ilustrado Documentation

Release 0.4.0

Matthew Evans

Jul 02, 2020

CONTENTS

1 Summary **3**

2 Limitations: **5**
2.1 v0.4 5
2.2 v0.3 5
2.3 ilustrado 6

Python Module Index **15**

Index **17**

CHAPTER

ONE

SUMMARY

ilustrado is a Python package that implements a highly-customisable massively-parallel genetic algorithm for *ab initio* crystal structure prediction (CSP), with a focus on mapping out compositional phase diagrams. The aim of ilustrado was to provide a method of extending CSP results generated with random structure searching (AIRSS) or extrapolating from known chemically-relevant systems (species-swapping/prototyping).

The API is fully-documented and the source code can be found on GitHub. ilustrado makes extensive use of the matador API and interfaces with CASTEP for DFT-level relaxations. Written by Matthew Evans (me388@cam.ac.uk).

By default, fitnesses are evaluated as the distance from a binary or ternary convex hull that is passed as input. Duplicate structures are filtered out post-relaxation based on pair distribution function overlap. The standard mutation operators are implemented: cell and position noise, vacancies, and atom permutation and transmutation. Additionally a Voronoi-based mutation has been implemented:

1. Each of the N atoms of randomly chosen species **A** are removed from the cell.
2. The Voronoi decomposition of the remaining atoms is calculated.
3. K-means clustering is applied to the Voronoi points to split them into $N \pm D$ clusters, where D is a random integer less than $\sqrt{(N)}$.
4. $N \pm D$ atoms of species **A** are added to the cell at the centres of these clusters. The species **A** can be specified by the user or chosen randomly. This mutation is effective when studying materials that have one light, mobile element, for example Li.

Crossover is performed with the standard cut-and-splice method [1] to ensure transferrability over many types of material systems. This method cuts a random fraction from each parent cell and splices them back together; in ilustrado efforts are made to align the cells such that the cutting axes are approximately commensurate before crossover.

Several physical constraints (minimum atomic separations, densities, cell shapes) are applied to the trial structures before relaxation to improve efficiency. In order to maintain population diversity as the simulation progresses, the user can optionally disfavour frequently-visited regions of composition space.

The entrypoint is a Python script that creates an ArtificialSelector object with the desired parameters. Many examples can be found in `examples/`. There are two modes in which to run ilustrado:

- `compute_mode='direct'` involves one ilustrado processes spawning `mpirun` jobs either on local or remote partitions (i.e. either a node list is passed for running on a local cluster, or ilustrado itself is submitted as a HPC job). In this case, the user must manually restart the GA when the job finishes.
- `compute_mode='slurm'` performs the GA in interruptible steps; submitting ilustrado as a job will lead to the submission of many slurm array jobs for the relaxation, and a dependency job that re-runs the ilustrado process to check on the relaxations. In this case, the user only needs to submit one job (tested on 6400 cores/200 nodes without issue).

[1] Deaven, D. M.; Ho, K. M. Molecular Geometry Optimization with a Genetic Algorithm. Phys. Rev. Lett. 1995, 75, 288, 10.1103/PhysRevLett.75.288.

CHAPTER

TWO

LIMITATIONS:

- There are no stopping criteria implemented in ilustrado. Due to the multi-composition nature of the method, it is not easy to define either success or stagnation. Instead, users can perform their own analysis on the output of each generation.
- Due to the way it initialised in our group, generation of random structures is not performed as part of ilustrado. Instead, we typically start from a phase diagram that has already been well-sampled with a random search.
- None of the mutation or crossover operators make use of symmetry.

2.1 v0.4

- Updated matador dependency PyPI versions > 0.9.
- Added `compute_mode="manual"` to just generate structures without relaxing
- Added GitHub CI
- Rescale cell volume after crossover to maintain density
- QoL updates to printing and loading mutations
- Allow arbitrary ASE calculator to be used in relaxations

2.2 v0.3

- New keyword: `sandbagging`. When enabled, fitness penalties will be applied to successive sampling of the same region of composition space. By default, the modifier is a multiplicative factor of 0.95 to all compositions within a hypersphere of radius 0.05.
- `compute_mode='slurm'` that makes use of array jobs for "infinite" horizontal scalability
- improved documentation and examples

2.3 ilustrado

2.3.1 ilustrado package

Subpackages

Submodules

ilustrado.adapt module

This file contains a wrapper for mutation and crossover.

`ilustrado.adapt.`**`adapt`**(*possible_parents*, *mutation_rate*, *crossover_rate*, *mutations=None*, *max_num_mutations=3*, *max_num_atoms=40*, *structure_filter=None*, *minsep_dict=None*, *debug=False*)

Take a list of possible parents and randomly adapt according to given mutation weightings.

Parameters

- **possible_parents** (*list (dict)*) – list of all breeding stock,
- **mutation_rate** (*float*) – rate of mutations relative to crossover,
- **crossover_rate** (*float*) – see *mutation_rate*.

Keyword Arguments

- **mutations** (*list (str)*) – list of desired mutations to choose from (as strings),
- **max_num_mutations** (*int*) – rand(1, this) mutations will be performed,
- **max_num_atoms** (*int*) – any structures with more than this many atoms will be filtered out.
- **structure_filter** (*callable (dict)*) – custom filter to pass to check_feasible.
- **minsep_dict** (*dict*) – dictionary containing element-specific minimum separations, e.g. *{('K', 'K'): 2.5, ('K', 'P'): 2.0}*.

Returns the mutated/newborn structure.

Return type dict

`ilustrado.adapt.`**`check_feasible`**(*mutant*, *parents*, *max_num_atoms*, *structure_filter=None*, *minsep_dict=None*, *debug=False*)

Check if a mutated/newly-born cell is "feasible". Here, feasible means:

- number density within 25% of pre-mutation/birth level,
- no overlapping atoms, parameterised by minsep_dict,
- cell angles between 50 and 130 degrees,
- fewer than max_num_atoms in the cell,
- ensure number of atomic types is maintained,
- any custom filter is obeyed.

Parameters

- **mutant** (*dict*) – matador doc containing new structure.
- **parents** (*list (dict)*) – list of doc(s) containing parent structures.

- **max_num_atoms** (*int*) – any structures with more than this many atoms will be filtered out.

Keyword Arguments

- **structure_filter** (*callable*) – any function that takes a matador document and returns True or False.
- **minsep_dict** (*dict*) – dictionary containing element-specific minimum separations, e.g. {('K', 'K'): 2.5, ('K', 'P'): 2.0}.

Returns True if structure is feasible, else False.

Return type bool

`ilustrado.adapt.`**`minseps_feasible`**(*mutant*, *minsep_dict=None*, *debug=False*)

Check if minimum separations between species of atom are satisfied by mutant.

Parameters

- **mutant** (*dict*) – trial mutated structure
- **minsep_dict** (*dict*) – dictionary containing element-specific minimum separations, e.g. {('K', 'K'): 2.5, ('K', 'P'): 2.0}.

Returns True if minseps are greater than desired value else False.

Return type bool

ilustrado.analysis module

Some assorted analysis functions.

`ilustrado.analysis.`**`fitness_swarm_plot`**(*generations*, *ax=None*, *save=False*)

Make a swarm plot of the fitness of all generations.

`ilustrado.analysis.`**`plot_new_2d_hull`**(*generations*, *hull*, *points=True*, *label_hull=True*, *save=True*)

Add new structures to old ConvexHull plot.

ilustrado.crossover module

This file implements crossover functionality.

`ilustrado.crossover.`**`crossover`**(*parents*, *method='random_slice'*, *debug=False*)

Attempt to create a child structure from two parents structures.

Parameters

- **parents** (*list (dict)*) – list of two parent structures
- **method** (*str*) – currently only 'random_slice'

Returns newborn structure from parents.

Return type dict

`ilustrado.crossover.`**`random_slice`**(*parent_seeds*, *standardize=True*, *supercell=True*, *shift=True*, *debug=False*)

Simple cut-and-splice crossover of two parents.

The overall size of the child can vary between 0.5 and 1.5 the size of the parent structures. Both parent structures are cut and spliced along the same crystallographic axis.

Parameters

- **parents** (*list (dict)*) – parent structures to crossover,
- **standardize** (*bool*) – use spglib to standardize parents pre-crossover,
- **supercell** (*bool*) – make a random supercell to rescale parents,
- **shift** (*bool*) – randomly shift atoms in parents to unbias.

Returns newborn structure from parents.

Return type dict

ilustrado.fitness module

This file implements all notions of fitness.

class ilustrado.fitness.**FitnessCalculator**(*fitness_metric='dummy', fitness_function=None, hull=None, sandbagging=False, debug=False*)

Bases: object

This class calculates the fitnesses of generations, by some global definition of generation-agnostic fitness.

Parameters

- **fitness_metric** (*str*) – either 'dummy', 'hull' or 'hull_test'.
- **fitness_function** (*callable*) – function to operate on numpy array of raw fitness values,
- **hull** (*QueryConvexHull*) – matador hull from which to calculate metastability,
- **sandbagging** (*bool*) – whether or not to "sandbag" particular compositions, i.e. lower a structure's fitness based on the number of nearby phases

evaluate(*generation*)

Assign normalised fitnesses to an entire generation. Normalisation uses the logistic function such that

*fitness = 1 - tanh(2*distance_from_hull)*,

Parameters **generation** (*Generation/list*) – list/iterator over optimised structures,

update_sandbag_multipliers(*generation, modifier=0.95*)

Assign composition penalty based on number of nearby structures. Updates fitness.sandbag_multipliers to a dictionary with chemical concentration as keys and values of fitness penalty.

Parameters **generation** (Generation) – list of optimised structures.

apply_sandbag_multipliers(*generation, locality=0.05*)

Scale the generation's fitness by the sandbag modifier. This updates the 'fitness' key and the 'modifier' key (total scaling) of each document in the generation.

Parameters **generation** (Generation) – list of optimised structures.

Keyword Arguments **locality** (*float*) – tolerance by which two structures are "nearby"

ilustrado.fitness.**default_fitness_function**(*raw, c=50, offset=0.075*)

Default fitness function: logistic function.

Parameters **raw** (*ndarray*) – 1D array of raw fitness values.

Returns 1D array of rescaled fitnesses.

Return type ndarray

ilustrado.generation module

This file implements the Generation class which is used to store each generation of structures, and to evaulate their fitness.

class ilustrado.generation.**Generation**(*run_hash: str, generation_idx: int, num_survivors: int, num_accepted: int, populace=None, dumpfile=None, fitness_calculator=None*)

Bases: object

Stores each generation of structures.

Parameters

- **run_hash** (*str*) – hash for this GA run,
- **generation_idx** (*int*) – index of this generation,
- **num_survivors** (*int*) – number of structures to aim for per generation,
- **num_accepted** (*int*) – number to accept from this generation, i.e. excluding elites,

Keyword Arguments

- **populace** (*list (dict)*) – initial structures to populate generation with (optional)
- **dumpfile** (*str*) – dumpfile name for this generation (optional)
- **fitness_calculator** (*str*) – fitness metric to use, e.g. 'hull'.

dump(*gen_suffix*)

Dump the current generation to JSON file.

Parameters **gen_suffix** (*str*) – typically gen<gen_number>.

dump_bourgeoisie(*gen_suffix*)

Dump the current generation's bourgeoisie to JSON file.

Parameters **gen_suffix** (*str*) – typically gen<gen_number>.

load(*gen_fname*)

Load populace of the generation from a JSON dump.

Parameters **gen_fname** (*str*) – filename to load.

load_bourgeoisie(*bourge_fname*)

Load bourgeoisie of the generation from a JSON dump.

Parameters **bourge_fname** (*str*) – filename to load.

birth(*populum: dict*)

Add a structure to the populace.

Parameters **populum** (*dict*) – structure to add.

rank()

Evaluate the fitness of all structures in the generation.

clean()

Remove structures with pathological formation enthalpies.

Returns number of pathological structures removed.

Return type num_removed (int)

set_bourgeoisie(*elites=None, best_from_stoich=True*)

Set the structures that will continue to the next generation, i.e. the bourgeoisie.

Keyword Arguments

- **list** (*elites*) – list of elite structures to include from the previous generation,
- **best_from_stoich** (*bool*) – whether to include one structure from each stoichiometry.

calc_pdfs()
Compute PDFs for each structure in the generation.

is_dupe(*doc*, *sim_tol=0.05*, *extra_pdfs=None*)
Compare doc with all other structures at same stoichiometry via PDF overlap.

Parameters **doc** (*dict*) – structure to compare.

Keyword Arguments

- **sim_tol** (*float*) – similarity tolerance to compare to
- **extra_pdfs** (*list (dict)*) – list of structures with extra pdfs to compare against

property pdfs
Returns list of PDFs for generation, calculating if necessary.

property fitnesses
Return list of normalised fitnesses for population.

property raw_fitnesses
Return list of raw fitnesses for population.

property average_pleb_fitness
Return the average normalised fitness of the generation.

property average_bourgeois_fitness
Return the average normalised fitness of the bourgeoisie.

ilustrado.ilustrado module

This file implements the GA algorithm and acts as main().

class ilustrado.ilustrado.**ArtificialSelector**(***kwargs*)
Bases: object

ArtificialSelector takes an initial gene pool and applies a genetic algorithm to optimise some fitness function.

Keyword Arguments

- **gene_pool** (*list (dict)*) – initial cursor to use as "Generation 0",
- **seed** (*str*) – seed name of cell and param files for CASTEP,
- **seed_prefix** (*str*) – if not specifying a seed, this name will prefix all runs
- **fitness_metric** (*str*) – currently either 'hull' or 'test',
- **hull** (*QueryConvexHull*) – matador QueryConvexHull object to calculate distances,
- **res_path** (*str*) – path to folder of res files to create hull, if no hull object passed
- **mutation_rate** (*float*) – rate at which to perform single-parent mutations (DEFAULT: 0.5)
- **crossover_rate** (*float*) – rate at which to perform crossovers (DEFAULT: 0.5)
- **num_generations** (*int*) – number of generations to breed before quitting (DEFAULT: 5)

- **num_survivors** (*int*) – number of structures to survive to next generation for breeding (DEFAULT: 10)
- **population** (*int*) – number of structures to breed in any given generation (DEFAULT: 25)
- **failure_ratio** (*int*) – maximum number of attempts per success (DEFAULT: 5)
- **elitism** (*float*) – fraction of next generation to be comprised of elite structures from previous generation (DEFAULT: 0.2)
- **best_from_stoich** (*bool*) – whether to always include the best structure from a stoichiomtery in the next generation,
- **mutations** (*list (str)*) – list of mutation names to use,
- **structure_filter** (*fn (doc)*) – any function that takes a matador doc and returns True or False,
- **check_dupes** (*bool*) – if True, filter relaxed structures for uniqueness on-the-fly (DEFAULT: True)
- **check_dupes_hull** (*bool*) – compare pdf with all hull structures (DEFAULT: True)
- **sandbagging** (*bool*) – whether or not to disfavour nearby compositions (DEFAULT: False)
- **minsep_dict** (*dict*) – dictionary containing element-specific minimum separations, e.g. {('K', 'K'): 2.5, ('K', 'P'): 2.0}. These should only be set such that atoms do not overlap; let the DFT deal with bond lengths. No effort is made to push apart atoms that are too close, the trial will simply be discarded. (DEFAULT: None)
- **max_num_mutations** (*int*) – maximum number of mutations to perform on a single structure,
- **max_num_atoms** (*int*) – most atoms allowed in a structure post-mutation/crossover,
- **nodes** (*list (str)*) – list of node names to run on,
- **ncores** (*int or list (int)*) – specifies the number of cores used by listed *nodes* per thread,
- **nprocs** (*int*) – total number of processes,
- **recover_from** (*str*) – recover from previous run_hash, by default ilustrado will recover if it finds only one run hash in the folder
- **load_only** (*bool*) – only load structures, do not continue breeding (DEFAULT: False)
- **executable** (*str*) – path to DFT binary (DEFAULT: castep)
- **compute_mode** (*str*) – either *direct*, *slurm*, *manual* (DEFAULT: direct)
- **max_num_nodes** (*int*) – amount of array jobs to run per generation in *slurm* mode,
- **walltime_hrs** (*int*) – maximum walltime for a SLURM array job,
- **slurm_template** (*str*) – path to template slurm script that includes module loads etc,
- **entrypoint** (*str*) – path to script that initialised this object, such that it can be called by SLURM
- **debug** (*bool*) – maximum printing level
- **testing** (*bool*) – run test code only if true
- **verbosity** (*int*) – extra printing level,

- **loglevel** (*str*) – follows std library logging levels.

start()
Start running GA.

breed_generation()
Build next generation from mutations/crossover of current and perform relaxations if necessary.

write_unrelaxed_generation()
Perform mutations and write res files for the resulting structures. Additionally, dump an unrelaxed json file.

batch_birth()
Assess whether a generation has been relaxed already. This is done by checking for the existence of a file called <run_hash>-genunrelaxed.json.

If so, match the relaxations up with the cached unrelaxed structures and rank them ready for the next generation.

If not, create a new generation of structures, dump the unrelaxed structures to file, create the jobscripts to relax them, submit them and the job to check up on the relaxations, then exit.

slurm_submit_relaxations_and_monitor()
Prepare and submit the appropriate slurm files.

continuous_birth()
Create new generation and relax "as they come", filling the compute resources allocated.

enforce_elitism()
Add elite structures from previous generations to bourgeoisie of current generation, through the merit of their ancestors alone.

reset_and_dump()
Add now complete generation to generation list, reset the next_gen variable and write dump files.

birth_new_structure()
Generate a new structure from current settings.

Returns newborn structure to be optimised

Return type dict

scrape_result(*result*, *proc=None*, *newborns=None*)
Check process for result and scrape into self.next_gen if successful, with duplicate detection if desired. If the optional arguments are provided, extra logging info will be found when running in *direct* mode.

Parameters **result** (*dict*) – containing output from process

Keyword Arguments

- **proc** (*tuple*) – standard process tuple from above,
- **newborns** (*list*) – of new structures to append result to.

kill_all(*procs*)
Loop over processes and kill them all.

Parameters **procs** (*list*) – list of `NewbornProcess` in form documented above.

recover()
Attempt to recover previous generations from files in cwd named '<run_hash>_gen{}.json'.format(gen_idx).

seed_generation_0(*gene_pool*)
Set up first generation from gene pool.

Parameters **gene_pool** (*list (dict)*) – list of structure with which to seed generation.

is_newborn_dupe (*newborn*, *extra_pdfs=None*)
Check each generation for a duplicate structure to the current newborn, using PDF calculator from matador.

Parameters **newborn** (*dict*) – new structure to screen against the existing,

Keyword Arguments **extra_pdfs** (*list (dict)*) – any extra PDFs to compare to, e.g. other hull structures not used to seed any generation

Returns True if duplicate, else False.

Return type bool

finalise_files_for_export ()
Move unique structures from gen1 onwards to folder "<run_hash>-results".

ilustrado.mutate module

This file implements all possible single mutant mutations.

ilustrado.mutate.**mutate** (*parent*, *mutations=None*, *max_num_mutations=2*, *debug=False*)
Wrap _mutate to check for null/invalid mutations.

Parameters **parent** (*dict*) – parent structure to mutate,

Keyword Arguments

- **mutations** (*list (fn)*) – list of possible mutation functions to apply,
- **max_num_mutations** (*int*) – maximum number of mutations to apply.

ilustrado.mutate.**permute_atoms** (*mutant*, *debug=False*)
Swap the positions of random pairs of atoms.

Parameters **mutant** (*dict*) – structure to mutate in-place.

Raises **RuntimeError** – if only one type of atom is present.

ilustrado.mutate.**transmute_atoms** (*mutant*, *debug=False*)
Transmute one atom for another type in the cell.

Parameters **mutant** (*dict*) – structure to mutate in-place.

Raises **RuntimeError** – if only one type of atom is present.

ilustrado.mutate.**vacancy** (*mutant*, *debug=False*)
Remove a random atom from the structure.

Parameters **mutant** (*dict*) – structure to mutate in-place.

ilustrado.mutate.**voronoi_shuffle** (*mutant*, *element_to_remove=None*, *preserve_stoich=False*, *debug=False*, *testing=False*)
Remove all atoms of type element, then perform Voronoi analysis on the remaining sublattice. Cluster the nodes with KMeans, then repopulate the clustered Voronoi nodes with atoms of the removed element.

Parameters **mutant** (*dict*) – structure to mutate in-place.

Keyword Arguments

- **element_to_remove** (*str*) – symbol of element to remove,
- **preserve_stoich** (*bool*) – whether to always reinsert the same number of atoms.
- **testing** (*bool*) – write a cell at each step, with H atoms indicating Voronoi nodes.

Raises `RuntimeError` – if unable to perform Voronoi shuffle.

`ilustrado.mutate.`**`random_strain`**(*mutant*, *debug=False*)

Apply random strain tensor to unit cell from 6 epsilon_i components with values between -1 and 1. The cell is then scaled to the parent's volume.

Parameters **`mutant`** (*dict*) – structure to mutate in-place.

`ilustrado.mutate.`**`nudge_positions`**(*mutant*, *amplitude=0.5*, *debug=False*)

Apply Gaussian noise to all atomic positions.

Parameters **`mutant`** (*dict*) – structure to mutate in-place.

Keyword Arguments **`amplitude`** (*float*) – amplitude of random noise in Angstroms.

`ilustrado.mutate.`**`null_nudge_positions`**(*mutant*, *debug=False*)

Apply minimal Gaussian noise to all atomic positions, mostly for testing purposes.

Parameters **`mutant`** (*dict*) – structure to mutate in-place.

ilustrado.util module

Catch-all file for utility functions.

`ilustrado.util.`**`strip_useless`**(*doc*, *to_run=False*)

Strip useless information from a matador doc.

Parameters

- **`doc`** (*dict*) – structure to strip information from.
- **`to_run`** (*bool*) – whether the structure needs to be rerun, i.e. whether to delete data from previous run.

Returns matador document stripped of useless keys

Return type dict

`class` `ilustrado.util.`**`FakeComputeTask`**(**args*, ***kwargs*)

Bases: `matador.compute.compute.ComputeTask`

Fake Relaxer for testing, with same parameters as the real one from matador.compute.

`relax`()

Alias for backwards-compatibility.

`class` `ilustrado.util.`**`NewbornProcess`**(*newborn_id*, *node*, *process*, *ncores=None*)

Bases: `object`

Simple container of process data.

`class` `ilustrado.util.`**`AseRelaxation`**(*doc*, *queue*, *calculator=None*)

Bases: `object`

Perform a variable cell relaxation with ASE, using a predefined calculator.

`relax`()

PYTHON MODULE INDEX

i

ilustrado, 6
ilustrado.adapt, 6
ilustrado.analysis, 7
ilustrado.crossover, 7
ilustrado.fitness, 8
ilustrado.generation, 9
ilustrado.ilustrado, 10
ilustrado.mutate, 13
ilustrado.util, 14

INDEX

A

adapt() (*in module ilustrado.adapt*), 6
apply_sandbag_multipliers() (*ilustrado.fitness.FitnessCalculator method*), 8
ArtificialSelector (*class in ilustrado.ilustrado*), 10
AseRelaxation (*class in ilustrado.util*), 14
average_bourgeois_fitness() (*ilustrado.generation.Generation property*), 10
average_pleb_fitness() (*ilustrado.generation.Generation property*), 10

B

batch_birth() (*ilustrado.ilustrado.ArtificialSelector method*), 12
birth() (*ilustrado.generation.Generation method*), 9
birth_new_structure() (*ilustrado.ilustrado.ArtificialSelector method*), 12
breed_generation() (*ilustrado.ilustrado.ArtificialSelector method*), 12

C

calc_pdfs() (*ilustrado.generation.Generation method*), 10
check_feasible() (*in module ilustrado.adapt*), 6
clean() (*ilustrado.generation.Generation method*), 9
continuous_birth() (*ilustrado.ilustrado.ArtificialSelector method*), 12
crossover() (*in module ilustrado.crossover*), 7

D

default_fitness_function() (*in module ilustrado.fitness*), 8
dump() (*ilustrado.generation.Generation method*), 9
dump_bourgeoisie() (*ilustrado.generation.Generation method*), 9

E

enforce_elitism() (*ilustrado.ilustrado.ArtificialSelector method*), 12
evaluate() (*ilustrado.fitness.FitnessCalculator method*), 8

F

FakeComputeTask (*class in ilustrado.util*), 14
finalise_files_for_export() (*ilustrado.ilustrado.ArtificialSelector method*), 13
fitness_swarm_plot() (*in module ilustrado.analysis*), 7
FitnessCalculator (*class in ilustrado.fitness*), 8
fitnesses() (*ilustrado.generation.Generation property*), 10

G

Generation (*class in ilustrado.generation*), 9

I

ilustrado
    module, 6
ilustrado.adapt
    module, 6
ilustrado.analysis
    module, 7
ilustrado.crossover
    module, 7
ilustrado.fitness
    module, 8
ilustrado.generation
    module, 9
ilustrado.ilustrado
    module, 10
ilustrado.mutate
    module, 13
ilustrado.util
    module, 14
is_dupe() (*ilustrado.generation.Generation method*), 10

is_newborn_dupe() (*ilustrado.ilustrado.ArtificialSelector method*), 13

K

kill_all() (*ilustrado.ilustrado.ArtificialSelector method*), 12

L

load() (*ilustrado.generation.Generation method*), 9
load_bourgeoisie() (*ilustrado.generation.Generation method*), 9

M

minseps_feasible() (*in module ilustrado.adapt*), 7
module
    ilustrado, 6
    ilustrado.adapt, 6
    ilustrado.analysis, 7
    ilustrado.crossover, 7
    ilustrado.fitness, 8
    ilustrado.generation, 9
    ilustrado.ilustrado, 10
    ilustrado.mutate, 13
    ilustrado.util, 14
mutate() (*in module ilustrado.mutate*), 13

N

NewbornProcess (*class in ilustrado.util*), 14
nudge_positions() (*in module ilustrado.mutate*), 14
null_nudge_positions() (*in module ilustrado.mutate*), 14

P

pdfs() (*ilustrado.generation.Generation property*), 10
permute_atoms() (*in module ilustrado.mutate*), 13
plot_new_2d_hull() (*in module ilustrado.analysis*), 7

R

random_slice() (*in module ilustrado.crossover*), 7
random_strain() (*in module ilustrado.mutate*), 14
rank() (*ilustrado.generation.Generation method*), 9
raw_fitnesses() (*ilustrado.generation.Generation property*), 10
recover() (*ilustrado.ilustrado.ArtificialSelector method*), 12
relax() (*ilustrado.util.AseRelaxation method*), 14
relax() (*ilustrado.util.FakeComputeTask method*), 14
reset_and_dump() (*ilustrado.ilustrado.ArtificialSelector method*), 12

S

scrape_result() (*ilustrado.ilustrado.ArtificialSelector method*), 12
seed_generation_0() (*ilustrado.ilustrado.ArtificialSelector method*), 12
set_bourgeoisie() (*ilustrado.generation.Generation method*), 9
slurm_submit_relaxations_and_monitor() (*ilustrado.ilustrado.ArtificialSelector method*), 12
start() (*ilustrado.ilustrado.ArtificialSelector method*), 12
strip_useless() (*in module ilustrado.util*), 14

T

transmute_atoms() (*in module ilustrado.mutate*), 13

U

update_sandbag_multipliers() (*ilustrado.fitness.FitnessCalculator method*), 8

V

vacancy() (*in module ilustrado.mutate*), 13
voronoi_shuffle() (*in module ilustrado.mutate*), 13

W

write_unrelaxed_generation() (*ilustrado.ilustrado.ArtificialSelector method*), 12