*

Description

Prints an ‘sz_space‘ object.

Usage

```
## S3 method for class 'sz_space'
print(x, ...)
```

Arguments

x	An ‘sz_space‘ object (from [sz_space()]).
...	Ignored.

Value

‘x‘, invisibly.

`Problem` *Subclassable R6 base for a search-space problem.*

Description

Mirrors ‘sezgi.Problem‘ (‘py-sezgi/python/sezgi/problem.py‘) exactly: subclass and override ‘evaluate(x)‘ and ‘space()‘ (both required – the defaults below ‘stop()‘ with a "not implemented" message); ‘optimum()‘ is an optional override, default ‘NULL‘.

Details

Genotype -> R conversion table for ‘evaluate(x)‘’s own ‘x‘ argument (PINNED; mirrors M4-1’s pinned block -> Python table exactly, see ‘py-sezgi/python/sezgi/problem.py‘’s own module doc):

Float block a numeric (double) vector

Int block an integer vector

Categorical block an integer vector of category INDICES ‘0..k‘, not labels

Binary block a logical vector

Permutation block an integer vector, 0-based

A single-block space’s ‘x‘ is that one block’s own converted value, passed BARE. A multi-block space’s ‘x‘ is an (unnamed) R ‘list‘ of per-block converted values, in ‘space()‘’s own block order.

‘batch_evaluate(xs)‘’s own ‘xs‘ is a ‘list‘ of ‘x‘ values (one per individual, each in the SAME per-‘x‘ shape as ‘evaluate(x)‘’s own ‘x‘ above), matching py-sezgi’s ‘sezgi.Problem.batch_evaluate‘ input shape exactly (‘py-sezgi/python/sezgi/problem.py‘) – the Python side is the authority for this shape, not an independent R design.

Methods:

‘evaluate(x)‘ ‘x‘ -> a numeric scalar, the fitness/objective value at ‘x‘. Subclasses MUST override this. See the class doc’s conversion table above for ‘x‘’s exact shape. Default: ‘stop()‘s with "not implemented".

‘space()‘ Declares the search space. Subclasses MUST override this to return an ‘sz_space(...)‘ object. Default: ‘stop()‘s with "not implemented".

‘optimum()‘ The problem’s known optimum (a numeric scalar), or ‘NULL‘ (the default) if it has none. Optional override.

‘batch_evaluate(xs)‘ ‘xs‘ -> a numeric vector, one entry per ‘x‘ in ‘xs‘, in the SAME order (default: loop ‘self$evaluate‘, mirroring ‘sezgi.Problem.batch_evaluate‘’s own default ‘[self.evaluate(x) for x in xs]‘ exactly). Override for a vectorized objective; called ONCE PER GENERATION with the WHOLE population as ‘xs‘ (see ‘.sz_make_evaluate_shim‘’s own doc).

Methods

Public methods:

- Problem$evaluate()
- Problem$space()
- Problem$optimum()
- Problem$batch_evaluate()
- Problem$clone()

`Problem$evaluate()`:

Usage:

```
Problem$evaluate(x)
```

`Problem$space()`:

Usage:

```
Problem$space()
```

`Problem$optimum()`:

Usage:

```
Problem$optimum()
```

`Problem$batch_evaluate()`:

Usage:

```
Problem$batch_evaluate(xs)
```

`Problem$clone()`: The objects of this class are cloneable with this method.

Usage:

```
Problem$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
Sphere <- R6::R6Class("Sphere", inherit = Problem, public = list(
  space = function() sz_space(sz_float(-5, 5, 2)),
  evaluate = function(x) sum(x^2),
  optimum = function() 0
))
prob <- Sphere$new()
prob$evaluate(c(1, 2))
```

sezgi_version *Version of the underlying sezgi Rust core.*

Description

Version of the underlying sezgi Rust core.

Usage

```
sezgi_version()
```

Value

A character scalar with the crate version string.

SzRng *An owned per-call RNG handle exposed to an R-authored ‘generate‘/ ‘initialize‘ callback as the ‘rng‘ argument – an external-pointer-backed ‘#[savvy]‘ object (‘SzRng$new‘/‘SzRng$from_master‘/method closures are all auto-generated by ‘savvy-cli‘, the SAME mechanism ‘EvalSession‘ already uses, ‘session.rs‘’s own module doc). Wraps a CLONE of the stage’s live ‘RngStream‘ – mirrors ‘PyRng‘ (‘py-sezgi/src/lib.rs‘) exactly: every method here delegates 1:1 to ‘RngStream‘’s own ‘pub‘ API (‘crates/core/src/rng.rs:16-59‘), no new RNG logic.*

Description

Clone-out / write-back / null-out protocol (research doc §C5): [‘RGenerator::generate‘]/[‘RInitializer::initialize‘] clone ‘*ctx.rng‘ into a fresh ‘SzRng‘ BEFORE the call, ‘into_external_pointer()‘ it, and hand the raw pointer to R as one argument. After the call returns, the mutated state is read back via [‘take_external_pointer_value‘] – which hands back OWNERSHIP of the (possibly-advanced) ‘SzRng‘ AND nulls the R-side pointer (‘R_ClearExternalPtr‘) in the SAME call. This is a DELIBERATE IMPROVEMENT over an earlier stale-handle fix, which was docstring-only ("valid only for the duration of one ‘generate()‘ call"): here, a callback that STORES ‘ctx$rng‘ in its own enclosure across iterations gets a clear ‘savvy::Error::InvalidPointer‘ ("This external pointer is already consumed or deleted") on any later use, rather than silently reading stale state – a hazard turned into an honest error, not just a documented one.

Usage

```
SzRng
```

`sz_algorithm` *Declares a pure-R metaheuristic algorithm to run over an 'EvalSession'.*

Description

Base R only (no R6/S4): the result is a plain classed 'list' holding the two closures and a name – there is no method dispatch, subclassing, or mutable state of its own; all run state lives in the 'ctx' environment 'sz_algo_solve()' builds fresh for each call. Mirrors py-sezgi's 'sezgi.Algorithm' ABC (subclass, implement 'setup()'/'step()'), expressed as two plain closures instead of two abstract methods.

Usage

```
sz_algorithm(setup, step, name = "custom")
```

Arguments

`setup`	A function 'function(ctx)', called once at the start of a run.
`step`	A function 'function(ctx)', called repeatedly while budget remains. Each call MUST evaluate at least one point (via 'ctx$evaluate()') or let 'ctx$evaluate()' raise 'sz_budget_exhausted' – a 'step()' that returns having consumed no budget is a driver error (see 'sz_algo_solve()').
`name`	Character scalar, the algorithm's label – recorded as the 'algo' field of 'sz_algo_solve()''s result and (when the session has IOH logging enabled) the IOH archive's algorithm name. Default '"custom"'.

Value

A classed 'list' ('"sz_algorithm"') with elements 'setup', 'step', 'name'.

`sz_algo_solve` *Drives an 'sz_algorithm' over an 'EvalSession' to exhaustion.*

Description

Semantics MIRROR py-sezgi's 'Algorithm.solve()' exactly ('py-sezgi/python/sezgi/algo.py'):

- 'set.seed(seed)' is called ONCE, up front – R's global RNG IS the 'ctx' RNG for the whole run (see '.sz_algo_context()''s doc); 'algo$setup'/'algo$step' calling 'ctx$random_point()' (or drawing from R's RNG directly, e.g. 'runif()'/'sample()') are both driven by this one seeded stream, so a run is fully reproducible from 'seed' alone.
- The driver calls 'algo$setup(ctx)' once, then 'algo$step(ctx)' repeatedly while 'ctx$remaining() > 0'. A 'step()' call that leaves 'ctx$evals_used()' UNCHANGED is a driver error ('stop()' with a message containing '"consumed no budget"') – a step must evaluate at least one point or let 'ctx$evaluate()' raise 'sz_budget_exhausted'.

- ‘ctx$evaluate(points)‘ signals the condition class ‘"sz_budget_exhausted"‘ when ‘points‘ would exceed the remaining budget – checked BEFORE the session is touched, so a rejected batch spends nothing. ‘sz_algo_solve()‘’s own ‘tryCatch‘ catches ONLY this condition class (wrapping both ‘algo$setup()‘ and the ‘step()‘ loop), ending the run cleanly; any OTHER error ‘setup()‘/‘step()‘ raises (including the "consumed no budget" guard above) propagates out of ‘sz_algo_solve()‘ uncaught.
- A run that ends having evaluated NOTHING at all (‘ctx$best()‘ is ‘NULL‘) errors.
- ‘session$finish()‘ is GUARANTEED to run EXACTLY ONCE, on every exit path – success, the "consumed no budget"/"no evaluations" guards, or any error ‘setup()‘/‘step()‘ itself raises – via ‘on.exit(..., add = TRUE)‘ registered before the run starts (base R’s ‘finally‘ equivalent). This mirrors py-sezgi’s own ‘try‘/ ‘finally‘ structure, which a Task 4 (M3-4) final-review finding added specifically so a run that dies mid-flight does not silently lose its IOH archive; applied here from the start rather than rediscovered later.

Usage

```
sz_algo_solve(algo, session, seed)
```

Arguments

`algo`	An ‘sz_algorithm‘ (from ‘sz_algorithm()‘).
`session`	An ‘EvalSession‘ object (from ‘sz_eval_session()‘ or a sibling constructor) – NOT yet finished; ‘sz_algo_solve()‘ finishes it.
`seed`	Integer/numeric scalar, the master RNG seed. Passed straight to ‘set.seed()‘; also recorded verbatim as the result’s ‘seed‘ field (purely documentation of the seed used – it plays no other role once ‘set.seed()‘ has run).

Value

A named ‘list‘ with fields ‘algo‘ (character, ‘algo$name‘), ‘seed‘, ‘budget‘ (‘session$budget()‘), ‘evals_used‘, ‘best_x‘ (numeric vector), ‘best_f‘ (numeric scalar), ‘f_opt‘ (numeric scalar, or ‘NULL‘ if the problem has no known optimum – e.g. an f0 session), ‘gap‘ (‘best_f - f_opt‘, or ‘NULL‘ iff ‘f_opt‘ is ‘NULL‘) – mirrors py-sezgi’s ‘SolveResult‘ field names exactly.

`sz_as_problem` *Accepts a ‘Problem‘ subclass instance.*

Description

Mirrors py-sezgi’s ‘as_native_problem()‘ (‘py-sezgi/python/sezgi/problem.py‘) in spirit – unlike py-sezgi, r-sezgi builds no separate "native problem handle" type to convert to (there is no ‘_sezgi.Problem‘ counterpart), so a ‘Problem‘ subclass instance already IS what this task’s Rust bridge needs; this function’s job is purely to give a friendly, dedicated error for the "not a Problem" case, at the point of use rather than deep inside the solve machinery.

Usage

```
sz_as_problem(obj)
```

Arguments

obj	An object to check.

Value

‘obj‘, unchanged, if it inherits ‘"Problem"‘.

---

sz_bayesian_plackett_luce

Bayesian Plackett-Luce posterior via Gibbs sampling.

Description

Mirrors py-sezgi’s ‘stats.bayesian_plackett_luce()‘. Caron & Doucet (2012) latent exponential-race Gibbs augmentation under independent ‘Gamma(1, 1)‘ priors; see ‘crates/stats/src/bayesian.rs‘ for the full PINNED sampler and determinism guarantees. The same ‘(rankings, samples, burn_in, seed)‘ always produces a bit-identical result – R calls the exact same seeded Rust core as Python and Rust.

Usage

```
sz_bayesian_plackett_luce(rankings, samples = 2000, burn_in = 500, seed = 1)
```

Arguments

rankings	A ‘list‘ of integer (or integer-valued numeric) vectors, each a full ranking of the same ‘k‘ items as **1-based item ids** (same convention as ‘sz_stats_plackett_luce()‘; converted to 0-based indices via the SAME converter). Best (rank 1) first.
samples	Number of post-burn-in Gibbs iterations to record. Default 2000.
burn_in	Number of initial Gibbs iterations to discard. Default 500.
seed	Master RNG seed. Default 1.

Value

A named list with ‘mean_worths‘, ‘ci_low‘, ‘ci_high‘, ‘p_best‘, ‘samples‘.

`sz_bias_central` *Central-bias scan: run an algorithm spec on paired centered/shifted BBOB conditions and test whether its performance gap differs between them (Kudela's center-bias-exploitation method). See 'crates/bias/src/central.rs' for the full method provenance.*

Description

Central-bias scan: run an algorithm spec on paired centered/shifted BBOB conditions and test whether its performance gap differs between them (Kudela's center-bias-exploitation method). See 'crates/bias/src/central.rs' for the full method provenance.

Usage

```
sz_bias_central(
  spec_json,
  dim,
  budget,
  fids = NULL,
  instances_shifted = NULL,
  runs_per = 20,
  seed = 0
)
```

Arguments

`spec_json`	Algorithm spec as JSON.
`dim`	Problem dimension (BBOB requires 'dim >= 2').
`budget`	Per-run evaluation budget (overrides the spec's own budget).
`fids`	Optional numeric vector of BBOB function ids. 'NULL' (default) uses the verified defaults 'c(1, 4, 13)'. Every entry must be translation-invariant – fids 5, 6, 20, 24 are rejected with an error (their 'x_opt' participates directly in the objective formula, not just as a coordinate shift).
`instances_shifted`	Optional numeric vector of BBOB instance numbers. 'NULL' (default) uses the verified defaults 'c(1, 2)'.
`runs_per`	Independent runs per '(fid, instance)' pair. Default 20 – Kudela's own verified run count.
`seed`	Master RNG seed, shared by both conditions (a paired design; see 'crates/bias/src/central.rs''s "Pairing rationale").

Value

A named list with 'gap_centered', 'gap_shifted' (numeric vectors, same length/order – index 'i' in both is one matched pair), 'wilcoxon' (named list: 'w_statistic', 'z', 'p_value', 'n_effective', 'method'), 'effect' (Cliff's delta, 'gap_shifted' vs 'gap_centered'), 'verdict', 'detail' – mirrors pysezgi's 'sezgi.bias.central()' dict keys exactly.

sz_bias_report	*One-call bias report: runs both the structural and central bias scans on one algorithm spec and assembles a single report with a ready-to-paste LaTeX summary table. See ‘crates/bias/src/report.rs‘ for the full assembly/LaTeX-rendering details.*

Description

T6 (the Rajwar-Deep signature-bias test) is DEFERRED in the underlying ‘sezgi-bias‘ crate itself – the method could not be pinned from accessible sources (no reachable source fully specifies its statistical procedure). There is no ‘sz_bias_signature()‘; the returned list’s ‘signature‘ element is mapped through properly (not hardcoded), so it is ‘NULL‘ today only because the crate’s own result is always ‘NULL‘ today – see below – and ‘latex_summary‘ renders an honest "not run" row for it instead of a fabricated result.

Usage

```
sz_bias_report(
  spec_json,
  dim,
  budget,
  seed = 0,
  structural_runs = NULL,
  central_fids = NULL,
  central_instances = NULL,
  central_runs_per = NULL
)
```

Arguments

spec_json	Algorithm spec as JSON.
dim	Shared dimensionality for both scans (BBOB requires ‘dim >= 2‘).
budget	Shared per-run evaluation budget for both scans.
seed	Shared master RNG seed for both scans.
structural_runs	Optional; ‘NULL‘ (default) uses the structural scan’s own verified default (30).
central_fids	Optional numeric vector; ‘NULL‘ (default) uses ‘c(1, 4, 13)‘.
central_instances	Optional numeric vector; ‘NULL‘ (default) uses ‘c(1, 2)‘.
central_runs_per	Optional; ‘NULL‘ (default) uses 20.

Value

A named list with ‘structural‘ (same shape as ‘sz_bias_structural()‘’s return), ‘central‘ (same shape as ‘sz_bias_central()‘’s return), ‘signature‘ (‘NULL‘ today, see above; when non-‘NULL‘, a two-element list ‘list(verdict = ..., detail = ...)‘, same shape as ‘structural‘’s/‘central‘’s own ‘verdict‘/‘detail‘), ‘latex_summary‘ (character scalar, never contains the literal ‘"NaN"‘), ‘plot_data‘ (named list: ‘final_positions‘, ‘gap_centered‘, ‘gap_shifted‘ – the same raw vectors already inside ‘structural‘/‘central‘) – mirrors py-sezgi’s ‘sezgi.bias.report()‘ dict keys exactly.

`sz_bias_structural` *Structural-bias scan: run an algorithm spec repeatedly on the f0 random-function null problem and test its final positions for departure from uniformity (BIAS-toolbox method; Kononova et al. 2015 / Vermetten et al. 2022). See ‘crates/bias/src/structural.rs‘ for the full method provenance.*

Description

Structural-bias scan: run an algorithm spec repeatedly on the f0 random-function null problem and test its final positions for departure from uniformity (BIAS-toolbox method; Kononova et al. 2015 / Vermetten et al. 2022). See ‘crates/bias/src/structural.rs‘ for the full method provenance.

Usage

```
sz_bias_structural(spec_json, dim, budget, runs = 30, seed = 0)
```

Arguments

`spec_json`	Algorithm spec as JSON (e.g. from ‘sz_preset_random_search()‘).
`dim`	f0’s domain dimensionality (>= 1).
`budget`	Per-run evaluation budget (overrides the spec’s own budget).
`runs`	Number of independent runs. Default 30 – the BIAS toolbox’s own verified minimum/default (‘sezgi_bias::structural::DEFAULT_RUNS‘).
`seed`	Master RNG seed, mixed into both f0’s own RNG and the engine’s ‘master_seed‘.

Value

A named list with ‘per_dim_ks‘ (list of ‘list(d=, p_value=, n=)‘, length ‘dim‘), ‘per_dim_ad‘ (list of ‘list(a2=, p_value=, n=)‘, same length/order), ‘holm_rejections_ks‘, ‘holm_rejections_ad‘, ‘verdict‘ (‘"no_evidence"‘ or ‘"evidence"‘), ‘detail‘ (‘NULL‘ iff ‘verdict == "no_evidence"‘, else a character scalar), ‘final_positions‘ (list of ‘runs‘ numeric vectors, each length ‘dim‘) – mirrors py-sezgi’s ‘sezgi.bias.structural()‘ dict keys exactly.

sz_bias_structural_positions
: *Statistics-only structural-bias scan over externally-collected final positions – the bias bridge for algorithms authored OUTSIDE this package's own spec/engine (M3-5 Task 6), e.g. a pure-R* `sz_algorithm` *driven by* `sz_algo_solve` *over* `sz_eval_session_f0`*, one run at a time. Runs the SAME KS/AD/ Holm battery as* `sz_bias_structural` *over caller-supplied 'final_positions' instead of driving an algorithm spec through the engine itself – see* `crates/bias/src/structural.rs`*'s 'scan_from_positions' for the full method provenance. Mirrors py-sezgi's 'sezgi.bias.structural_positions()' 1:1.*

Description

Statistics-only structural-bias scan over externally-collected final positions – the bias bridge for algorithms authored OUTSIDE this package's own spec/engine (M3-5 Task 6), e.g. a pure-R `sz_algorithm` driven by `sz_algo_solve` over `sz_eval_session_f0`, one run at a time. Runs the SAME KS/AD/ Holm battery as `sz_bias_structural` over caller-supplied 'final_positions' instead of driving an algorithm spec through the engine itself – see `crates/bias/src/structural.rs`'s 'scan_from_positions' for the full method provenance. Mirrors py-sezgi's 'sezgi.bias.structural_positions()' 1:1.

Usage

```
sz_bias_structural_positions(final_positions)
```

Arguments

final_positions
: A numeric matrix (rows = independent runs' final positions, columns = dimension) or a `list` of numeric vectors, one per run – the same two shapes `EvalSession$evaluate()` itself accepts. Must have at least 5 rows/elements (the BIAS toolbox's own verified minimum run count), all the same length (dimension); a shorter/longer row (ragged input) is rejected.

Value

Same named-list shape as `sz_bias_structural`'s return: 'per_dim_ks', 'per_dim_ad', 'holm_rejections_ks', 'holm_rejections_ad', 'verdict' ('"no_evidence"' or '"evidence"'), 'detail' ('NULL' iff 'verdict == "no_evidence"'), 'final_positions'.

sz_binary *A binary block: ‘n‘ bits.*

Description

Mirrors ‘sezgi.Binary‘ (‘py-sezgi/python/sezgi/spaces.py‘) exactly – see [sz_float()]’s doc for the no-construction-time-validation note.

Usage

```
sz_binary(n)
```

Arguments

n	Numeric/integer scalar, the number of bits.

Value

An object of class ‘c("sz_block_binary", "sz_block")‘ with element ‘n‘.

sz_builtin_bbob *A built-in-problem descriptor for [Algorithm]’s ‘run()‘.*

Description

r-sezgi has no general "native problem handle" type the way py-sezgi’s ‘as_native_problem()‘ does (every existing ‘sz_solve_*‘/‘sz_eval_session*‘ binding takes ‘fid‘/‘dim‘/‘instance‘ as separate scalar arguments, never a wrapped object) – this is a MINIMAL, purpose-built descriptor letting ‘Algorithm$run()‘’s single ‘problem‘ argument select T3’s ‘sz_solve_r_generator_bbob()‘ entry point (the "built-in form" of a ‘run()‘ call, mirroring py-sezgi’s ‘sezgi.bbob(fid, dim, instance)‘ handles reaching ‘Algorithm.run()‘ the same way) instead of the ‘Problem‘-subclass path. Deliberately narrow: it exists ONLY to route ‘Algorithm$run()‘, not as a general native-problem abstraction (a richer type, if ever needed by a future task, is out of this task’s scope – see the task-4 report’s concerns for T5).

Usage

```
sz_builtin_bbob(fid, dim, instance = 1)
```

Arguments

fid	BBOB function id (‘1..=24‘).
dim	Problem dimension (>= 1).
instance	BBOB instance id (>= 1). Default ‘1‘.

Value

An object of class ‘"sz_builtin_bbob"‘ with elements ‘fid‘, ‘dim‘, ‘instance‘.

`sz_categorical` *A categorical block: ‘n‘ genes, each a category index in ‘0..k‘.*

Description

Mirrors ‘sezgi.Categorical‘ (‘py-sezgi/python/sezgi/spaces.py‘) exactly – see [sz_float()]’s doc for the no-construction-time-validation note.

Usage

```
sz_categorical(k, n)
```

Arguments

k	Numeric/integer scalar, the number of categories.
n	Numeric/integer scalar, the number of genes. Required – ‘spaces.py‘’s ‘Categorical.n‘ has no default either.

Value

An object of class ‘c("sz_block_categorical", "sz_block")‘ with elements ‘k‘, ‘n‘.

`sz_cec2014_evaluate` *Direct, one-shot evaluation of a CEC 2014 (Liang, Qu & Suganthan 2013) function at ‘x‘, bypassing ‘sz_solve_bbob‘-style budget/engine machinery entirely – binds [‘Cec2014::new‘] + [‘Cec2014::evaluate_batch‘] exactly. Mirrors ‘sz_cec2022_evaluate‘ (M3-6 Task 10 – see py-sezgi’s ‘sezgi.problems.cec2014_evaluate‘ for the identical binding on the Python side). See [‘Cec2014::new‘]’s own doc for the exact ‘fid‘/‘dim‘ domain.*

Description

Direct, one-shot evaluation of a CEC 2014 (Liang, Qu & Suganthan 2013) function at ‘x‘, bypassing ‘sz_solve_bbob‘-style budget/engine machinery entirely – binds [‘Cec2014::new‘] + [‘Cec2014::evaluate_batch‘] exactly. Mirrors ‘sz_cec2022_evaluate‘ (M3-6 Task 10 – see py-sezgi’s ‘sezgi.problems.cec2014_evaluate‘ for the identical binding on the Python side). See [‘Cec2014::new‘]’s own doc for the exact ‘fid‘/‘dim‘ domain.

Usage

```
sz_cec2014_evaluate(fid, dim, x)
```

Arguments

fid	CEC 2014 function id, ‘1..=30‘ (double, cast to ‘u32‘).
dim	Problem dimension, one of ‘10‘, ‘30‘ (double, cast to ‘usize‘).
x	A numeric vector of exactly ‘dim‘ coordinates.

Value

A numeric scalar.

Errors A savvy error for ‘fid‘ outside ‘1..=30‘, ‘dim‘ outside ‘{10,30}‘, or ‘length(x) != dim‘.

---

sz_cec2014_f_star

The report's pinned ‘F_i‘ bias for a CEC 2014 function – binds [‘Cec2014::f_star‘] (‘F_i* = 100*fid‘). Does not depend on ‘dim‘, so an internal probe ‘dim = 10‘ is used purely to validate ‘fid‘.*

Description

The report's pinned ‘F_i*‘ bias for a CEC 2014 function – binds [‘Cec2014::f_star‘] (‘F_i* = 100*fid‘). Does not depend on ‘dim‘, so an internal probe ‘dim = 10‘ is used purely to validate ‘fid‘.

Usage

```
sz_cec2014_f_star(fid)
```

Arguments

fid	CEC 2014 function id, ‘1..=30‘ (double, cast to ‘u32‘).

Value

A numeric scalar.

Errors A savvy error if ‘fid‘ is outside ‘1..=30‘.

sz_cec2017_evaluate	*Direct, one-shot evaluation of a CEC 2017 (Awad, Ali, Liang, Qu & Suganthan 2016) function at ‘x‘, bypassing ‘sz_solve_bbob‘-style budget/engine machinery entirely – binds [‘Cec2017::new‘] + [‘Cec2017::evaluate_batch‘] exactly. Mirrors ‘sz_cec2014_evaluate‘ (M3-6 Task 10 – see py-sezgi’s ‘sezgi.problems.cec2017_evaluate‘ for the identical binding on the Python side). See [‘Cec2017::new‘]’s own doc for the exact ‘fid‘/‘dim‘ domain.*

Description

Direct, one-shot evaluation of a CEC 2017 (Awad, Ali, Liang, Qu & Suganthan 2016) function at ‘x‘, bypassing ‘sz_solve_bbob‘-style budget/engine machinery entirely – binds [‘Cec2017::new‘] + [‘Cec2017::evaluate_batch‘] exactly. Mirrors ‘sz_cec2014_evaluate‘ (M3-6 Task 10 – see py-sezgi’s ‘sezgi.problems.cec2017_evaluate‘ for the identical binding on the Python side). See [‘Cec2017::new‘]’s own doc for the exact ‘fid‘/‘dim‘ domain.

Usage

```
sz_cec2017_evaluate(fid, dim, x)
```

Arguments

fid	CEC 2017 function id, ‘1‘ or ‘3..=30‘ (double, cast to ‘u32‘); ‘fid = 2‘ ("Sum of Different Powers") was officially withdrawn from the suite.
dim	Problem dimension, one of ‘10‘, ‘30‘ (double, cast to ‘usize‘).
x	A numeric vector of exactly ‘dim‘ coordinates.

Value

A numeric scalar.

Errors A savvy error for ‘fid‘ outside ‘{1} union {3..=30}‘, ‘dim‘ outside ‘{10,30}‘, or ‘length(x) != dim‘. ‘fid = 2‘ raises a dedicated error – the Rust [‘sezgi_problems::Cec2017Error::Withdrawn‘] message is surfaced VERBATIM, distinct from an ordinary out-of-range ‘fid‘.

sz_cec2017_f_star	*The report’s pinned ‘F_i*‘ bias for a CEC 2017 function – binds [‘Cec2017::f_star‘] (‘F_i* = 100*fid‘, the fid-gapped C dispatch bias, not the report’s contiguous renumbering – see ‘Cec2017::new‘’s own module doc). Does not depend on ‘dim‘, so an internal probe ‘dim = 10‘ is used purely to validate ‘fid‘.*

Description

The report's pinned 'F_i*' bias for a CEC 2017 function – binds ['Cec2017::f_star'] ('F_i* = 100*fid', the fid-gapped C dispatch bias, not the report's contiguous renumbering – see 'Cec2017::new''s own module doc). Does not depend on 'dim', so an internal probe 'dim = 10' is used purely to validate 'fid'.

Usage

```
sz_cec2017_f_star(fid)
```

Arguments

fid	CEC 2017 function id, '1' or '3..=30' (double, cast to 'u32').

Value

A numeric scalar.

Errors A savvy error if 'fid' is outside '{1} union {3..=30}' ('fid = 2' included, via the ['sezgi_problems::Cec2017Error::Wit
message).

`sz_cec2022_evaluate` *Direct, one-shot evaluation of a CEC 2022 function at 'x', bypassing 'sz_solve_bbob'-style budget/engine machinery entirely – binds ['Cec2022::new'] + ['Cec2022::evaluate_batch'] exactly. See 'Cec2022::new''s own doc for the exact 'fid'/'dim' domain.*

Description

Direct, one-shot evaluation of a CEC 2022 function at 'x', bypassing 'sz_solve_bbob'-style budget/engine machinery entirely – binds ['Cec2022::new'] + ['Cec2022::evaluate_batch'] exactly. See 'Cec2022::new''s own doc for the exact 'fid'/'dim' domain.

Usage

```
sz_cec2022_evaluate(fid, dim, x)
```

Arguments

fid	CEC 2022 function id, '1..=12' (double, cast to 'u32').
dim	Problem dimension, one of '2', '10', '20' (double, cast to 'usize'); 'dim = 2' is additionally rejected for a hybrid function ('fid' 6-8).
x	A numeric vector of exactly 'dim' coordinates.

Value

A numeric scalar.

Errors A savvy error for 'fid' outside '1..=12', 'dim' outside '{2,10,20}', 'dim = 2' for a hybrid function, or 'length(x) != dim'.

`sz_cec2022_f_star` *The report's pinned 'F_i*' bias for a CEC 2022 function – binds ['Cec2022::f_star'] (module doc section 1.2's table). 'f_star' does not depend on 'dim', so an internal probe 'dim = 10' is used purely to validate 'fid' (every 'fid' in '1..=12' accepts 'dim = 10', hybrids included).*

Description

The report's pinned 'F_i*' bias for a CEC 2022 function – binds ['Cec2022::f_star'] (module doc section 1.2's table). 'f_star' does not depend on 'dim', so an internal probe 'dim = 10' is used purely to validate 'fid' (every 'fid' in '1..=12' accepts 'dim = 10', hybrids included).

Usage

```
sz_cec2022_f_star(fid)
```

Arguments

`fid`	CEC 2022 function id, '1..=12' (double, cast to 'u32').

Value

A numeric scalar.

Errors A savvy error if 'fid' is outside '1..=12'.

`sz_coco_export` *Exports the IOH archive at 'log_root' as a COCO/BBOB "old format" archive rooted at 'out_dir' – see 'sezgi_bench::coco_export'. Returns the list of written file paths (as strings), sorted for determinism.*

Description

BBOB-only: COCO's "old format" IS the BBOB archive format and has no CEC counterpart, so this errors (naming the offending suite) if 'log_root' holds any non-BBOB scenario, rather than silently merging it into a 'bbob'-labeled archive. A mixed BBOB+CEC tree must be filtered to its BBOB records before exporting.

Usage

```
sz_coco_export(log_root, out_dir)
```

Arguments

`log_root`	Path to the IOH archive directory (as passed to 'sz_run_experiment(..., log_dir = ...)').
`out_dir`	Directory to write the COCO/BBOB archive under.

Value

A character vector of written file paths, sorted.

`sz_ecdf` *ECDF/anytime curve(s) over an on-disk IOH archive.*

Description

Mirrors py-sezgi's 'ecdf()'. See 'crates/bench/src/anytime.rs' for the full PINNED definitions ('default_targets', the ECDF denominator convention, 'hit_time').

Usage

```
sz_ecdf(log_root, targets = NULL, per_algo = TRUE)
```

Arguments

`log_root`	Path to the IOH archive directory (as passed to 'sz_run_experiment(..., log_dir = ...)').
`targets`	Optional numeric vector of precision targets; 'NULL' (default) uses the COCO-convention 51-value default target set.
`per_algo`	'TRUE' (default) returns a named list, one 'list(evals=, proportion=)' entry per distinct algo in the archive, named by algo (first-appearance order); 'FALSE' returns a single pooled 'list(evals=, proportion=)' over every scenario. Grouping (in both modes) is by algo only, not '(algo, suite)': a mixed-suite tree pools both suites' runs into one algo's curve – read a per-suite tree or filter by suite first for a suite-specific curve.

Value

A named list (see 'per_algo').

`sz_eval_session` *Start an ask/tell evaluation session over a BBOB problem.*

Description

The session is the sole keeper of evaluation, tamper-proof counting, best-tracking and (optionally) IOH logging, driven entirely by candidate points an EXTERNAL algorithm supplies – the spec's engine-inside-out promise. Mirrors py-sezgi's 'sezgi.EvalSession'.

Usage

```
sz_eval_session(
  fid,
  dim,
  instance,
  budget,
  log_dir = NULL,
  algo_name = "custom",
  seed = 0
)
```

Arguments

fid	BBOB function id (>= 1).
dim	Problem dimension (>= 1).
instance	BBOB instance id (>= 1).
budget	Evaluation budget (non-negative whole number). A fractional value is REJECTED (M2d-3: 'f64_to_u64' now errors with "expected a whole number" instead of truncating toward zero), matching py-sezgi's 'budget: u64' parameter, whose type makes pyo3 reject a fractional Python value outright at the FFI boundary.
log_dir	Optional directory: when given, IOH-profiler logging is wired up in the constructor, before any evaluation is possible (constructor-only, like 'sezgi.EvalSession'). Default 'NULL' (no logging).
algo_name	Algorithm label recorded in the IOH archive (only meaningful when 'log_dir' is given). Default '"custom"'.
seed	Caller-declared reproducibility label recorded in the IOH archive meta only – the session itself never draws random numbers, so this is purely documentation of the CALLER's RNG seed. Default '0'.

Details

The returned object is a reference (external-pointer) handle, not a value: assigning it to another variable (e.g. `s2 <- s`) does NOT copy the session – both names alias the SAME mutable session, so a `$evaluate()` through either one advances the same counter and best. This is standard R external-pointer/environment behavior (the object literally IS an `environment`), not something specific to this package.

The returned object exposes:

- `$evaluate(x)` – batch-evaluates 'x' (a numeric matrix with rows = points and 'dim' columns, or a 'list' of 'dim'-length numeric vectors) and returns a numeric vector of 'f' values, one per point. All-or-nothing: on any error (dimension mismatch, a non-finite coordinate, or a batch that would cross the remaining budget) nothing is counted.
- `$evals_used()` – number of evaluations counted so far.
- `$budget()` – the session's total evaluation budget.

- `$best()` – `list(x = <numeric vector>, f = <numeric scalar>)` for the best evaluation seen so far, or `NULL` if nothing has been evaluated yet.
- `$f_opt()` – the problem's known optimum value, or `NULL` if it has none (e.g. an f0 session – see `sz_eval_session_f0`).
- `$dim()` – the search space's dimensionality.
- `$bounds()` – a length-2 numeric vector `c(lo, hi)`, the uniform bounds of this session's continuous space (errors for a non-uniform/non-float space; unreachable through any constructor this package exposes today).
- `$finish()` – flushes the IOH log (if enabled) and marks the session finished; every method call afterward, including a second `$finish()`, raises an error containing `"session finished"`.

`$dim()`, `$bounds()`, and every method above except `$f_opt()` behave identically regardless of which constructor built the session (`sz_eval_session()`, `sz_eval_session_cec2022`, or `sz_eval_session_f0`).

Value

An 'EvalSession' object (see above).

`sz_eval_session_cec2014` *Start an ask/tell evaluation session over a CEC 2014 problem.*

Description

The generic-session counterpart to `sz_eval_session` (which is BBOB-only), mirroring `sz_eval_session_cec2022` exactly: builds a session over `sz_cec2014_evaluate`'s same `Cec2014` problem, with IOH logging allowed. Session metadata (`suite = "sezgi-cec2014"`, `name = "cec2014-f<fid>"`, `instance = 1`, `f_opt` = the function's pinned `F_i* = 100*fid` bias) matches py-sezgi's `EvalSession.for_problem` exactly for a CEC 2014 problem handle, so an IOH record produced by an R session and one produced by an equivalent Python session reconstruct to the identical `(suite, fid,name, instance)` key and `f_opt`.

Usage

```
sz_eval_session_cec2014(
  fid,
  dim,
  budget,
  log_dir = NULL,
  algo_name = "custom",
  seed = 0
)
```

Arguments

fid	CEC 2014 function id (1..=30).
dim	Problem dimension, one of 10, 30.
budget	Evaluation budget (non-negative whole number).
log_dir	Optional directory: when given, IOH-profiler logging is wired up in the constructor, before any evaluation is possible. Default NULL (no logging).
algo_name	Algorithm label recorded in the IOH archive (only meaningful when log_dir is given). Default "custom".
seed	Caller-declared reproducibility label recorded in the IOH archive meta only. Default 0.

Details

See sz_eval_session for the full list of methods the returned object exposes (identical here, including $dim() / $bounds()).

Value

An 'EvalSession' object (see sz_eval_session).

---

sz_eval_session_cec2017 *Start an ask/tell evaluation session over a CEC 2017 problem.*

Description

The generic-session counterpart to sz_eval_session (which is BBOB-only), mirroring sz_eval_session_cec2014 exactly: builds a session over sz_cec2017_evaluate's same Cec2017 problem, with IOH logging allowed. Session metadata (suite = "sezgi-cec2017", name = "cec2017-f<fid>", instance = 1, f_opt = the function's pinned F_i* = 100*fid bias) matches py-sezgi's EvalSession.for_problem exactly for a CEC 2017 problem handle.

Usage

```
sz_eval_session_cec2017(
  fid,
  dim,
  budget,
  log_dir = NULL,
  algo_name = "custom",
  seed = 0
)
```

Arguments

fid	CEC 2017 function id (1 or 3..=30).
dim	Problem dimension, one of 10, 30.
budget	Evaluation budget (non-negative whole number).
log_dir	Optional directory: when given, IOH-profiler logging is wired up in the constructor, before any evaluation is possible. Default NULL (no logging).
algo_name	Algorithm label recorded in the IOH archive (only meaningful when log_dir is given). Default "custom".
seed	Caller-declared reproducibility label recorded in the IOH archive meta only. Default 0.

Details

fid = 2 ("Sum of Different Powers") was officially withdrawn from the CEC 2017 suite: this constructor raises the dedicated sezgi_problems::Cec2017Error::Withdrawn message VERBATIM for it, distinct from an ordinary out-of-range fid.

See sz_eval_session for the full list of methods the returned object exposes (identical here, including $dim() / $bounds()).

Value

An 'EvalSession' object (see sz_eval_session).

---

sz_eval_session_cec2022 *Start an ask/tell evaluation session over a CEC 2022 problem.*

Description

The generic-session counterpart to sz_eval_session (which is BBOB-only): builds a session over sz_cec2022_evaluate's same Cec2022 problem, with IOH logging allowed (unlike sz_eval_session_f0, whose problem has no known optimum for a log archive to record). Session metadata (suite = "sezgi-cec2022", name = "cec2022-f<fid>", instance = 1, f_opt = the function's known optimum) matches py-sezgi's EvalSession.for_problem exactly for a CEC 2022 problem handle, so an IOH record produced by an R session and one produced by an equivalent Python session reconstruct to the identical (suite, fid, name,instance) key and f_opt.

Usage

```
sz_eval_session_cec2022(
  fid,
  dim,
  budget,
  log_dir = NULL,
  algo_name = "custom",
  seed = 0
)
```

Arguments

fid	CEC 2022 function id (1..=12).
dim	Problem dimension, one of 2, 10, 20 (dim = 2 is additionally rejected for a hybrid function, fid 6-8).
budget	Evaluation budget (non-negative whole number).
log_dir	Optional directory: when given, IOH-profiler logging is wired up in the constructor, before any evaluation is possible. Default NULL (no logging).
algo_name	Algorithm label recorded in the IOH archive (only meaningful when log_dir is given). Default "custom".
seed	Caller-declared reproducibility label recorded in the IOH archive meta only. Default 0.

Details

See sz_eval_session for the full list of methods the returned object exposes (identical here, including $dim() / $bounds()).

Value

An 'EvalSession' object (see sz_eval_session).

---

sz_eval_session_f0 *Start an ask/tell evaluation session over the f0 BIAS-toolbox null problem.*

Description

f0 (sezgi_bias::F0Random on the Rust side) has no landscape at all: every evaluation is an independent U(0,1) draw over the domain [0,1]^dim, uncorrelated with the point being queried. It exists purely as a probe for an algorithm's OWN structural bias – see sz_bias_structural.

Usage

```
sz_eval_session_f0(dim, f0_seed, budget)
```

Arguments

dim	f0's domain dimensionality (>= 1); the space is [0,1]^dim.
f0_seed	Seed for f0's own RNG stream – distinct from any engine RNG seed.
budget	Evaluation budget (non-negative whole number).

Details

Deliberately has NO log_dir/algo_name/seed (log) arguments: f0 has no known optimum, and IOH logging requires one (see sz_eval_session's log_dir for the BBOB/CEC-2022 case). $f_opt() on the returned session always returns NULL.

See sz_eval_session for the full list of methods the returned object exposes (identical here, including $dim() / $bounds(), which returns c(0, 1)).

Value

An 'EvalSession' object (see sz_eval_session).

`sz_eval_session_tsp` *Start an ask/tell evaluation session over a vendored TSPLIB (TSP) instance.*

Description

M3-8 Task 8: the PERMUTATION-typed counterpart to sz_eval_session and its continuous siblings – the R mirror of py-sezgi's M3-8 Task 7 sezgi.EvalSession.for_problem(sezgi.problems.tsp(...)) typed session. $kind() on the returned session is "permutation" (every other constructor in this file returns a "float"-kind session).

Usage

```
sz_eval_session_tsp(name, budget, seed = 0)
```

Arguments

name	A vendored TSPLIB instance name ("berlin52", "eil51", "st70"). UNLIKE sz_tsp_load()/ sz_tsp_tour_length(), raw TSPLIB file text is not accepted here – mirrors sz_solve_tsp()'s own vendored-only restriction, and py-sezgi's sezgi.problems.tsp(name).
budget	Evaluation budget (non-negative whole number).
seed	Master RNG seed for this session's own $random_permutation() draws (distinct from any IOH-log seed, since this session never logs). Default 0.

Details

UNLIKE every other sz_eval_session_*() constructor, this session's $evaluate() rows and $random_permutation()/$best() tour values are all 1-BASED (a permutation of 1:n_cities) – see sz_tsp_tour_length's own doc, "Index-convention decision" – NOT the dim-length numeric coordinates every continuous session above expects. The returned object additionally exposes:

- $kind() – always "permutation" for a session built by this constructor.
- $random_permutation() – a uniformly random 1-based tour (a permutation of 1:dim()), drawn from this session's own seeded Rust-side RNG stream (see "RNG parity with py-sezgi" below).

`$dim()` returns the instance's `n_cities`; `$bounds()` errors (there is no uniform `(lo, hi)` domain for a tour – use `$random_permutation()` instead of `ctx$random_point()` when authoring an algorithm over this session via `sz_algorithm`). Every other method (`$evaluate()`, `$evals_used()`, `$budget()`, `$best()`, `$f_opt()`, `$finish()`) behaves the same as for a continuous session, just over 1-based tour rows instead of coordinate rows.

RNG parity with py-sezgi: `$random_permutation()` draws from a Rust-side `RngStream` seeded via `RngStream::from_master(seed,&[PERM_SESSION_RNG_TAG])` – the IDENTICAL tag value and derivation py-sezgi's own `PermSession` uses, over the SAME shared `sezgi_components::perm::fisher_yates_shuf` core (see `src/rust/src/session.rs`'s module doc, "Permutation-typed sessions"). So for the SAME `seed`, an R session built here and a Python session built via `sezgi.EvalSession.for_problem(sezgi.problems.ts` draw the bit-identical UNDERLYING 0-based permutation on their first `random_permutation()` call – R simply displays it shifted `+1`.

Deliberately has NO `log_dir`/`algo_name` arguments: IOH logging is not wired up for permutation-typed sessions (no suite/fid identity exists to log a TSP run against) – same choice `sz_eval_session_f0` already makes for its own unsupported-logging case.

Value

An 'EvalSession' object (see above and `sz_eval_session`).

`sz_float` *A continuous block: 'n' coordinates, each in '[lo, hi]'.*

Description

Mirrors 'sezgi.Float' ('py-sezgi/python/sezgi/spaces.py') exactly: a plain value object, fields stored verbatim with no construction-time validation (bounds are enforced only once the space reaches the Rust core, e.g. 'SearchSpace::new', 'crates/core/src/space.rs').

Usage

```
sz_float(lo, hi, n)
```

Arguments

`lo`	Numeric scalar, the lower bound.
`hi`	Numeric scalar, the upper bound.
`n`	Numeric/integer scalar, the number of coordinates. Required – 'spaces.py''s 'Float.n' has no default either.

Value

An object of class 'c("sz_block_float", "sz_block")' with elements 'lo', 'hi', 'n'.

Examples

```
sz_float(-5, 5, 3)
```

sz_int *An integer block: ‘n‘ coordinates, each in ‘[lo, hi]‘ (inclusive).*

Description

Mirrors ‘sezgi.Int‘ (‘py-sezgi/python/sezgi/spaces.py‘) exactly – see [sz_float()]’s doc for the no-construction-time-validation note.

Usage

```
sz_int(lo, hi, n)
```

Arguments

lo	Numeric scalar, the lower bound.
hi	Numeric scalar, the upper bound.
n	Numeric/integer scalar, the number of coordinates. Required – ‘spaces.py‘’s ‘Float.n‘ has no default either.

Value

An object of class ‘c("sz_block_int", "sz_block")‘ with elements ‘lo‘, ‘hi‘, ‘n‘.

Examples

```
sz_int(0L, 10L, 2)
```

sz_mo_evaluate *Direct, one-shot objective evaluation of a decision vector ‘x‘ against any ‘sz_mo_*‘ problem string, bypassing ‘sz_nsga2()‘’s population/budget machinery entirely. Added (M3-7) so fixture-value tests can pin an exact ‘x‘ (‘sz_nsga2()‘’s randomly-initialized population cannot), mirroring the existing ‘sz_cec2022_evaluate()‘/‘sz_cec2014_evaluate()‘/ ‘sz_cec2017_evaluate()‘ one-shot-evaluation convention.*

Description

Direct, one-shot objective evaluation of a decision vector ‘x‘ against any ‘sz_mo_*‘ problem string, bypassing ‘sz_nsga2()‘’s population/budget machinery entirely. Added (M3-7) so fixture-value tests can pin an exact ‘x‘ (‘sz_nsga2()‘’s randomly-initialized population cannot), mirroring the existing ‘sz_cec2022_evaluate()‘/‘sz_cec2014_evaluate()‘/ ‘sz_cec2017_evaluate()‘ one-shot-evaluation convention.

Usage

```
sz_mo_evaluate(problem, dim, x, m = NULL, k = NULL, l = NULL)
```

Arguments

problem	Same mapping as ‘sz_nsga2()‘.
dim	Decision-space dimensionality – same rule as ‘sz_nsga2()‘.
x	A numeric vector, the decision vector to evaluate. For zdt5 (the only all-Binary problem reachable here), each entry is read as a bit (‘!= 0.0‘ -> ‘TRUE‘) – the same ‘0.0‘/‘1.0‘ convention ‘sz_nsga2()‘’s own ‘individuals‘ uses.
m	Number of objectives – same rule as ‘sz_nsga2()‘.
k	Optional WFG position-related-parameter count; wfg-only.
l	Optional WFG distance-related-parameter count; wfg-only.

Value

A numeric vector, length equal to the problem’s own objective count.

---

sz_mo_evaluate_constraints

The matching one-shot constraint-row evaluation for ‘sz_mo_evaluate()‘, same calling convention.

Description

The matching one-shot constraint-row evaluation for ‘sz_mo_evaluate()‘, same calling convention.

Usage

```
sz_mo_evaluate_constraints(problem, dim, x, m = NULL, k = NULL, l = NULL)
```

Arguments

problem	Same mapping as ‘sz_nsga2()‘.
dim	Decision-space dimensionality – same rule as ‘sz_nsga2()‘.
x	A numeric vector, the decision vector to evaluate.
m	Number of objectives – same rule as ‘sz_nsga2()‘.
k	Optional WFG position-related-parameter count; wfg-only.
l	Optional WFG distance-related-parameter count; wfg-only.

Value

A numeric vector (‘g_1..g_ncon‘, ‘g_j >= 0‘ meaning SATISFIED), or ‘NULL‘ for an unconstrained problem (every zdt/wfg problem, and dtlz1-7).

`sz_mo_hypervolume` *General-M exact hypervolume (While, Bradstreet & Barone 2012, the WFG algorithm; ‘M == 2‘ delegates internally to the SAME [‘sezgi_stats::hypervolume_2d‘]) – binds [‘sezgi_stats::hypervolume‘] (M3-7 Task 8/11). Unlike ‘sz_mo_hypervolume_2d‘, ‘front‘/‘ref_point‘ may have any number ‘M >= 1‘ of objectives.*

Description

‘ref_point‘ is REQUIRED, with no default: ‘sezgi_stats::moo_indicators‘’s own module doc, "Choosing a reference point: explicit-always, contested in the literature" section, deliberately never picks one for the caller. One common convention from that literature (also critiqued by Ishibuchi, Imada, Setoguchi & Nojima 2018, "How to Specify a Reference Point in Hypervolume Calculation for Fair Performance Comparison", GECCO Companion) is the analytic front's nadir point (the componentwise worst value across the front) scaled by ‘1.1‘ – a caller-supplied choice, never defaulted here.

Usage

```
sz_mo_hypervolume(front, ref_point)
```

Arguments

`front`	A numeric matrix, rows = points, any number of columns – see this module's own doc, "Container-idiom decisions".
`ref_point`	A numeric vector, same length as ‘front‘’s column count.

Value

A numeric scalar. An EMPTY ‘front‘ is NOT an error – it returns ‘0.0‘ (the algorithm's own base case).

Errors A savvy error for any [‘sezgi_stats::StatsError‘] (‘ref_point‘ empty, a ‘front‘ row with a different number of objectives than ‘ref_point‘, or a non-finite value), or if ‘front‘ is not a matrix.

`sz_mo_hypervolume_2d` *Exact 2-objective hypervolume (Zitzler & Thiele 1999 S-metric, reference-point variant; minimization) – binds [‘sezgi_stats::hypervolume_2d‘] exactly. See that function's doc for the pinned definition.*

Description

Exact 2-objective hypervolume (Zitzler & Thiele 1999 S-metric, reference-point variant; minimization) – binds [‘sezgi_stats::hypervolume_2d‘] exactly. See that function's doc for the pinned definition.

Usage

```
sz_mo_hypervolume_2d(front, ref_point)
```

Arguments

front	A numeric matrix, rows = points, 2 columns (‘f1‘, ‘f2‘) – R-idiomatic (unlike py-sezgi’s ‘list[[f1,f2],...]‘); see this module’s own doc, "Container-idiom decisions". Build with ‘rbind()‘/‘matrix()‘.
ref_point	A 2-element numeric vector.

Value

A numeric scalar.

Errors A savvy error if ‘ref_point‘ does not have exactly 2 values, if ‘front‘ is not a matrix, or for any [‘sezgi_stats::StatsError‘] (empty front, a non-2-objective row, or a non-finite value).

---

sz_mo_igd — *Inverted Generational Distance (Ishibuchi et al. 2015, eq. 12, ‘p = 1‘) – binds [‘sezgi_stats::igd‘] exactly. Any (equal, consistent) number of objectives across both ‘front‘ and ‘reference_front‘.*

Description

Inverted Generational Distance (Ishibuchi et al. 2015, eq. 12, ‘p = 1‘) – binds [‘sezgi_stats::igd‘] exactly. Any (equal, consistent) number of objectives across both ‘front‘ and ‘reference_front‘.

Usage

```
sz_mo_igd(front, reference_front)
```

Arguments

front	A numeric matrix, rows = points – see this module’s own doc, "Container-idiom decisions".
reference_front	A numeric matrix, rows = points, same column count as ‘front‘.

Value

A numeric scalar.

Errors A savvy error if either argument is not a matrix, or for any [‘sezgi_stats::StatsError‘] (an empty ‘front‘/‘reference_front‘, a dimension mismatch, or a non-finite value).

`sz_mo_pareto_front` *A deterministic ‘n‘-point sample of the analytic Pareto front in OBJECTIVE space, if known. See ‘MoProblem::pareto_front‘ in ‘crates/core/src/mo.rs‘.*

Description

A deterministic ‘n‘-point sample of the analytic Pareto front in OBJECTIVE space, if known. See ‘MoProblem::pareto_front‘ in ‘crates/core/src/mo.rs‘.

Usage

```
sz_mo_pareto_front(problem, dim, n, m = NULL, k = NULL, l = NULL)
```

Arguments

`problem`	Same ‘problem‘/‘dim‘/‘m‘/‘k‘/‘l‘ mapping as ‘sz_nsga2()‘.
`dim`	Decision-space dimensionality – same rule as ‘sz_nsga2()‘.
`n`	Number of front points to sample.
`m`	Number of objectives – same rule as ‘sz_nsga2()‘.
`k`	Optional WFG position-related-parameter count; wfg-only.
`l`	Optional WFG distance-related-parameter count; wfg-only.

Value

A ‘list‘ of ‘n‘ numeric vectors (mirrors ‘sz_nsga2()‘’s ‘objectives‘/‘individuals‘ container convention – a list, not a matrix), or ‘NULL‘ when the problem has no known analytic front sample at this ‘m‘ (verified cases: DTLZ5/DTLZ6 with ‘m > 3‘; WFG1/WFG2 unconditionally) – mirrors py-sezgi’s ‘sezgi.mo.pareto_front()‘ return exactly.

`sz_mo_read_moa` *Reads a "sezgi-moa v1" archive file written by ‘sz_nsga2(..., log_dir =, label =)‘ (M3-7 Task 9/11) – binds [‘sezgi_bench::read_moa‘]. Returns a named list: - ‘algo‘ (character): the logging algorithm name – always ‘"nsga2"‘ today (‘nsga2_run_logged‘’s own fixed ‘NSGA2_ALGO_NAME‘). - ‘problem‘ (character): the ‘label‘ ‘sz_nsga2‘ was called with. **Kept as the literal on-disk header key name** (‘crates/bench/src/mo_archive.rs‘’s own format grammar: the header line is ‘problem <label>‘, not ‘label <label>‘) rather than renamed here to ‘"label"‘ – this binding stays a thin, direct mirror of ‘MoArchiveRun‘’s own field names, so a reader cross-checking against the Rust struct (or the Python binding, M3-7 Task 10) sees the SAME key everywhere. - ‘m‘, ‘seed‘, ‘budget‘ (double, whole-number-valued – R has no native integer64). - ‘kind‘ (character): ‘"float"‘ or ‘"binary"‘. - ‘records‘ (list, in file/eval order): each entry a named list with ‘eval_index‘ (double), ‘objectives‘ (numeric vector), ‘genotype‘ (numeric vector for ‘kind = "float"‘, ‘logical‘ vector for ‘kind = "binary"‘ – see this module’s own doc, "Container-idiom decisions"). - ‘archive‘ (list of numeric vectors): the reconstructed nondominated archive at evaluation budget ‘at‘, via ‘MoArchiveRun::archive_at‘.*

Description

‘# sezgi decision:‘ ‘at = NULL‘ (the default) resolves to the file’s own logged ‘budget‘ header field (the full run’s final archive) – mirrors py-sezgi’s own ‘mo.read_moa(path, at=None)‘ default exactly.

Usage

```
sz_mo_read_moa(path, at = NULL)
```

Arguments

`path`	Path to a sezgi-moa v1 file.
`at`	Optional evaluation budget (double, cast to ‘u64‘) at which to reconstruct the archive; ‘NULL‘ (default) uses the file’s own logged ‘budget‘.

Details

This is directly ‘@export‘ed (no hand-written wrapper needed): ‘at‘ is the only optional parameter and trails ‘path‘, so savvy already emits ‘at = NULL‘ in the generated R signature – same pattern as ‘sz_mo_hypervolume_2d‘/‘sz_mo_igd‘.

Value

A named list – see this function’s own doc above.

Errors A savvy error for any [‘sezgi_bench::MoArchiveError‘] (missing file, a malformed header, or a malformed record line).

sz_nsga2	*NSGA-II (Deb, Pratap, Agarwal & Meyarivan 2002) run on a ZDT/DTLZ/WFG multi-objective test problem. See ‘crates/components/src/nsga2.rs‘ for the full algorithm provenance.*

Description

NSGA-II (Deb, Pratap, Agarwal & Meyarivan 2002) run on a ZDT/DTLZ/WFG multi-objective test problem. See ‘crates/components/src/nsga2.rs‘ for the full algorithm provenance.

Usage

```
sz_nsga2(
  problem,
  dim,
  m = NULL,
  pop_size,
  budget,
  seed = 0,
  eta_c = 20,
  eta_m = 20,
  p_c = 0.9,
  p_m = NULL,
  p_c_bin = 0.9,
  p_m_bin = NULL,
  p_c_cat = 0.9,
  p_m_cat = NULL,
  k = NULL,
  l = NULL,
  log_dir = NULL,
  label = NULL
)
```

Arguments

problem	One of ‘"zdt1"‘, ‘"zdt2"‘, ‘"zdt3"‘, ‘"zdt4"‘, ‘"zdt5"‘ (binary-coded, fixed 80-bit layout), ‘"zdt6"‘, ‘"dtlz1"‘..‘"dtlz9"‘ (‘"dtlz8"‘/‘"dtlz9"‘ are constrained), or ‘"wfg1"‘..‘"wfg9"‘.
dim	Decision-space dimensionality. REQUIRED (may be ‘NULL‘, but the argument itself must be supplied) for zdt1-4/6 and dtlz1-9; REJECTED (must be ‘NULL‘) for zdt5 (fixed 80-bit layout) and wfg1-9 (dimension is derived from ‘k‘/‘l‘).
m	Number of objectives. REQUIRED for dtlz/wfg problems; must be ‘NULL‘ (default, and the only valid value) for zdt problems, which are always 2-objective by construction – passing ‘m‘ for a zdt problem is an error, not a silently-ignored argument.

`pop_size` Population size. Must be '>= 4' and a multiple of 4 (KanGAL's double-permutation tournament pairing requires it – NOT merely "even, >= 4"; 'pop_size = 6' is rejected the same as 'pop_size = 7').

`budget` Total evaluation budget (init + every generation).

`seed` Master RNG seed. Default '0'.

`eta_c` SBX distribution index. Default '20.0' (Deb et al. 2002, Sec. IV.A's own experimental setting).

`eta_m` Polynomial-mutation distribution index. Default '20.0'.

`p_c` SBX crossover probability. Default '0.9'.

`p_m` Per-variable mutation probability. 'NULL' (default) resolves on the Rust side to '1 / n_variables' (the paper's own default), never re-derived here.

`p_c_bin` Binary-genotype crossover probability (M3-7). Default '0.9', mirroring 'p_c''s own default (the paper gives no verified binary-specific default) – only consulted when 'problem' builds an all-Binary space (zdt5 today).

`p_m_bin` Optional per-bit binary mutation probability. 'NULL' (default) resolves on the Rust side to '1 / l' ('l' = the space's total bit count, the paper's own stated binary-coded default) – only consulted for an all-Binary space.

`p_c_cat` Categorical-genotype crossover probability (post-M3-8 deferral cleanup). Default '0.9', mirroring 'p_c_bin''s own default (no paper/reference precedent exists for Categorical) – only consulted when 'problem' builds a 'Mixed' space containing a 'Block::Categorical' block. No 'problem' string accepted here builds one today (zdt1-6, dtlz1-9, wfg1-9 are all-Float or, for zdt5, all-Binary), so this is currently validated but inert against every reachable problem – exposed anyway for symmetry with 'p_c_bin'/'p_m_bin'.

`p_m_cat` Optional per-gene Categorical mutation probability (post-M3-8 deferral cleanup). 'NULL' (default) resolves on the Rust side to '1 / n_cat' ('n_cat' = the space's total flattened 'Block::Categorical' dimension), mirroring 'p_m'/'p_m_bin''s own 'NULL'-resolves-to-a-formula design – only consulted for a 'Mixed' space containing a Categorical block (currently unreachable through this catalog – see 'p_c_cat' above).

`k` Optional WFG position-related-parameter count. 'NULL' (default) resolves to the toolkit's own recommended value ('k = 4' for 'm = 2', 'k = 2*(m-1)' for 'm >= 3'); wfg-only, an error for any other problem family.

`l` Optional WFG distance-related-parameter count. 'NULL' (default) resolves to '20'; wfg-only.

`log_dir` Optional sezgi-moa v1 log directory (M3-7 archive-first run logging). When given, 'label' is required and the run additionally streams every feasible archive insertion to '<log_dir>/<label>-s<seed>.moa' – see 'sz_mo_read_moa()'. Omitting 'log_dir' runs exactly as before (byte-identical to the unlogged path).

`label` Optional sezgi-moa run label; REQUIRED iff 'log_dir' is given, otherwise ignored.

Value

A named list with ‘individuals‘ (list of numeric vectors, one per final-population member, each length ‘dim‘ – flattened across every block; a ‘Block::Binary‘ block’s bits are flattened to ‘0.0‘/‘1.0‘, e.g. zdt5’s own path), ‘objectives‘ (list of numeric vectors, parallel to ‘individuals‘, each length equal to the problem’s objective count), ‘front0‘ (numeric vector of 1-based positions, R convention, into ‘individuals‘/‘objectives‘ for the final population’s non-dominated set – NOTE: py-sezgi’s ‘front0‘ is 0-based; this binding converts), ‘evals_used‘ (double), and (present ONLY for a constrained problem – dtlz8/dtlz9 today) ‘violations‘ (numeric vector, ‘<= 0.0‘, ‘0.0‘ = feasible, parallel to ‘individuals‘/‘objectives‘) – mirrors py-sezgi’s ‘sezgi.mo.nsga2()‘ dict keys exactly.

Examples

```
r <- sz_nsga2("zdt1", dim = 5, pop_size = 8, budget = 40, seed = 1)
length(r$front0)
```

---

sz_permutation *A permutation block: one permutation of ‘0..n‘.*

Description

Mirrors ‘sezgi.Permutation‘ (‘py-sezgi/python/sezgi/spaces.py‘) exactly – see [sz_float()]’s doc for the no-construction-time-validation note.

Usage

```
sz_permutation(n)
```

Arguments

n	Numeric/integer scalar, the permutation length.

Value

An object of class ‘c("sz_block_permutation", "sz_block")‘ with element ‘n‘.

---

sz_per_budget_packages *Build one paper-package statistics list PER DISTINCT BUDGET present in a ‘sz_run_experiment()‘ data.frame, in ascending budget order.*

Description

Mirrors py-sezgi’s ‘per_budget_packages()‘. Piotrowski et al. (2025) show algorithm rankings on benchmark comparisons can flip depending on which evaluation budget is examined, so this makes multi-budget reporting the default rather than a single, arbitrarily-chosen budget’s report: compare algorithms per budget, never pooled across budgets.

Usage

```
sz_per_budget_packages(
  df,
  rope = 0,
  samples = 20000,
  seed = 1,
  aggregate = "mean"
)
```

Arguments

df	A data.frame as returned by ‘sz_run_experiment()‘. ‘suite‘ is optional – a data.frame without it is treated as all-BBOB (see ‘sz_results_matrix()‘).
rope	Region of practical equivalence half-width (>= 0) for the Bayesian signed-rank test, forwarded to every budget's package. Default 0.
samples	Number of Monte Carlo samples per pair. Default 20000.
seed	Master RNG seed, forwarded to every budget's package. Default 1.
aggregate	How to combine a (problem, algorithm) cell's per-seed gaps: ‘"mean"‘ or ‘"median"‘. Default ‘"mean"‘.

Value

A named list, one entry per distinct budget in ascending order, named by the budget (as a string); each value has exactly the shape ‘sz_stats_paper_package()‘ returns.

---

sz_preset_abc

Builds an Artificial Bee Colony spec (Karaboga 2005, TR-06 / Karaboga & Basturk 2007, Journal of Global Optimization – a labeled metaphor preset; see ‘crates/components/src/abc.rs‘’s module doc for the full provenance extraction against the author's own ‘ABCorig.m‘ plus the official ‘Python_ABC‘ port, the pop<->food-source convention resolution, the fitness-transform monotonicity proof, and the phase-design adjudication – a SINGLE stage, not two symmetric stages like ‘tlbo‘: ‘gen/abc-employed‘ + the new ‘replace/abc-trial-greedy‘, plus the new ‘adapter/abc-onlooker-scout‘ folding the onlooker AND scout phases together) as JSON, ready to pass to ‘sz_solve_bbob()‘.

Description

Builds an Artificial Bee Colony spec (Karaboga 2005, TR-06 / Karaboga & Basturk 2007, Journal of Global Optimization – a labeled metaphor preset; see ‘crates/components/src/abc.rs‘’s module doc for the full provenance extraction against the author's own ‘ABCorig.m‘ plus the official ‘Python_ABC‘ port, the pop<->food-source convention resolution, the fitness-transform monotonicity proof, and the phase-design adjudication – a SINGLE stage, not two symmetric stages like ‘tlbo‘: ‘gen/abc-employed‘ + the new ‘replace/abc-trial-greedy‘, plus the new ‘adapter/abc-onlooker-scout‘ folding the onlooker AND scout phases together) as JSON, ready to pass to ‘sz_solve_bbob()‘.

Usage

```
sz_preset_abc(pop_size, budget)
```

Arguments

pop_size	Population size (food-source count ‘SN‘, NOT Karaboga’s colony size ‘NP=2*SN‘). Canonical is 20.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_alo` *Builds an Ant Lion Optimizer spec (Mirjalili 2015, Advances in Engineering Software – a labeled metaphor preset; see ‘crates/components/src/alo.rs‘’s module doc for the full provenance extraction against the author’s own ‘ALO.m‘/‘Random_walk_around_ antlion.m‘/‘RouletteWheelSelection.m‘, the faithful-full-walk cost decision, and the elitism design adjudication – the antlion population itself is the persisted memory via ‘replace/mu-plus-lambda‘, no blackboard state needed) as JSON, ready to pass to ‘sz_solve_bbob()‘.*

Description

Builds an Ant Lion Optimizer spec (Mirjalili 2015, Advances in Engineering Software – a labeled metaphor preset; see ‘crates/components/src/alo.rs‘’s module doc for the full provenance extraction against the author’s own ‘ALO.m‘/‘Random_walk_around_ antlion.m‘/‘RouletteWheelSelection.m‘, the faithful-full-walk cost decision, and the elitism design adjudication – the antlion population itself is the persisted memory via ‘replace/mu-plus-lambda‘, no blackboard state needed) as JSON, ready to pass to ‘sz_solve_bbob()‘.

Usage

```
sz_preset_alo(pop_size, budget)
```

Arguments

pop_size	Population size (ant/antlion count). Canonical is 25.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_bat` *Builds a Bat Algorithm spec (Yang, X.-S. 2010, NICSO – a labeled metaphor preset, see 'crates/components/src/ba.rs''s module doc for the tier note, citation, the verified 'bat_algorithm.m' loop structure, the verified fixed-loudness/pulse-rate finding, the two composing sign-inversion deltas in the frequency draw and velocity term, and the design adjudication for the new 'replace/bat-loudness-greedy' acceptance-coupled replacer) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Bat Algorithm spec (Yang, X.-S. 2010, NICSO – a labeled metaphor preset, see 'crates/components/src/ba.rs''s module doc for the tier note, citation, the verified 'bat_algorithm.m' loop structure, the verified fixed-loudness/pulse-rate finding, the two composing sign-inversion deltas in the frequency draw and velocity term, and the design adjudication for the new 'replace/bat-loudness-greedy' acceptance-coupled replacer) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_bat(pop_size, budget)
```

Arguments

`pop_size`	Population size (number of bats). Canonical is 30.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_classes` *Table-driven preset-backed algorithm wrapper classes.*

Description

One R6 class per non-GA/DE 'sz_preset_*' builder ('R/000-wrappers.R' + 'R/presets.R', both mirroring 'crates/components/src/presets.rs') – 26 classes, generated from ONE row table ('.sz_preset_table') by ONE factory ('.sz_make_preset_class()', both internal to 'R/builtins.R'). Every class shares the identical shape: '$new(pop_size = NULL, ...)' stores constructor kwargs only (no computation happens at construction time); '$run(problem, budget, seed = 0, run_id = 0)' builds the preset's 'spec_json' via the matching 'sz_preset_*(...)' call (bit-identical to calling that function directly with the same arguments – see 'test-oop-builtins.R''s own table-driven anchor test), routes it through 'sz_solve_bbob()'/'sz_solve_r_problem()' depending on 'problem''s own form, and wraps the raw 3-field result via T4's '.sz_wrap_result()' into an 'sz_result' (the SAME shape [Algorithm]'s own 'run()' returns).

Details

'pop_size' semantics depend on the preset's own Rust signature (see each 'sz_preset_*()''s own doc in 'R/000-wrappers.R' for its "Canonical is N" convention, where stated):

most presets 'sz_preset_*(pop_size, budget, ...)' – 'pop_size' defaults to the class's own documented convention (the source paper's canonical value where '000-wrappers.R' states one; '20' otherwise, matching this repo's existing dominant convention for presets with no single canonical source), and MAY be overridden at '$new()'.

'SimulatedAnnealing' 'sz_preset_sa(budget)' has NO 'pop_size' parameter at all (a single search trajectory) – 'pop_size' is fixed at '1'; a non-'NULL', non-'1' value at '$new()' raises a clear error.

'CMAESIpop'/'LSHADE'/'NelderMead' 'sz_preset_*(dim, budget, ...)' DERIVES its own population size from the problem's dimensionality at '$run()' time ('dim' is read from the resolved problem, NOT 'pop_size') – ANY 'pop_size' given at '$new()' raises a clear error naming the derivation formula.

'EvolutionStrategy' additionally accepts 'dist='/'mean='/'sigma='/ 'loc='/'scale='/'alpha='/'nu=' at '$new()' (flowing through to 'sz_preset_es_mu_plus_lambda()' unchanged, via '...') – the ONLY table row whose preset takes kwargs beyond 'pop_size'/'budget'.

Problem-form support matrix (identical for every class in this family, and for [GeneticAlgorithm]/[DifferentialEvolution]): 'problem' is EITHER a [Problem] subclass instance (routed through 'sz_solve_r_problem()', 'f_opt = prob$optimum()') OR an [sz_builtin_bbob()] descriptor (routed through 'sz_solve_bbob()', 'f_opt = NULL') – the SAME two forms [Algorithm]'s own 'run()' accepts. No CEC2014/CEC2017/CEC2022/TSP/onemax/etc. native-descriptor path exists yet.

`sz_preset_cmaes` *Builds a (mu/mu_w,lambda)-CMA-ES algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a (mu/mu_w,lambda)-CMA-ES algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_cmaes(pop_size, budget)
```

Arguments

`pop_size`	Population size (lambda).
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_cmaes_ipop` *Builds a CMA-ES with IPOP-style stagnation restarts algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a CMA-ES with IPOP-style stagnation restarts algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_cmaes_ipop(dim, budget)
```

Arguments

`dim`	Problem dimension (determines the initial population size).
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_cuckoo_search`
Builds a Cuckoo Search algorithm spec (Yang & Deb 2009 – a labeled metaphor preset, see 'crates/components/src/cs.rs''s module doc for the tier note, citation and pinned draw order) as JSON, ready to pass to 'sz_solve_bbob()'.

Description

Builds a Cuckoo Search algorithm spec (Yang & Deb 2009 – a labeled metaphor preset, see 'crates/components/src/cs.rs''s module doc for the tier note, citation and pinned draw order) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_cuckoo_search(pop_size, budget)
```

Arguments

`pop_size`	Population size (nest count). Canonical is 25.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_de_best_1` *Builds a DE/best/1/bin algorithm spec (uniform init, clamp boundary, one-to-one-greedy replacement) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a DE/best/1/bin algorithm spec (uniform init, clamp boundary, one-to-one-greedy replacement) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_de_best_1(pop_size, budget)
```

Arguments

`pop_size`	Population size.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_de_rand_1` *Builds a DE/rand/1/bin algorithm spec (uniform init, clamp boundary, one-to-one-greedy replacement) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a DE/rand/1/bin algorithm spec (uniform init, clamp boundary, one-to-one-greedy replacement) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_de_rand_1(pop_size, budget)
```

Arguments

`pop_size`	Population size.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

Examples

```
sz_preset_de_rand_1(pop_size = 10, budget = 100)
```

`sz_preset_es_mu_plus_lambda` *Builds a (mu/mu_w,lambda)-ES algorithm spec (mutation step drawn from ‘dist‘) as JSON, ready to pass to ‘sz_solve_bbob()‘.*

Description

Mirrors py-sezgi’s ‘sezgi.presets.es_mu_plus_lambda()‘.

Usage

```
sz_preset_es_mu_plus_lambda(
  pop_size,
  budget,
  dist = "gaussian",
  mean = 0,
  sigma = 0.5,
  loc = 0,
  scale = 1,
  alpha = 1.5,
  nu = 3
)
```

Arguments

`pop_size`	Population size.
`budget`	Evaluation budget.
`dist`	Mutation distribution: one of ‘"uniform"‘, ‘"gaussian"‘, ‘"cauchy"‘, ‘"levy"‘, ‘"student_t"‘, ‘"laplace"‘. Default ‘"gaussian"‘. An unrecognized value raises an error naming it and the recognized set.
`mean`	Gaussian mean (used only when ‘dist = "gaussian"‘). Default 0.
`sigma`	Gaussian std-dev (used only when ‘dist = "gaussian"‘). Default 0.5.
`loc`	Cauchy/Laplace location (used only when ‘dist‘ is ‘"cauchy"‘ or ‘"laplace"‘). Default 0.
`scale`	Cauchy/Laplace scale (used only when ‘dist‘ is ‘"cauchy"‘ or ‘"laplace"‘). Default 1.
`alpha`	Levy stability parameter (used only when ‘dist = "levy"‘). Default 1.5.
`nu`	Student-t degrees of freedom (used only when ‘dist = "student_t"‘). Default 3.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_firefly` *Builds a Firefly Algorithm (Yang, X.-S., *Nature-Inspired Metaheuristic Algorithms*, 2nd ed., Luniver Press, 2010 – a labeled metaphor preset, see 'crates/components/src/fa.rs''s module doc for the tier note, citation, the verified 'fa_ndim.m'/'ffa_move.m' loop structure, the floored attractiveness formula, the closed-form 'alpha' decay, and the hybrid in-place-self/live-distance/frozen-target double-loop semantics) spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Firefly Algorithm (Yang, X.-S., *Nature-Inspired Metaheuristic Algorithms*, 2nd ed., Luniver Press, 2010 – a labeled metaphor preset, see 'crates/components/src/fa.rs''s module doc for the tier note, citation, the verified 'fa_ndim.m'/'ffa_move.m' loop structure, the floored attractiveness formula, the closed-form 'alpha' decay, and the hybrid in-place-self/live-distance/frozen-target double-loop semantics) spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_firefly(pop_size, budget)
```

Arguments

`pop_size`	Population size (number of fireflies). Canonical is 25.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_fpa` *Builds a Flower Pollination Algorithm spec (Yang, X.-S. 2012, UCNC – a labeled metaphor preset, see 'crates/components/src/fpa.rs''s module doc for the tier note, citation, the verified 'fpa_demo.m' loop structure, the switch-branch orientation delta, the global-step sign delta reusing 'cs.rs''s 'cs_dim_step' verbatim, the local-step self-selection-not- excluded finding, and the min_pop adjustment from 3 down to 2) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Flower Pollination Algorithm spec (Yang, X.-S. 2012, UCNC – a labeled metaphor preset, see 'crates/components/src/fpa.rs''s module doc for the tier note, citation, the verified 'fpa_demo.m' loop structure, the switch-branch orientation delta, the global-step sign delta reusing 'cs.rs''s 'cs_dim_step' verbatim, the local-step self-selection-not- excluded finding, and the min_pop adjustment from 3 down to 2) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_fpa(pop_size, budget)
```

Arguments

pop_size	Population size (number of flowers). Canonical is 25.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_ga_bin` *Builds a Binary-space GA spec (tournament selection, uniform crossover, bit-flip mutation – 'gen/ga-bin' + 'replace/mu-plus-lambda') as JSON, ready to pass to 'sz_solve_onemax()'. Binds ['sezgi_components::presets::ga_bin'] exactly – M3-8 Task 10, mirroring py-sezgi's 'sezgi.presets.ga_bin' (M3-8 Task 9).*

Description

Builds a Binary-space GA spec (tournament selection, uniform crossover, bit-flip mutation – 'gen/ga-bin' + 'replace/mu-plus-lambda') as JSON, ready to pass to 'sz_solve_onemax()'. Binds ['sezgi_components::presets::ga_bin exactly – M3-8 Task 10, mirroring py-sezgi's 'sezgi.presets.ga_bin' (M3-8 Task 9).

Usage

```
sz_preset_ga_bin(pop_size, budget)
```

Arguments

pop_size	Population size.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_ga_cat` *Builds a Categorical-space GA spec (tournament selection, uniform crossover, random-reset mutation – 'gen/ga-cat' + 'replace/mu-plus-lambda') as JSON, ready to pass to 'sz_solve_cat_match()'. Binds ['sezgi_components::presets::ga_cat'] exactly – M3-8 Task 10, mirroring py-sezgi's 'sezgi.presets.ga_cat' (M3-8 Task 9).*

Description

Builds a Categorical-space GA spec (tournament selection, uniform crossover, random-reset mutation – 'gen/ga-cat' + 'replace/mu-plus-lambda') as JSON, ready to pass to 'sz_solve_cat_match()'. Binds ['sezgi_components::presets::ga_cat'] exactly – M3-8 Task 10, mirroring py-sezgi's 'sezgi.presets.ga_cat' (M3-8 Task 9).

Usage

```
sz_preset_ga_cat(pop_size, budget)
```

Arguments

`pop_size`	Population size.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_ga_int` *Builds an Int-space GA spec (tournament selection, SBX-style integer crossover, polynomial-style integer mutation – 'gen/ga-int' + 'replace/mu-plus-lambda') as JSON, ready to pass to 'sz_solve_int_quadratic()'. Binds ['sezgi_components::presets::ga_int'] exactly – M3-8 Task 10, mirroring py-sezgi's 'sezgi.presets.ga_int' (M3-8 Task 9).*

Description

Builds an Int-space GA spec (tournament selection, SBX-style integer crossover, polynomial-style integer mutation – 'gen/ga-int' + 'replace/mu-plus-lambda') as JSON, ready to pass to 'sz_solve_int_quadratic()'. Binds ['sezgi_components::presets::ga_int'] exactly – M3-8 Task 10, mirroring py-sezgi's 'sezgi.presets.ga_int' (M3-8 Task 9).

Usage

```
sz_preset_ga_int(pop_size, budget)
```

Arguments

pop_size	Population size.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_ga_perm` *Builds a permutation-space GA spec (tournament selection, order crossover, swap mutation – 'gen/ga-perm' + 'replace/mu-plus-lambda') as JSON, ready to pass to 'sz_solve_tsp()'. Binds ['sezgi_components::presets::ga_perm'] exactly – same preset py-sezgi's 'sezgi.presets.ga_perm' binds (M3-3 Task 9).*

Description

Builds a permutation-space GA spec (tournament selection, order crossover, swap mutation – 'gen/ga-perm' + 'replace/mu-plus-lambda') as JSON, ready to pass to 'sz_solve_tsp()'. Binds ['sezgi_components::presets::ga_perm' exactly – same preset py-sezgi's 'sezgi.presets.ga_perm' binds (M3-3 Task 9).

Usage

```
sz_preset_ga_perm(pop_size, budget)
```

Arguments

pop_size	Population size.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_ga_real` *Builds a real-coded GA (SBX crossover, polynomial mutation) algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a real-coded GA (SBX crossover, polynomial mutation) algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_ga_real(pop_size, budget)
```

Arguments

`pop_size`	Population size.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_goa` *Builds a Grasshopper Optimisation Algorithm spec (Saremi, Mirjalili & Lewis 2017 – a labeled metaphor preset, see 'crates/components/src/goa.rs''s module doc for the tier note, citation, the IMPLEMENTER-VERIFY distance-normalization resolution and the zero-RNG-draw arithmetic-order pin) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Grasshopper Optimisation Algorithm spec (Saremi, Mirjalili & Lewis 2017 – a labeled metaphor preset, see 'crates/components/src/goa.rs''s module doc for the tier note, citation, the IMPLEMENTER-VERIFY distance-normalization resolution and the zero-RNG-draw arithmetic-order pin) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_goa(pop_size, budget)
```

Arguments

`pop_size`	Population size (swarm size). Canonical is 30.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_gsa` *Builds a Gravitational Search Algorithm spec (Rashedi, Nezamabadi-pour & Saryazdi 2009, Information Sciences – a labeled metaphor preset, and the wave's LAST stateful/blackboard algorithm; see 'crates/components/src/gsa.rs''s module doc for the full provenance extraction against the author's own 'GSA.m'/'Gconstant.m'/ 'massCalculation.m'/'Gfield.m'/'move.m', the verified 'M_i'-free force delta, and the confirmation that GSA's own 'Fbest'/'Lbest' never feed back into the mechanism) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Gravitational Search Algorithm spec (Rashedi, Nezamabadi-pour & Saryazdi 2009, Information Sciences – a labeled metaphor preset, and the wave's LAST stateful/blackboard algorithm; see 'crates/components/src/gsa.rs''s module doc for the full provenance extraction against the author's own 'GSA.m'/'Gconstant.m'/ 'massCalculation.m'/'Gfield.m'/'move.m', the verified 'M_i'-free force delta, and the confirmation that GSA's own 'Fbest'/'Lbest' never feed back into the mechanism) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_gsa(pop_size, budget)
```

Arguments

`pop_size`	Population size (agent count). Canonical is 30.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_gwo` *Builds a Grey Wolf Optimizer algorithm spec (Mirjalili, Mirjalili & Lewis 2014 – a labeled metaphor preset, see 'crates/components/src/gwo.rs''s module doc for the tier note and citations) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Grey Wolf Optimizer algorithm spec (Mirjalili, Mirjalili & Lewis 2014 – a labeled metaphor preset, see 'crates/components/src/gwo.rs''s module doc for the tier note and citations) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_gwo(pop_size, budget)
```

Arguments

pop_size	Population size (pack size). Canonical is 30.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_harmony_search` *Builds a Harmony Search algorithm spec (Geem, Kim & Loganathan 2001 – a labeled metaphor preset, see 'crates/components/src/hs.rs''s module doc for the tier note and citations) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Harmony Search algorithm spec (Geem, Kim & Loganathan 2001 – a labeled metaphor preset, see 'crates/components/src/hs.rs''s module doc for the tier note and citations) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_harmony_search(pop_size, budget)
```

Arguments

pop_size	Population size (Harmony Memory Size, HMS). Canonical is 30.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

sz_preset_hho — *Builds a Harris Hawks Optimization spec (Heidari, Mirjalili, Faris, Aljarah, Mafarja & Chen 2019, Future Generation Computer Systems – a labeled metaphor preset, and the wave's most structurally complex one: a multi-branch escape-energy tree whose progressive rapid-dive sub-branches evaluate mid-‘generate()‘. See ‘crates/components/src/hho.rs‘'s module doc for the full provenance extraction against the paper author's own ‘HHO.m‘, the hard/soft besiege mapping delta, the mean(X)/random-hawk in-place semantics, and the prominent in-generator-evaluation eval-accounting design decision) as JSON, ready to pass to ‘sz_solve_bbob()‘.*

Description

Builds a Harris Hawks Optimization spec (Heidari, Mirjalili, Faris, Aljarah, Mafarja & Chen 2019, Future Generation Computer Systems – a labeled metaphor preset, and the wave's most structurally complex one: a multi-branch escape-energy tree whose progressive rapid-dive sub-branches evaluate mid-‘generate()‘. See ‘crates/components/src/hho.rs‘'s module doc for the full provenance extraction against the paper author's own ‘HHO.m‘, the hard/soft besiege mapping delta, the mean(X)/random-hawk in-place semantics, and the prominent in-generator-evaluation eval-accounting design decision) as JSON, ready to pass to ‘sz_solve_bbob()‘.

Usage

```
sz_preset_hho(pop_size, budget)
```

Arguments

pop_size	Population size (hawk count). Canonical is 30.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

sz_preset_jaya — *Builds a JAYA spec (Rao 2016 – a labeled metaphor preset, see ‘crates/components/src/jaya.rs‘'s module doc for the tier note, citation, the primary-paper-verified worked-example reproduction, the shared-per-dimension-per-generation ‘r1‘/‘r2‘ draw finding and the greedy-replacement delta vs mealpy's misleadingly-named ‘OriginalJA‘) as JSON, ready to pass to ‘sz_solve_bbob()‘.*

Description

Builds a JAYA spec (Rao 2016 – a labeled metaphor preset, see ‘crates/components/src/jaya.rs‘’s module doc for the tier note, citation, the primary-paper-verified worked-example reproduction, the shared-per-dimension-per-generation ‘r1‘/‘r2‘ draw finding and the greedy-replacement delta vs mealpy’s misleadingly-named ‘OriginalJA‘) as JSON, ready to pass to ‘sz_solve_bbob()‘.

Usage

```
sz_preset_jaya(pop_size, budget)
```

Arguments

pop_size	Population size (candidate count). Canonical is 30.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_jde` *Builds a jDE algorithm spec (self-adaptive F/CR DE) as JSON, ready to pass to ‘sz_solve_bbob()‘.*

Description

Builds a jDE algorithm spec (self-adaptive F/CR DE) as JSON, ready to pass to ‘sz_solve_bbob()‘.

Usage

```
sz_preset_jde(pop_size, budget)
```

Arguments

pop_size	Population size.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_lshade` *Builds an L-SHADE algorithm spec (population linearly reduced from '18 * dim') as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds an L-SHADE algorithm spec (population linearly reduced from '18 * dim') as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_lshade(dim, budget)
```

Arguments

`dim`	Problem dimension (determines the initial population size).
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_mfo` *Builds an MFO (Moth-Flame Optimization; Mirjalili 2015 – a labeled metaphor preset, see 'crates/components/src/mfo.rs''s module doc for the tier note, citation, the verified 'MFO.m' loop structure, the two subtle draw/index deltas found vs the plan's sketch, and the blackboard flame-memory design) spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds an MFO (Moth-Flame Optimization; Mirjalili 2015 – a labeled metaphor preset, see 'crates/components/src/mfo.rs''s module doc for the tier note, citation, the verified 'MFO.m' loop structure, the two subtle draw/index deltas found vs the plan's sketch, and the blackboard flame-memory design) spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_mfo(pop_size, budget)
```

Arguments

`pop_size`	Population size (number of search agents). Canonical is 30.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_nelder_mead` *Builds a Nelder-Mead simplex algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Nelder-Mead simplex algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_nelder_mead(dim, budget)
```

Arguments

`dim`	Problem dimension (population size is fixed to 'dim + 1').
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_pso` *Builds a PSO (Clerc-Kennedy constriction) algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a PSO (Clerc-Kennedy constriction) algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_pso(pop_size, budget)
```

Arguments

`pop_size`	Population size (swarm size).
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_random_search` *Builds a random search algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a random search algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_random_search(pop_size, budget)
```

Arguments

`pop_size`	Population size (resampled uniformly each generation).
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_sa` *Builds a simulated annealing (Metropolis, geometric cooling) algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a simulated annealing (Metropolis, geometric cooling) algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_sa(budget)
```

Arguments

`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_sca` *Builds a Sine Cosine Algorithm spec (Mirjalili 2016 – a labeled metaphor preset, see 'crates/components/src/sca.rs''s module doc for the tier note, citation, the 'SCA.m'-verified pinned draw order and the mealpy-'OriginalSCA' replacer delta) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Sine Cosine Algorithm spec (Mirjalili 2016 – a labeled metaphor preset, see 'crates/components/src/sca.rs''s module doc for the tier note, citation, the 'SCA.m'-verified pinned draw order and the mealpy-'OriginalSCA' replacer delta) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_sca(pop_size, budget)
```

Arguments

`pop_size`	Population size (number of search agents). Canonical is 30.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_shade` *Builds a SHADE algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a SHADE algorithm spec as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_shade(pop_size, budget)
```

Arguments

`pop_size`	Population size.
`budget`	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

sz_preset_ssa	*Builds an SSA (Salp Swarm Algorithm; Mirjalili et al. 2017 – a labeled metaphor preset, see ‘crates/components/src/ssa.rs‘’s module doc for the tier note, citation, the verified ‘SSA.m‘ half-population leader/follower split, the leader sign-branch pin, the verified in-place follower-chain semantics, and the persisted-food-vs-current-pop-best delta) spec as JSON, ready to pass to ‘sz_solve_bbob()‘.*

Description

Builds an SSA (Salp Swarm Algorithm; Mirjalili et al. 2017 – a labeled metaphor preset, see ‘crates/components/src/ssa.rs‘’s module doc for the tier note, citation, the verified ‘SSA.m‘ half-population leader/follower split, the leader sign-branch pin, the verified in-place follower-chain semantics, and the persisted-food-vs-current-pop-best delta) spec as JSON, ready to pass to ‘sz_solve_bbob()‘.

Usage

```
sz_preset_ssa(pop_size, budget)
```

Arguments

pop_size	Population size (number of salps). Canonical is 30.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

sz_preset_tlbo	*Builds a Teaching-Learning-Based Optimization spec (Rao, Savsani & Vakharia 2011, Computer-Aided Design – a labeled metaphor preset, and sezgi’s FIRST multi-stage preset: two ‘[[stages]]‘ (teacher, then learner) run in sequence every generation. See ‘crates/components/src/tlbo.rs‘’s module doc for the full provenance extraction against Yarpiz’s ‘tlbo.m‘ – explicitly labeled third-party, not Rao’s own code – the per-learner teaching-factor finding, the unconditionally-distinct partner-selection finding, the min_pop adjustment from 3 down to 2, and the "parameter-free" framing’s Črepinšek/Liu/Mernik (2012) counterpoint) as JSON, ready to pass to ‘sz_solve_bbob()‘.*

Description

Builds a Teaching-Learning-Based Optimization spec (Rao, Savsani & Vakharia 2011, Computer-Aided Design – a labeled metaphor preset, and sezgi's FIRST multi-stage preset: two '[[stages]]' (teacher, then learner) run in sequence every generation. See 'crates/components/src/tlbo.rs''s module doc for the full provenance extraction against Yarpiz's 'tlbo.m' – explicitly labeled third-party, not Rao's own code – the per-learner teaching-factor finding, the unconditionally-distinct partner-selection finding, the min_pop adjustment from 3 down to 2, and the "parameter-free" framing's Črepinšek/Liu/Mernik (2012) counterpoint) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_tlbo(pop_size, budget)
```

Arguments

pop_size	Population size (class size). Canonical is 30.
budget	Evaluation budget. A full generation costs '2 * pop_size' evaluations (both stages evaluate).

Value

A character scalar with the algorithm spec as JSON.

`sz_preset_woa` *Builds a Whale Optimization Algorithm spec (Mirjalili & Lewis 2016 – a labeled metaphor preset, see 'crates/components/src/woa.rs''s module doc for the tier note and citations) as JSON, ready to pass to 'sz_solve_bbob()'.*

Description

Builds a Whale Optimization Algorithm spec (Mirjalili & Lewis 2016 – a labeled metaphor preset, see 'crates/components/src/woa.rs''s module doc for the tier note and citations) as JSON, ready to pass to 'sz_solve_bbob()'.

Usage

```
sz_preset_woa(pop_size, budget)
```

Arguments

pop_size	Population size (school size). Canonical is 30.
budget	Evaluation budget.

Value

A character scalar with the algorithm spec as JSON.

`sz_read_ioh_records` *Reconstructs 'RunRecord's from an on-disk IOH archive at 'log_root' (as written by 'sz_run_experiment(..., log_dir = ...)'), one record per '(run, budget)' pair – see 'sezgi_bench::ioh_records''s doc comment for the exact 'best_f'/'evals_used' semantics and the curtailed-view-vs- independent-run distinction for budgets smaller than a run's logged budget.*

Description

Returns the SAME data.frame shape 'sz_run_experiment()' returns (via ['records_to_data_frame']), so 'sz_results_matrix()'/ 'sz_per_budget_packages()' accept it unchanged.

Usage

```
sz_read_ioh_records(log_root, budgets)
```

Arguments

`log_root`	Path to the IOH archive directory (as passed to 'sz_run_experiment(..., log_dir = ...)').
`budgets`	Numeric vector of evaluation budgets to reconstruct records at.

Value

A data.frame with the same columns as 'sz_run_experiment()'.

`sz_results_matrix` *Build a 'sezgi_stats'-shaped results matrix for one 'budget' from a 'sz_run_experiment()' data.frame.*

Description

Mirrors py-sezgi's 'results_matrix()'.

Usage

```
sz_results_matrix(df, budget, aggregate = "mean")
```

Arguments

df	A data.frame as returned by ‘sz_run_experiment()‘ (columns ‘algo‘, ‘fid‘, ‘dim‘, ‘instance‘, ‘seed‘, ‘budget‘, ‘suite‘, ‘best_f‘, ‘f_opt‘, ‘evals‘). ‘suite‘ is optional – a data.frame without it (e.g. from before this column existed) is treated as all-BBOB.
budget	Only rows with this budget are used.
aggregate	How to combine a (problem, algorithm) cell’s per-seed gaps (‘best_f - f_opt‘) into one number: ‘"mean"‘ or ‘"median"‘.

Value

A named list with ‘algo_names‘ (character vector), ‘problem_labels‘ (character vector, ‘f{fid}d{dim}i{instance}‘ for BBOB rows, ‘{short}-f{fid}d{dim}i{instance}‘ for any other suite, ordered by first appearance in ‘df‘), and ‘matrix‘ (numeric matrix, rows = problems, columns = algorithms; ‘matrix[i, j]‘ is the aggregated gap of ‘algo_names[j]‘ on ‘problem_labels[i]‘). Errors if a (problem, algorithm) pair present for one algorithm/problem is missing for another at this budget (an incomplete experiment).

---

sz_run_experiment *Run a sezgi experiment spec across algorithms x problems x instances x seeds x budgets.*

Description

Mirrors py-sezgi’s ‘run_experiment()‘: ‘journal = NULL‘ runs directly (in parallel or sequentially, per ‘parallel‘); a non-‘NULL‘ ‘journal‘ path runs through the checkpoint executor, which resumes from – and appends to – an existing journal file at that path.

Usage

```
sz_run_experiment(
  spec_toml,
  journal = NULL,
  parallel = TRUE,
  threads = NULL,
  log_dir = NULL
)
```

Arguments

spec_toml	Experiment spec as TOML. Schema (see ‘crates/bench/src/experiment.rs‘): top-level ‘name‘ (string), ‘seeds‘ (integer array), ‘budgets‘ (integer array); one or more ‘[[algorithms]]‘ tables, each with a ‘name‘ and either ‘preset = { kind = "...", pop_size = ... }‘ (pop_size optional/ignored for dim-linked presets: ‘lshade‘, ‘cmaes_ipop‘, ‘nelder_mead‘, ‘sa‘) or ‘spec_toml = "..."‘; one or more ‘[[problems]]‘ tables, each with ‘suite = "bbob"‘, ‘fid‘ (integer), ‘dim‘ (integer), ‘instances‘ (integer array).

journal	Optional path to a checkpoint journal file (JSONL). When given, results are loaded from and appended to this file, and re-running with the same file resumes without re-running completed cells.
parallel	Whether to run in parallel via rayon (default 'TRUE'). Results are bit-identical to the sequential ('FALSE') run.
threads	Optional rayon thread-pool size ('NULL' uses rayon's default).
log_dir	Optional directory: when given, every run this call actually EXECUTES is also logged in IOH-profiler format under that directory (a run resumed from 'journal' was executed in a PRIOR call and is never re-logged). Read back with 'sz_read_ioh_records()'.

Value

A data.frame with one row per run and columns 'algo', 'fid', 'dim', 'instance', 'seed', 'budget', 'suite', 'best_f', 'f_opt', 'evals'.

Examples

```
spec_toml <- '
name = "quick"
seeds = [1]
budgets = [100]

[[algorithms]]
name = "de1"
preset = { kind = "de_rand_1", pop_size = 10 }

[[problems]]
suite = "bbob"
fid = 1
dim = 2
instances = [1]
'
df <- sz_run_experiment(spec_toml)
df$evals
```

---

sz_solve_bbob *Runs an algorithm spec on a BBOB problem and returns the result.*

Description

Runs an algorithm spec on a BBOB problem and returns the result.

Usage

```
sz_solve_bbob(spec_json, fid, dim, instance, master_seed, run_id)
```

Arguments

spec_json	Algorithm spec as JSON (e.g. from ‘sz_preset_de_rand_1()‘).
fid	BBOB function id (>= 1).
dim	Problem dimension (>= 1).
instance	BBOB instance id (>= 1).
master_seed	Master RNG seed.
run_id	Run id (mixed into the seed for independent replicate streams).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ (double vector) – the best EVALUATED point (paired with ‘best_f‘). For most algorithms this always lies within the declared domain. It is not guaranteed to for every algorithm: some (e.g. HHO) charge raw, pre-boundary-repair trial points against the budget before boundary repair, and such a point can become the reported best if it happens to be the run’s own minimum.

Examples

```
spec <- sz_preset_de_rand_1(pop_size = 10, budget = 100)
r <- sz_solve_bbob(spec, fid = 1L, dim = 2L, instance = 1L, master_seed = 1, run_id = 0)
r$evals
```

---

sz_solve_cat_match

Runs an algorithm spec on [‘CatMatch‘] (a Categorical-block Hamming- distance-to-target matching problem; ‘sezgi_problems::diagnostics::CatMatch‘) and returns the result – M3-8 Task 10, mirroring ‘sz_solve_onemax‘ exactly. Pairs with ‘sz_preset_ga_cat(...)‘. Diagnostic only, see ‘CatMatch‘’s own module doc.

Description

Runs an algorithm spec on [‘CatMatch‘] (a Categorical-block Hamming- distance-to-target matching problem; ‘sezgi_problems::diagnostics::CatMatch‘) and returns the result – M3-8 Task 10, mirroring ‘sz_solve_onemax‘ exactly. Pairs with ‘sz_preset_ga_cat(...)‘. Diagnostic only, see ‘CatMatch‘’s own module doc.

Usage

```
sz_solve_cat_match(spec_json, k, n, seed, master_seed, run_id)
```

Arguments

spec_json	Algorithm spec as JSON (e.g. from ‘sz_preset_ga_cat()‘).
k	Category count per gene (double, cast to ‘u32‘).
n	Length of the single ‘Block::Categorical‘ (double, cast to ‘usize‘).
seed	Master seed the target category vector is drawn from (double, cast to ‘u64‘) – UNLIKE ‘sz_solve_int_quadratic‘’s ‘lo‘/‘hi‘, this is an explicit caller-supplied construction parameter, not derived.
master_seed	Master RNG seed for the solve itself.
run_id	Run id (mixed into the seed for independent replicate streams).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ (an INTEGER vector of category INDICES ‘0..k‘ – see ‘genotype_to_r‘’s doc).

Errors A savvy error for any [‘sezgi_core::spec‘] parse error, or any [‘sezgi_core::engine‘] run error.

`sz_solve_cec2014` *Runs an algorithm spec on a CEC 2014 (Liang, Qu & Suganthan 2013) function via [‘Cec2014::new‘] and returns the result – M3-6 Task 10, mirroring ‘sz_solve_cec2022‘ exactly (‘Engine::from_spec‘ + ‘engine.run‘ + result conversion): same ‘best_f‘/‘evals‘/‘best_x‘ shape. See [‘Cec2014::new‘]’s own doc for the exact ‘fid‘/‘dim‘ domain.*

Description

Runs an algorithm spec on a CEC 2014 (Liang, Qu & Suganthan 2013) function via [‘Cec2014::new‘] and returns the result – M3-6 Task 10, mirroring ‘sz_solve_cec2022‘ exactly (‘Engine::from_spec‘ + ‘engine.run‘ + result conversion): same ‘best_f‘/‘evals‘/‘best_x‘ shape. See [‘Cec2014::new‘]’s own doc for the exact ‘fid‘/‘dim‘ domain.

Usage

```
sz_solve_cec2014(spec_json, fid, dim, master_seed, run_id)
```

Arguments

spec_json	Algorithm spec as JSON (e.g. from ‘sz_preset_shade()‘).
fid	CEC 2014 function id, ‘1..=30‘ (double, cast to ‘u32‘).
dim	Problem dimension, one of ‘10‘, ‘30‘ (double, cast to ‘usize‘).
master_seed	Master RNG seed.
run_id	Run id (mixed into the seed for independent replicate streams).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ (double vector) – the best EVALUATED point (paired with ‘best_f‘). Same caveat as ‘sz_solve_bbob()‘: not every algorithm’s reported best is guaranteed to lie within the declared domain.

Errors A savvy error for ‘fid‘ outside ‘1..=30‘, ‘dim‘ outside ‘{10,30}‘, any [‘sezgi_core::spec‘] parse error, or any [‘sezgi_core::engine‘] run error.

`sz_solve_cec2017` *Runs an algorithm spec on a CEC 2017 (Awad, Ali, Liang, Qu & Suganthan 2016) function via [‘Cec2017::new‘] and returns the result – M3-6 Task 10, mirroring ‘sz_solve_cec2014‘ exactly. See [‘Cec2017::new‘]’s own doc for the exact ‘fid‘/‘dim‘ domain.*

Description

Runs an algorithm spec on a CEC 2017 (Awad, Ali, Liang, Qu & Suganthan 2016) function via [‘Cec2017::new‘] and returns the result – M3-6 Task 10, mirroring ‘sz_solve_cec2014‘ exactly. See [‘Cec2017::new‘]’s own doc for the exact ‘fid‘/‘dim‘ domain.

Usage

```
sz_solve_cec2017(spec_json, fid, dim, master_seed, run_id)
```

Arguments

`spec_json`	Algorithm spec as JSON (e.g. from ‘sz_preset_shade()‘).
`fid`	CEC 2017 function id, ‘1‘ or ‘3..=30‘ (double, cast to ‘u32‘); ‘fid = 2‘ was officially withdrawn from the suite.
`dim`	Problem dimension, one of ‘10‘, ‘30‘ (double, cast to ‘usize‘).
`master_seed`	Master RNG seed.
`run_id`	Run id (mixed into the seed for independent replicate streams).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ (double vector) – the best EVALUATED point (paired with ‘best_f‘). Same caveat as ‘sz_solve_bbob()‘: not every algorithm’s reported best is guaranteed to lie within the declared domain.

Errors A savvy error for ‘fid‘ outside ‘{1} union {3..=30}‘, ‘dim‘ outside ‘{10,30}‘, any [‘sezgi_core::spec‘] parse error, or any [‘sezgi_core::engine‘] run error. ‘fid = 2‘ raises a dedicated error – the Rust [‘sezgi_problems::Cec2017Error::Withdrawn‘] message is surfaced VERBATIM.

sz_solve_cec2022	*Runs an algorithm spec on a CEC 2022 (Kumar, Price, Mohamed, Hadi & Suganthan 2021) function via [‘Cec2022::new‘] and returns the result – added later than the rest of the CEC surface, closing the gap where r-sezgi previously bound only direct evaluation (‘sz_cec2022_evaluate‘/‘sz_cec2022_f_star‘), with no ‘solve()‘-integrated path, unlike py-sezgi’s ‘sezgi.problems.cec2022(...)‘ + ‘sezgi.solve()‘. Mirrors ‘sz_solve_bbob‘/‘sz_solve_tsp‘ exactly (‘Engine::from_spec‘ + ‘engine.run‘ + result conversion): same ‘best_f‘/‘evals‘/‘best_x‘ shape, not py-sezgi’s own ‘solve()‘ dict shape (‘best_f‘/‘best_x‘/‘evals_used‘/ ‘iterations‘) – the established r-sezgi ‘sz_solve_*‘ convention governs here too. See ‘Cec2022::new‘’s own doc for the exact ‘fid‘/‘dim‘ domain.*

Description

Runs an algorithm spec on a CEC 2022 (Kumar, Price, Mohamed, Hadi & Suganthan 2021) function via [‘Cec2022::new‘] and returns the result – added later than the rest of the CEC surface, closing the gap where r-sezgi previously bound only direct evaluation (‘sz_cec2022_evaluate‘/‘sz_cec2022_f_star‘), with no ‘solve()‘-integrated path, unlike py-sezgi’s ‘sezgi.problems.cec2022(...)‘ + ‘sezgi.solve()‘. Mirrors ‘sz_solve_bbob‘/‘sz_solve_tsp‘ exactly (‘Engine::from_spec‘ + ‘engine.run‘ + result conversion): same ‘best_f‘/‘evals‘/‘best_x‘ shape, not py-sezgi’s own ‘solve()‘ dict shape (‘best_f‘/‘best_x‘/‘evals_used‘/ ‘iterations‘) – the established r-sezgi ‘sz_solve_*‘ convention governs here too. See ‘Cec2022::new‘’s own doc for the exact ‘fid‘/‘dim‘ domain.

Usage

```
sz_solve_cec2022(spec_json, fid, dim, master_seed, run_id)
```

Arguments

spec_json	Algorithm spec as JSON (e.g. from ‘sz_preset_shade()‘).
fid	CEC 2022 function id, ‘1..=12‘ (double, cast to ‘u32‘).
dim	Problem dimension, one of ‘2‘, ‘10‘, ‘20‘ (double, cast to ‘usize‘); ‘dim = 2‘ is additionally rejected for a hybrid function (‘fid‘ 6-8).
master_seed	Master RNG seed.
run_id	Run id (mixed into the seed for independent replicate streams).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ (double vector) – the best EVALUATED point (paired with ‘best_f‘). Same caveat as ‘sz_solve_bbob()‘: not every algorithm’s reported best is guaranteed to lie within the declared domain.

Errors A savvy error for ‘fid‘ outside ‘1..=12‘, ‘dim‘ outside ‘{2,10,20}‘, ‘dim = 2‘ for a hybrid function, any [‘sezgi_core::spec‘] parse error, or any [‘sezgi_core::engine‘] run error.

sz_solve_int_quadratic

Runs an algorithm spec on [‘IntQuadratic‘] (an Int-block quadratic bowl around a fixed, deterministically-derived target; ‘sezgi_problems::diagnostics::IntQuadratic‘) and returns the result – M3-8 Task 10, mirroring ‘sz_solve_onemax‘ exactly. Pairs with ‘sz_preset_ga_int(...)‘. Diagnostic only, see ‘IntQuadratic‘’s own module doc.

Description

Runs an algorithm spec on [‘IntQuadratic‘] (an Int-block quadratic bowl around a fixed, deterministically-derived target; ‘sezgi_problems::diagnostics::IntQuadratic‘) and returns the result – M3-8 Task 10, mirroring ‘sz_solve_onemax‘ exactly. Pairs with ‘sz_preset_ga_int(...)‘. Diagnostic only, see ‘IntQuadratic‘’s own module doc.

Usage

```
sz_solve_int_quadratic(spec_json, lo, hi, n, master_seed, run_id)
```

Arguments

spec_json	Algorithm spec as JSON (e.g. from ‘sz_preset_ga_int()‘).
lo	Inclusive lower bound of the single ‘Block::Int‘ (double, cast to ‘i64‘; may be negative).
hi	Inclusive upper bound of the single ‘Block::Int‘ (double, cast to ‘i64‘; may be negative). Must be ‘> lo‘.
n	Length of the single ‘Block::Int‘ (double, cast to ‘usize‘).
master_seed	Master RNG seed.
run_id	Run id (mixed into the seed for independent replicate streams).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ (an INTEGER vector – see ‘genotype_to_r‘’s doc).

Errors A savvy error if ‘lo >= hi‘, for any [‘sezgi_core::spec‘] parse error, or any [‘sezgi_core::engine‘] run error.

sz_solve_mixed_diagnostic

Runs an algorithm spec on [‘MixedDiagnostic‘] (this file’s own Float+Int+Categorical+Binary mixed-space scaffold problem, see its own doc) and returns the result – M3-8 Task 10. Added SOLELY so ‘gen/compound‘ (Task 5) is reachable end to end through the NORMAL R solve path, proven with a mixed-space ‘AlgorithmSpec‘ authored as TOML (this task’s own test) – UNLIKE its three siblings above (which take ‘spec_json‘, pairing with ‘sz_preset_ga_bin/ga_int/ga_cat‘’s own ‘.to_json()‘ presets), this function takes ‘spec_toml‘ directly and parses it via [‘AlgorithmSpec::from_toml‘], the SAME entry point ‘sz_run_experiment_raw‘’s ‘ExperimentSpec::from_toml‘ already establishes the "hand a raw TOML document straight to the Rust core" convention for (‘experiment.rs‘) – no R-side TOML library exists or is needed (r-sezgi has none in ‘DESCRIPTION‘’s ‘Suggests‘; unlike py-sezgi’s test, which parses TOML with the stdlib’s own ‘tomllib‘ into a dict before handing it to ‘solve()‘, R has no such stdlib module, so parsing happens in Rust instead – ‘AlgorithmSpec::from_toml‘/‘::from_json‘ are just two serializations of the identical schema, so this is not a private shortcut, only a different serialization entry point already used elsewhere in this same file’s crate). Mirrors ‘sz_solve_onemax‘ otherwise, EXCEPT ‘run_id‘ is dropped (fixed to ‘0‘ internally) rather than taken as an explicit parameter – with ‘spec_toml‘ this function already sits at 7 R-facing parameters; adding ‘run_id‘ would push it to 8 and trip this workspace’s ‘clippy::too_many_arguments‘ gate (threshold 7, this file’s ONE pre-existing exception is ‘sz_preset_es_mu_plus_lambda_raw‘, not to be joined by a second). ‘run_id = 0‘ matches how this scaffold is actually exercised (this task’s own TOML test, mirroring py-sezgi’s ‘test_gen_compound_mixed_space_toml_spec_ solves_end_to_end‘, calls ‘solve(spec, problem, master_seed=42)‘ with no ‘run_id‘ override either – ‘solve()‘’s own Python signature defaults ‘run_id=0‘). Test scaffolding only – NOT one of Task 5’s brief-pinned diagnostics, and (unlike onemax/int_quadratic/cat_match) has no verified target: this file’s own convention never surfaces ‘Problem::optimum()‘ in a result anyway (see ‘sz_solve_bbob‘’s own ‘best_f‘/‘evals‘/‘best_x‘ shape), so that caveat needs no separate plumbing here.

Description

Runs an algorithm spec on [‘MixedDiagnostic‘] (this file’s own Float+Int+Categorical+Binary mixed-space scaffold problem, see its own doc) and returns the result – M3-8 Task 10. Added SOLELY so ‘gen/compound‘ (Task 5) is reachable end to end through the NORMAL R solve path, proven with a mixed-space ‘AlgorithmSpec‘ authored as TOML (this task’s own test) – UNLIKE its three siblings above (which take ‘spec_json‘, pairing with ‘sz_preset_ga_bin/ga_int/ga_cat‘’s own ‘.to_json()‘ presets), this function takes ‘spec_toml‘ directly and parses it via [‘AlgorithmSpec::from_toml‘], the

SAME entry point ‘sz_run_experiment_raw‘’s ‘ExperimentSpec::from_toml‘ already establishes the "hand a raw TOML document straight to the Rust core" convention for (‘experiment.rs‘) – no R-side TOML library exists or is needed (r-sezgi has none in ‘DESCRIPTION‘’s ‘Suggests‘; unlike py-sezgi’s test, which parses TOML with the stdlib’s own ‘tomllib‘ into a dict before handing it to ‘solve()‘, R has no such stdlib module, so parsing happens in Rust instead – ‘AlgorithmSpec::from_toml‘/‘::from_json‘ are just two serializations of the identical schema, so this is not a private shortcut, only a different serialization entry point already used elsewhere in this same file’s crate). Mirrors ‘sz_solve_onemax‘ otherwise, EXCEPT ‘run_id‘ is dropped (fixed to ‘0‘ internally) rather than taken as an explicit parameter – with ‘spec_toml‘ this function already sits at 7 R-facing parameters; adding ‘run_id‘ would push it to 8 and trip this workspace’s ‘clippy::too_many_arguments‘ gate (threshold 7, this file’s ONE pre-existing exception is ‘sz_preset_es_mu_plus_lambda_ra not to be joined by a second). ‘run_id = 0‘ matches how this scaffold is actually exercised (this task’s own TOML test, mirroring py-sezgi’s ‘test_gen_compound_mixed_space_toml_spec_ solves_end_to_end‘, calls ‘solve(spec, problem, master_seed=42)‘ with no ‘run_id‘ override either – ‘solve()‘’s own Python signature defaults ‘run_id=0‘). Test scaffolding only – NOT one of Task 5’s brief-pinned diagnostics, and (unlike onemax/int_quadratic/cat_match) has no verified target: this file’s own convention never surfaces ‘Problem::optimum()‘ in a result anyway (see ‘sz_solve_bbob‘’s own ‘best_f‘/‘evals‘/‘best_x‘ shape), so that caveat needs no separate plumbing here.

Usage

```
sz_solve_mixed_diagnostic(
  spec_toml,
  n_float,
  n_int,
  k_cat,
  n_cat,
  n_bin,
  master_seed
)
```

Arguments

spec_toml	Algorithm spec as TOML text (e.g. a mixed-space ‘gen/compound‘ document).
n_float	Length of the ‘Block::Float{-5,5,..}‘ block (double, cast to ‘usize‘).
n_int	Length of the ‘Block::Int{-5,5,..}‘ block (double, cast to ‘usize‘).
k_cat	Category count per gene of the ‘Block::Categorical‘ block (double, cast to ‘u32‘).
n_cat	Length of the ‘Block::Categorical‘ block (double, cast to ‘usize‘).
n_bin	Length of the ‘Block::Binary‘ block (double, cast to ‘usize‘).
master_seed	Master RNG seed. ‘run_id‘ is fixed to ‘0‘ (see this function’s own doc for why it is not a parameter here).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ – a MULTI-block genotype, surfaced as an unnamed list of 4 per-block vectors in ‘SearchSpace::blocks()‘ order (Float numeric, Int integer, Categorical integer, Binary logical – see ‘genotype_to_r‘’s doc).

Errors A savvy error for any [‘sezgi_core::spec‘] parse error, or any [‘sezgi_core::engine‘] run error.

---

sz_solve_onemax	*Runs an algorithm spec on [‘OneMax‘] (Goldberg 1989’s classic Binary-block GA diagnostic; ‘sezgi_problems::diagnostics::OneMax‘) and returns the result – M3-8 Task 10, mirroring ‘sz_solve_tsp‘/ ‘sz_solve_cec2022‘ exactly (‘Engine::from_spec‘ + ‘engine.run‘), except ‘best_x‘ is now typed via [‘genotype_to_r‘] rather than assumed ‘Float‘ (see that helper’s own doc for the full type-mapping table). Pairs with ‘sz_preset_ga_bin(...)‘. Diagnostic only – not a benchmark, see ‘OneMax‘’s own module doc.*

Description

Runs an algorithm spec on [‘OneMax‘] (Goldberg 1989’s classic Binary-block GA diagnostic; ‘sezgi_problems::diagnostics::OneMax‘) and returns the result – M3-8 Task 10, mirroring ‘sz_solve_tsp‘/ ‘sz_solve_cec2022‘ exactly (‘Engine::from_spec‘ + ‘engine.run‘), except ‘best_x‘ is now typed via [‘genotype_to_r‘] rather than assumed ‘Float‘ (see that helper’s own doc for the full type-mapping table). Pairs with ‘sz_preset_ga_bin(...)‘. Diagnostic only – not a benchmark, see ‘OneMax‘’s own module doc.

Usage

```
sz_solve_onemax(spec_json, n_bits, master_seed, run_id)
```

Arguments

spec_json	Algorithm spec as JSON (e.g. from ‘sz_preset_ga_bin()‘).
n_bits	Length of the single ‘Block::Binary‘ (double, cast to ‘usize‘).
master_seed	Master RNG seed.
run_id	Run id (mixed into the seed for independent replicate streams).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ (a LOGICAL vector, one per bit – see [‘genotype_to_r‘]’s doc).

Errors A savvy error for any [‘sezgi_core::spec‘] parse error, or any [‘sezgi_core::engine‘] run error.

sz_solve_tsp	*Runs an algorithm spec on a TSPLIB VENDORED instance (‘"berlin52"‘, ‘"eil51"‘, ‘"st70"‘ – via [‘Tsp::vendored‘]; UNLIKE ‘sz_tsp_load()‘/ ‘sz_tsp_tour_length()‘ in ‘problems.rs‘, raw TSPLIB text is not accepted here – mirrors py-sezgi’s ‘sezgi.problems.tsp(name)‘, which is likewise vendored-only) and returns the result. Same output shape as ‘sz_solve_bbob()‘ (‘best_f‘/‘evals‘/‘best_x‘), not py-sezgi’s own ‘solve()‘ dict shape (‘best_f‘/‘best_x‘/‘evals_used‘/‘iterations‘) – the established r-sezgi ‘sz_solve_*‘ convention governs here, not py-sezgi’s key names (see ‘problems.rs‘’s module doc, "Index-convention decision", for the general 1-based-vs-0-based rule this function’s ‘best_x‘ also follows).*

Description

Runs an algorithm spec on a TSPLIB VENDORED instance (‘"berlin52"‘, ‘"eil51"‘, ‘"st70"‘ – via [‘Tsp::vendored‘]; UNLIKE ‘sz_tsp_load()‘/ ‘sz_tsp_tour_length()‘ in ‘problems.rs‘, raw TSPLIB text is not accepted here – mirrors py-sezgi’s ‘sezgi.problems.tsp(name)‘, which is likewise vendored-only) and returns the result. Same output shape as ‘sz_solve_bbob()‘ (‘best_f‘/‘evals‘/‘best_x‘), not py-sezgi’s own ‘solve()‘ dict shape (‘best_f‘/‘best_x‘/‘evals_used‘/‘iterations‘) – the established r-sezgi ‘sz_solve_*‘ convention governs here, not py-sezgi’s key names (see ‘problems.rs‘’s module doc, "Index-convention decision", for the general 1-based-vs-0-based rule this function’s ‘best_x‘ also follows).

Usage

```
sz_solve_tsp(spec_json, name, master_seed, run_id)
```

Arguments

spec_json	Algorithm spec as JSON (e.g. from ‘sz_preset_ga_perm()‘).
name	A vendored TSPLIB instance name.
master_seed	Master RNG seed.
run_id	Run id (mixed into the seed for independent replicate streams).

Value

A named list with ‘best_f‘ (double), ‘evals‘ (double), and ‘best_x‘ (double vector – a 1-based permutation of ‘1:n_cities‘, the best EVALUATED tour, paired with ‘best_f‘).

Errors A savvy error if ‘name‘ is not one of the three vendored instances, for any [‘sezgi_core::spec‘] parse error, or any [‘sezgi_core::engine‘] run error.

sz_space *Composes one or more blocks into a search space, in the given order.*

Description

Mirrors ‘sezgi.Space‘ (‘py-sezgi/python/sezgi/spaces.py‘) exactly: at least one block is required, and every argument must inherit ‘"sz_block"‘ (i.e. be built by [sz_float()], [sz_int()], [sz_categorical()], [sz_binary()], or [sz_permutation()]) – these are the only two validation rules ‘Space.__init__‘ itself performs.

Usage

```
sz_space(...)
```

Arguments

...	One or more ‘sz_block‘ objects, in space order.

Value

An object of class ‘"sz_space"‘ with element ‘blocks‘ (a ‘list‘ of the given blocks, in order).

Examples

```
sz_space(sz_float(-5, 5, 3), sz_int(0L, 10L, 2))
```

sz_stats_bayesian_signed_rank *Bayesian signed-rank test with a region of practical equivalence (ROPE).*

Description

Mirrors py-sezgi’s ‘stats_bayesian_signed_rank()‘. Dirichlet-weighted Monte Carlo scheme (Benavoli et al. 2017 simplification); see ‘crates/stats/src/bayesian.rs‘ for the full algorithm and determinism guarantees. The same ‘(a, b, rope, samples, seed)‘ always produces a bit-identical result – R calls the exact same seeded Rust core as Python and Rust.

Usage

```
sz_stats_bayesian_signed_rank(a, b, rope = 0, samples = 20000, seed = 1)
```

Arguments

a	Numeric vector.
b	Numeric vector, same length as ‘a‘.
rope	Region of practical equivalence half-width (>= 0). Default 0.
samples	Number of Monte Carlo samples. Default 20000.
seed	Master RNG seed. Default 1.

Value

A named list with ‘p_left‘, ‘p_rope‘, ‘p_right‘.

---

sz_stats_cliffs_delta *Cliff's delta effect size for two independent (unpaired) samples.*

Description

Cliff's delta effect size for two independent (unpaired) samples.

Usage

```
sz_stats_cliffs_delta(a, b)
```

Arguments

a	Numeric vector.
b	Numeric vector.

Value

A numeric scalar in ‘[-1, 1]‘.

---

sz_stats_cliffs_magnitude *Qualitative magnitude label for a Cliff's delta value (Romano et al. 2006 thresholds).*

Description

Qualitative magnitude label for a Cliff's delta value (Romano et al. 2006 thresholds).

Usage

```
sz_stats_cliffs_magnitude(delta)
```

Arguments

delta	A Cliff's delta value (as returned by 'sz_stats_cliffs_delta()').

Value

A character scalar: one of '"negligible"', '"small"', '"medium"', '"large"'.

---

sz_stats_friedman *Runs the Friedman test on a results matrix.*

Description

Runs the Friedman test on a results matrix.

Usage

```
sz_stats_friedman(m)
```

Arguments

m	A numeric matrix, rows = problems, columns = algorithms, lower is better (e.g. built with 'rbind()' or 'matrix()').

Value

A named list with 'statistic', 'p_value', 'mean_ranks' (mirrors py-sezgi's 'stats_friedman()' dict keys exactly).

---

sz_stats_paper_package *Comprehensive statistical analysis package for algorithm comparison.*

Description

Mirrors py-sezgi's 'stats_paper_package()': Friedman test, Nemenyi critical difference, all-pairs Wilcoxon signed-rank with Holm adjustment, Cliff's delta, Bayesian signed-rank, Plackett-Luce ranking, and LaTeX summary/pairwise-comparison tables. See 'crates/stats/src/report.rs'.

Usage

```
sz_stats_paper_package(
  algo_names,
  problem_names,
  m,
  rope = 0,
  samples = 20000,
  seed = 1
)
```

Arguments

algo_names	Character vector of algorithm names.
problem_names	Character vector of problem names.
m	A numeric matrix, rows = problems (‘length(problem_names)‘), columns = algorithms (‘length(algo_names)‘), lower is better.
rope	Region of practical equivalence half-width (>= 0) for the Bayesian signed-rank test. Default 0.
samples	Number of Monte Carlo samples per pair for the Bayesian signed-rank test. Default 20000.
seed	Master RNG seed; per-pair seeds are ‘seed + pair_index‘ (wrapping). Default 1.

Value

A named list with ‘friedman‘, ‘nemenyi_cd‘, ‘pairwise_wilcoxon_holm‘, ‘cliffs‘, ‘bayes‘, ‘plackett_luce‘, ‘latex_summary‘, ‘latex_tests‘ – see ‘sz_stats_paper_package_raw()‘ for the full field-by-field shape.

---

sz_stats_plackett_luce

Plackett-Luce maximum-likelihood ranking (Hunter 2004 MM algorithm; see ‘sezgi_stats::plackett_luce‘).

Description

Plackett-Luce maximum-likelihood ranking (Hunter 2004 MM algorithm; see ‘sezgi_stats::plackett_luce‘).

Usage

```
sz_stats_plackett_luce(rankings)
```

Arguments

rankings	A ‘list‘ of integer (or integer-valued numeric) vectors, each a full ranking of the same ‘k‘ items as **1-based item ids** (R’s natural indexing; e.g. ‘list(c(1, 2, 3), c(2, 1, 3))‘ for k = 3 items, best first). Converted to 0-based indices before calling the Rust core, which expects a permutation of ‘0..k‘.

Value

A named list with ‘worths‘, ‘p_best‘, ‘iterations‘ (mirrors py-sezgi’s ‘stats_plackett_luce()‘ dict keys exactly).

`sz_stats_wilcoxon` *Wilcoxon signed-rank test for two paired samples (Pratt zero-handling and tie correction; see ‘sezgi_stats::wilcoxon_signed_rank‘).*

Description

Wilcoxon signed-rank test for two paired samples (Pratt zero-handling and tie correction; see ‘sezgi_stats::wilcoxon_signed_rank‘).

Usage

```
sz_stats_wilcoxon(a, b)
```

Arguments

`a`	Numeric vector.
`b`	Numeric vector, same length as ‘a‘.

Value

A named list with ‘w_statistic‘, ‘z‘, ‘p_value‘, ‘n_effective‘, ‘method‘ (‘"exact"‘ or ‘"normal_approx"‘; mirrors py-sezgi’s ‘stats_wilcoxon()‘ dict keys exactly). ‘"exact"‘ is used when ‘n_effective <= 25‘ and there are no zero differences or tied ‘|d|‘ ranks; see ‘sezgi_stats::wilcoxon_signed_rank‘’s doc comment for the full eligibility rule (the exact p-value formula is semver-pinned).

Examples

```
sz_stats_wilcoxon(c(1, 2, 3, 4, 5), c(2, 1, 4, 3, 6))
```

`sz_tsp_load` *Loads a TSPLIB ‘EUC_2D‘ instance, either a vendored instance name (‘"berlin52"‘, ‘"eil51"‘, ‘"st70"‘) or raw TSPLIB file text (see [‘load_tsp‘]).*

Description

Loads a TSPLIB ‘EUC_2D‘ instance, either a vendored instance name (‘"berlin52"‘, ‘"eil51"‘, ‘"st70"‘) or raw TSPLIB file text (see [‘load_tsp‘]).

Usage

```
sz_tsp_load(name_or_text)
```

Arguments

`name_or_text`	A vendored instance name, or raw TSPLIB ‘.tsp‘ file text.

Value

A named list with ‘name‘ (character scalar), ‘n_cities‘ (double), ‘coords‘ (an ‘n_cities‘ x 2 numeric matrix, row ‘i‘ is city ‘i‘’s ‘(x, y)‘ coordinate pair – see this module’s own doc, "Index-convention decision"), ‘known_optimum‘ (double, or ‘NULL‘ for an instance parsed from raw text rather than a vendored name).

Errors A savvy error for any [‘TspError‘] (unknown vendored name that also fails to parse as TSPLIB text, malformed TSPLIB text, unsupported ‘EDGE_WEIGHT_TYPE‘, ...).

Examples

```
tsp <- sz_tsp_load("berlin52")
tsp$n_cities
```

`sz_tsp_tour_length` *Closed-tour length of a 1-based ‘tour‘ (a permutation of ‘1:n_cities‘ – see this module’s own doc, "Index-convention decision") on the instance named/parsed by ‘name_or_text‘ (see [‘load_tsp‘]), via [‘Tsp::evaluate_batch‘]’s ‘nint‘-rounded ‘EUC_2D‘ sum (‘tsp.rs‘’s module doc).*

Description

UNLIKE ‘Tsp::evaluate_batch‘ itself (which returns ‘f64::INFINITY‘ for a malformed genotype, since genotype validity is normally the engine’s ‘SearchSpace::validate‘ job, not ‘Tsp‘’s own) – this binding validates ‘tour‘ itself (exact length, every entry in ‘1..=n_cities‘, no repeats) and raises a precise error instead, since a ‘tour‘ coming directly from R has no engine-side validation gate in front of it.

Usage

```
sz_tsp_tour_length(name_or_text, tour)
```

Arguments

`name_or_text`	A vendored instance name, or raw TSPLIB ‘.tsp‘ file text (same as ‘sz_tsp_load‘).
`tour`	A numeric vector, a permutation of ‘1:n_cities‘ (1-based).

Value

A numeric scalar.

Errors A savvy error for any [‘TspError‘] resolving ‘name_or_text‘, or if ‘tour‘ is not a permutation of ‘1:n_cities‘ (wrong length, an out-of-range entry, or a repeated entry).

Index