```
# Schaffer:
## results:
rank   name     med        iqr                                                  10%              30%              50%              70%              90%
==========================================================================================================================================================
   1   DE,    2.0,           0.0   (*                |                )           2.00,            2.00,            2.00,            2.00,            2.00
   1   SA,    2.9e+09,    7.8e+09   (-   *      -----|-----           )    97433149.49,     931952794.75,    2918896169.39,    7061467050.51,    14519397165.94
   2   MWS,   4.5e+09,    9.4e+09   (--     *       -|-------        )    162946665.19,    1689515599.56,    4472539333.25,    9438909195.14,    15840600984.07
   2   PSO,   4.9e+09,      1e+10   (--      *      -|--------       )    192241766.81,    1733022451.28,    4884121021.14,    9568503411.88,    16602073309.17
   3   GA,    5.4e+09,    1.2e+10   (--      *      |---------       )    164273521.00,    1671205148.69,    5391641903.54,    11131950934.38,   17159948413.69

## time:
rank   name     med      iqr                                                 10%        30%        50%        70%        90%
==========================================================================================================================
   1   MWS,   0.0039,   0.0075   (*-               |               )      0.00,      0.00,      0.00,      0.01,      0.03
   1   SA,    0.0045,   0.0042   (*                |               )      0.00,      0.00,      0.00,      0.01,      0.02
   1   GA,    0.0047,   0.0024   (*                |               )      0.00,      0.00,      0.00,      0.01,      0.01
   1   PSO,   0.0071,   0.0061   (*-               |               )      0.00,      0.00,      0.01,      0.01,      0.02
   2   DE,    0.073,     0.014   (      *-         |               )      0.06,      0.07,      0.07,      0.08,      0.09

# Kursawe:
## results:
rank   name      med        iqr                                                 10%        30%        50%        70%        90%
==========================================================================================================================
   1   DE,    -1.4e+01,   3.6e-15   (*                |                )    -13.73,    -13.73,    -13.73,    -13.73,    -13.73
   1   SA,    -2.5,          6.0   (     ----   * -|---             )     -8.56,     -4.68,     -2.46,     -0.05,      4.25
   1   MWS,   -2.4,          4.8   (       --   *--|-                )     -7.12,     -4.62,     -2.42,     -0.88,      2.37
   2   GA,    -1.2,          7.2   (       ---   * |-----           )     -7.17,     -4.32,     -1.24,      1.82,      6.26
   2   PSO,   -1.1,          6.8   (      -----  * |----            )     -7.53,     -3.38,     -1.06,      1.69,      5.74

## time:
rank   name     med      iqr                                                 10%        30%        50%        70%        90%
==========================================================================================================================
   1   SA,    0.0052,   0.0035   (*                |               )      0.00,      0.00,      0.01,      0.01,      0.02
   1   MWS,   0.0054,   0.0029   (*                |               )      0.00,      0.00,      0.01,      0.01,      0.02
   1   PSO,   0.0069,   0.0064   (*-               |               )      0.00,      0.01,      0.01,      0.01,      0.03
   1   GA,    0.015,    0.0042   ( *-              |               )      0.01,      0.01,      0.02,      0.02,      0.03
   2   DE,    0.12,      0.021   (           - *-|               )      0.10,      0.11,      0.12,      0.12,      0.14

# Fonseca:
## results:
rank   name   med      iqr                                                10%        30%        50%        70%        90%
==========================================================================================================================
   1   DE,    1.0,    0.0018   (*               |              )      1.00,      1.00,      1.00,      1.00,      1.00
   2   SA,    2.0,    0.0033   (                |           --*)      1.92,      2.00,      2.00,      2.00,      2.00
   2   PSO,   2.0,   0.00093   (                |            -*)      1.97,      2.00,      2.00,      2.00,      2.00
   2   GA,    2.0,    0.0016   (                |            -*)      1.96,      2.00,      2.00,      2.00,      2.00
   2   MWS,   2.0,   2.3e-07   (                |             *)      2.00,      2.00,      2.00,      2.00,      2.00

## time:
rank   name     med      iqr                                                 10%        30%        50%        70%        90%
==========================================================================================================================
   1   MWS,   0.0054,   0.0025   (*----           |               )      0.01,      0.01,      0.01,      0.01,      0.03
   1   SA,    0.0064,   0.0097   (*----           |               )      0.01,      0.01,      0.01,      0.01,      0.03
   1   PSO,   0.0086,   0.0045   ( *-             |               )      0.00,      0.01,      0.01,      0.01,      0.02
   1   GA,    0.016,    0.0028   (  *-            |               )      0.01,      0.02,      0.02,      0.02,      0.03
   2   DE,    0.086,      0.02   (                |-* -----       )      0.08,      0.08,      0.09,      0.10,      0.12

# ZDT1:
## results:
rank   name     med       iqr                                                 10%        30%        50%        70%        90%
==========================================================================================================================
   1   DE,    1.8,       0.034   (*               |               )      1.79,      1.80,      1.82,      1.83,      1.85
   1   SA,    9.6,         1.1   (               -| *-             )      8.56,      9.24,      9.64,     10.09,     10.84
   1   PSO,   9.9,         1.3   (                | *--           )      8.73,      9.43,      9.93,     10.42,     11.11
   2   GA,    1e+01,       1.3   (                | * -           )      8.74,      9.43,      9.98,     10.48,     11.22
   3   MWS,   1.4e+01,     1.3   (                |       -- *-   )     12.44,     13.35,     13.92,     14.38,     14.85

## time:
rank   name     med      iqr                                                 10%        30%        50%        70%        90%
==========================================================================================================================
   1   MWS,   0.0058,    0.004   (*               |               )      0.00,      0.00,      0.01,      0.01,      0.01
   1   SA,    0.0078,   0.0028   (*               |               )      0.00,      0.01,      0.01,      0.01,      0.01
   2   PSO,   0.019,      0.01   (*               |               )      0.01,      0.01,      0.02,      0.02,      0.03
   3   GA,    0.033,    0.0077   ( *-             |               )      0.03,      0.03,      0.03,      0.04,      0.05
   3   DE,    0.16,      0.025   (         *--    |               )      0.15,      0.16,      0.16,      0.18,      0.21

# ZDT3:
## results:
rank   name    med     iqr                                               10%        30%        50%        70%        90%
==========================================================================================================================
   1   DE,    0.69,   0.0084   (*               |              )      0.69,      0.69,      0.69,      0.70,      0.70
   1   SA,    5.2,       3.7   (    ---   *  --|--            )      2.54,      4.05,      5.22,      6.95,      9.05
   1   PSO,   5.7,       4.2   (    ---    *   |---           )      2.80,      4.24,      5.71,      7.65,      9.49
   2   MWS,   5.9,       3.5   (        -  *   |--------      )      4.50,      4.94,      5.86,      7.62,     11.80
   2   GA,    6.1,       4.2   (    ----    *  |---           )      2.82,      4.69,      6.14,      8.15,      9.49

## time:
rank   name     med      iqr                                                 10%        30%        50%        70%        90%
==========================================================================================================================
   1   MWS,   0.0061,   0.0036   (*               |               )      0.00,      0.00,      0.01,      0.01,      0.01
```

```
      1   SA,    0.0093,   0.0041   (*             |             )      0.01,      0.01,      0.01,      0.01,      0.02
      2   PSO,   0.021,     0.012   ( *            |             )      0.01,      0.02,      0.02,      0.03,      0.04
      3   GA,    0.038,    0.0078   (  *           |             )      0.03,      0.04,      0.04,      0.04,      0.06
      3   DE,    0.19,      0.019   (         -*---|             )      0.18,      0.19,      0.19,      0.20,      0.28

# Viennet3:
## results:
rank   name     med        iqr                                          10%        30%        50%        70%        90%
==========================================================================================================================
   1   DE,    1.6e+01,   3.5e-12   (*                |             )     15.94,     15.94,     15.94,     15.94,     15.94
   2   SA,    2.1e+01,       8.0   (- * ------       |             )     16.93,     18.17,     20.88,     24.36,     33.34
   2   PSO,   2.2e+01,       9.7   (-  * ------      |             )     17.02,     19.37,     22.00,     26.30,     36.68
   3   MWS,   2.4e+01,     1e+01   ( -  *  ------    |             )     17.90,     21.11,     23.82,     29.57,     40.16
   3   GA,    2.4e+01,   1.2e+01   (--  *  --------|-            )     17.16,     20.05,     23.91,     29.52,     45.77

## time:
rank   name     med       iqr                                        10%        30%        50%        70%        90%
==========================================================================================================================
   1   SA,    0.0033,   0.00076   (*                |             )      0.00,      0.00,      0.00,      0.00,      0.01
   1   MWS,   0.0038,    0.0076   (*                |             )      0.00,      0.00,      0.00,      0.01,      0.01
   1   PSO,   0.007,     0.0066   (*-               |             )      0.00,      0.00,      0.01,      0.01,      0.02
   1   GA,    0.013,     0.0037   ( *-              |             )      0.01,      0.01,      0.01,      0.02,      0.02
   1   DE,    0.096,      0.016   (            -*-|--            )      0.09,      0.09,      0.10,      0.10,      0.13

# DTLZ7:
## results:
rank   name     med        iqr                                           10%         30%         50%         70%         90%
==========================================================================================================================
   1   DE,    2.4e+02,     1e+01   (*                |             )    228.90,     233.08,     236.86,     240.96,     248.18
   1   SA,    4e+03,     3.7e+03   (*                |             )   2763.73,    3242.23,    4013.27,    5849.16,   15716.41
   1   MWS,   4.4e+03,   7.2e+02   (*                |             )   3896.94,    4208.92,    4424.30,    4748.17,    6963.53
   2   GA,    5e+03,     7.6e+03   (*                |             )   3100.17,    3831.79,    5021.28,    8585.11,   33167.45
   2   PSO,   5.4e+03,   6.4e+03   (*                |             )   3089.75,    3905.46,    5415.91,    8431.10,   24242.95

## time:
rank   name     med      iqr                                         10%        30%        50%        70%        90%
==========================================================================================================================
   1   MWS,   0.0081,   0.0043   (*                |             )      0.01,      0.01,      0.01,      0.01,      0.02
   1   SA,    0.01,     0.0044   (*                |             )      0.01,      0.01,      0.01,      0.01,      0.02
   2   PSO,   0.025,     0.016   (*                |             )      0.01,      0.02,      0.03,      0.03,      0.05
   3   GA,    0.052,     0.012   ( *               |             )      0.04,      0.05,      0.05,      0.06,      0.07
   3   DE,    0.27,       0.04   (         *-      |             )      0.25,      0.26,      0.27,      0.29,      0.32

# Osyczka:
## results:
rank   name     med        iqr                                           10%         30%         50%         70%         90%
==========================================================================================================================
   1   DE,    -2e+02,      2e+01   ( *--             |             )   -210.39,    -206.08,    -200.39,    -191.34,    -165.01
   2   SA,    2.9e+01,   4.5e+01   (                 |--- * ---    )    -23.40,      13.03,      28.97,      46.74,      79.01
   3   GA,    3.2e+01,   5.4e+01   (                 |---  * --    )    -24.78,      14.41,      32.33,      55.74,      87.99
   3   PSO,   3.3e+01,   5.5e+01   (                 |---  * --    )    -24.89,      13.81,      33.24,      55.30,      86.48
   4   MWS,   4e+01,     6.2e+01   (                 | --- *  --   )     -8.64,      20.15,      39.97,      69.69,      97.29

## time:
rank   name    med      iqr                                        10%        30%        50%        70%        90%
==========================================================================================================================
   1   SA,    0.047,    0.03   (*                |             )      0.03,      0.04,      0.05,      0.06,      0.10
   2   MWS,   0.11,    0.074   (*                |             )      0.06,      0.10,      0.11,      0.16,      0.20
   2   PSO,   0.13,    0.081   ( *               |             )      0.08,      0.10,      0.13,      0.16,      0.23
   3   DE,    0.4,     0.067   (  -*             |             )      0.32,      0.36,      0.40,      0.42,      0.47
   3   GA,    2.3,      0.78   (                 |---   * ---  )      1.65,      2.00,      2.39,      2.59,      2.93

# Schwefel(10):
## results:
rank   name     med        iqr                                            10%           30%           50%           70%           90%
==========================================================================================================================
   1   DE,    3.8e+04,   6.8e+04   (*                |             )     9398.82,     24155.45,     38456.01,     71216.43,    119617.43
   2   MWS,   5.3e+05,   4.1e+05   (   -- *  ---     |             )   299754.04,    428716.27,    535147.31,    727662.07,    996311.00
   2   SA,    5.8e+05,   3.7e+05   (   --  * ---     |             )   311589.45,    450820.68,    577994.00,    734772.72,   1020091.39
   2   PSO,   6e+05,     3.9e+05   (   --  * ----    |             )   304960.73,    448933.10,    595883.35,    756940.30,   1079664.46
   3   GA,    6.7e+05,   3.8e+05   (    --  * ---- |             )   356376.43,    541309.31,    674641.20,    833176.48,   1134702.15

## time:
rank   name     med      iqr                                        10%        30%        50%        70%        90%
==========================================================================================================================
   1   MWS,   0.0096,   0.0049   (*                |             )      0.01,      0.01,      0.01,      0.01,      0.02
   1   SA,    0.015,    0.0073   (*-               |             )      0.01,      0.01,      0.01,      0.02,      0.03
   2   GA,    0.028,    0.0046   ( -*              |             )      0.03,      0.03,      0.03,      0.03,      0.04
   2   PSO,   0.03,      0.026   (- *-             |             )      0.01,      0.02,      0.03,      0.04,      0.05
   2   DE,    0.19,      0.016   (                 |-*-          )      0.18,      0.19,      0.19,      0.20,      0.21

# Schwefel(20):
## results:
rank   name     med        iqr                                            10%           30%           50%           70%           90%
==========================================================================================================================
   1   DE,    4.6e+05,   2.8e+05   (-*-              |             )   227277.66,    334030.49,    457143.03,    563344.58,    699900.82
   2   PSO,   2.4e+06,     1e+06   (       --  * --|             )  1513605.69,   2017507.82,   2380463.37,   2812731.32,   3432390.30
   3   SA,    2.4e+06,     1e+06   (       --  * --|-            )  1572395.16,   2039636.70,   2405237.25,   2803980.96,   3523037.98
   4   MWS,   2.5e+06,   1.4e+06   (      ---   * -|--           )  1414517.69,   1902301.78,   2490319.58,   2973030.54,   3712973.82
   4   GA,    2.6e+06,   1.1e+06   (        --  * -|---         )  1772343.14,   2237513.07,   2577732.90,   3030661.56,   3867330.52
```

```
## time:
rank   name    med      iqr                                          10%        30%        50%        70%        90%
================================================================================================================
   1   MWS,   0.011,   0.0032   (*             |             )      0.01,      0.01,      0.01,      0.01,      0.02
   2   SA,    0.029,    0.013   (-*            |             )      0.02,      0.02,      0.03,      0.04,      0.04
   3   GA,    0.039,   0.0065   ( *            |             )      0.04,      0.04,      0.04,      0.04,      0.05
   4   PSO,   0.079,    0.049   ( -- *         |             )      0.03,      0.07,      0.08,      0.09,      0.11
   5   DE,    0.33,      0.03   (              |-*---        )      0.30,      0.32,      0.33,      0.34,      0.39

# Schwefel(40):
## results:
rank   name     med        iqr                                            10%            30%            50%            70%            90%
================================================================================================================================================
   1   DE,    2.7e+06,   9.3e+05   (  *            |             )    1874682.16,    2334096.55,    2676555.13,    3051196.70,    3658540.95
   2   SA,    9.9e+06,   3.1e+06   (         -- * -|-            )    7419991.44,    8755148.67,    9905213.57,   11102180.11,   13180752.53
   3   MWS,   1e+07,     2.9e+06   (         --  * |--           )    7764891.05,    9020019.28,   10337733.36,   11412644.29,   13562228.20
   3   PSO,   1e+07,     3.4e+06   (         --- * |---          )    7477745.76,    9275101.85,   10478688.75,   11897964.72,   14066680.84
   4   GA,    1.1e+07,   3.3e+06   (         ---  *|---          )    7888093.43,    9663087.02,   10899431.15,   12201015.74,   14089331.82

## time:
rank   name    med      iqr                                          10%        30%        50%        70%        90%
================================================================================================================
   1   MWS,   0.016,   0.0061   (*             |             )      0.01,      0.01,      0.02,      0.02,      0.02
   1   GA,    0.07,     0.011   ( *            |             )      0.06,      0.07,      0.07,      0.08,      0.09
   2   SA,    0.097,    0.049   ( -*-          |             )      0.06,      0.08,      0.10,      0.11,      0.16
   3   PSO,   0.23,      0.16   (       *      |             )      0.07,      0.10,      0.23,      0.25,      0.28
   4   DE,    0.71,     0.074   (              |   -- *---   )      0.63,      0.69,      0.71,      0.75,      0.84
```

```
from __future__ import division, print_function, unicode_literals

import random
import functools
import math
import itertools
import collections


def pretty_input(t):
    float_format = lambda x: '{:.2f}'.format(x)
    str_tuple = tuple(float_format(x) for x in t)
    return ', '.join(s for s in str_tuple)


def pairs(xs):
    # from https://docs.python.org/2/library/itertools.html
    a, b = itertools.tee(xs)
    next(b, None)
    for p in itertools.izip(a, b):
        yield p


class memo(object):  # noqa -- TODO: rethink this name
    '''adapted from https://github.com/timm/sbse14/wiki/basepy'''

    def __init__(self, **kwargs):
        self.__dict__.update(kwargs)

    def to_str(self, depth=0, indent=4, infix=': ', sep=', ', d=None):
        return '{' + self._to_str(
            depth=depth + 1,
            indent=indent,
            infix=infix,
            sep=sep,
            d=self.__dict__ if d is None else d) + '}'

    def _to_str(self, depth, indent, infix, sep, d):
        after, before = [], []
        rv = ''
        for k in sorted([s for s in d.keys() if s[0] != '_']):
            val = d[k]
            if isinstance(val, memo) or type(val) == dict:
                after.append(k)
            else:
                before.append('{}{}{}'.format(k, infix, repr(val)))
        if before:
            rv += '\n' + ' ' * depth * indent
            rv += sep.join(before)
        rv += '\n'

        for k in after:
            rv += ''.join([' ' * depth * indent, k, infix, '{'])
            k = d[k]
            k = k if type(k) == dict else k.__dict__
            rv += ''.join([self._to_str(depth=depth+1, indent=indent,
                           infix=infix, sep=sep, d=k),
                           ' ' * depth * indent,
                           '}\n'])

        return rv


def memoize(f):
    'memoizer for single-arg functions'
    d = {}

    @functools.wraps(f)
    def wrapper(x):
        try:
            return d[x]
        except KeyError:
            d[x] = f(x)
```



```
            return d[x]

    return wrapper


@memoize
def memo_sqrt(x):
    return math.sqrt(x)


def tuple_replace(t, replace_at, value):
    return tuple(value if i == replace_at else v for i, v in enumerate(t))


def random_index(x):
    '''
    Given a dict, list, tuple, or a subclass of one of these, return a random
    valid key for it.
    '''
    if isinstance(x, dict) or issubclass(x.__class__, dict):
        return random.choice(x.keys)
    if isinstance(x, (list, tuple)) or issubclass(x.__class__, (list, tuple)):
        return random.randint(0, len(x) - 1)
    raise ValueError('{} is not a dict, list, or tuple'.format(x))


class StringBuilder(object):
    def __init__(self, *args):
        self._s = ''.join(args)
        self._next = []

    def append(self, arg):
        'recurse through iterables in args, adding all strings to _next '
        'raises TypeError if it finds a non-Iterable non-string'
        if isinstance(arg, basestring):
            self._next.append(arg)
        elif isinstance(arg, collections.Iterable):
            map(self.append, arg)
        else:
            raise TypeError('{} not a string or iterable'.format(arg))

    def __iadd__(self, arg):
        self.append(arg)
        return self

    def as_str(self):
        'build and cache _s if necessary, then return it.'
        if self._next:
            self._s += ''.join(self._next)
            self._next = []
        return self._s

    def __repr__(self):
        return "{}('{}')".format(self.__class__.__name__, self.as_str())


class NullObject(object):
    __slots__ = ()

    def __init__(self, *args, **kw):
        return None

    def _return_self(self, *name, **kw):
        return self

    __getattribute__ = _return_self
    __setattr__ = _return_self
    __iadd__ = _return_self
    __call__ = _return_self

    def __bool__(self, *args, **kw):
        return False
    __nonzero__ = __bool__
```

```
from __future__ import division, print_function

import itertools
import base


def median(xs, is_sorted=False):
    """
    Return the median of the integer-indexed object passed in. To save sorting
    time, the client can pass in is_sorted=True to skip the sorting step.
    """
    # implementation from http://stackoverflow.com/a/10482734/3408454
    if not is_sorted:
        xs = sorted(xs)
    n = len(xs)
    return xs[n // 2] if n % 2 else (xs[n // 2] + xs[n // 2 - 1]) / 2


def mean(xs):
    "Returns the mean of the iterable argument."
    return sum(xs) / len(xs)


def iqr(xs):
    n = len(xs)
    return xs[int(n * .75)] - xs[int(n * .25)]


_mean = mean  # 'mean' alias for use by standard_deviation, which shadows it


def standard_deviation(xs, mean=None):
    if mean is None:
        mean = _mean(xs)
    return base.memo_sqrt(sum((x - mean) ** 2 for x in xs))


def norm(x, a, b):
    lo, hi = min(a, b), max(a, b)
    try:
        return (x - lo) / (hi - lo)
    except ZeroDivisionError:
        return .5


def value_at_proportion(p, xs):
    return xs[int(round(len(xs) - 1) * p)]


def percentile(x, xs, is_sorted=False):
    if not is_sorted:
        xs = sorted(xs)
    before = len(tuple(itertools.takewhile(lambda y: y < x, xs)))
    return before / len(xs)


def xtile(xs, lo=0, hi=0.001, width=50,
          chops=[0.1, 0.3, 0.5, 0.7, 0.9], marks=["-", " ", " ", "-", " "],
          bar="|", star="*", show=" {: >6.2f}",
          as_list=False):
    """Take an iterable of numbers and present them as a horizontal xtile
    ascii chart. The default is a contracted quintile showing the 10th, 30th,
    50th, 70th, and 90th percentiles. These breaks can be customized with the
    chops parameter.
    """

    xs = sorted(xs)

    lo, hi = min(lo, xs[0]), max(hi, xs[-1])
    if hi == lo:
        hi += .001  # ugh

    out = [' '] * width
```



```
    out_index_for_value = lambda x: min(width-1,
                                        int(len(out) * norm(x, lo, hi)))

    values_at_chops = tuple(xs[int(len(xs) * p)] for p in chops)
    where = [out_index_for_value(n) for n in values_at_chops]

    for one, two in base.pairs(where):
        for i in range(one, two):
            out[i] = marks[0]
        marks = marks[1:]

    out[int(width / 2)] = bar
    out[out_index_for_value(xs[int(len(xs) * 0.5)])] = star

    if as_list:
        rv = ['(' + ''.join(out) + ")"]
        rv.extend(show.format(x) for x in values_at_chops)
        return rv

    return ''.join(out) + "," + ','.join([show.format(x)
                                          for x in values_at_chops])
```

```
from __future__ import division, print_function

import random
import functools
import collections
import itertools

from sortedcontainers import SortedList

from witschey import base


class Log(object):
    """Keep a random sample of stuff seen so far. Based on Dr. Menzies'
    implementation."""

    MAX_SIZE = 256

    def __init__(self, inits=None, label=None, max_size=MAX_SIZE):
        self._cache = SortedList()
        self._report = None
        self.label = label or ''
        self._n = 0
        self.max_size = max_size
        self._valid_statistics = False
        self._invalidate_statistics()
        if inits:
            map(self.__iadd__, inits)

    def random_index(self):
        return base.random_index(self._cache)

    @classmethod
    def wrap(cls, x, max_size=MAX_SIZE):
        if isinstance(x, cls):
            return x
        return cls(inits=x, max_size=max_size)

    def __len__(self):
        return len(self._cache)

    def extend(self, xs):
        if not isinstance(xs, collections.Iterable):
            raise TypeError()
        map(self.__iadd__, xs)

    def __iadd__(self, x):
        if x is None:
            return x

        self._n += 1

        if issubclass(x.__class__, Log):
            map(self.__iadd__, x._cache)
            return self

        changed = False

        # if cache has room, add item
        if self.max_size is None or len(self._cache) < self.max_size:
            changed = True
            self._cache.add(x)
        # cache is full: maybe replace an old item
        else:
            # items less likely to be replaced later in the run:
            # leads to uniform sample of entire run
            if random.random() <= self.max_size / len(self):
                changed = True
                self._cache.remove(random.choice(self._cache))
                self._cache.add(x)

        if changed:
            self._invalidate_statistics()
```



```
            self._change(x)

        return self

    def __add__(self, x, max_size=MAX_SIZE):
        inits = itertools.chain(self._cache, x._cache)
        return self.__class__(inits=inits, max_size=max_size)

    def any(self):
        return random.choice(self._cache)

    def report(self):
        if self._report is None:
            self._report = self._generate_report()
        return self._report

    def setup(self):
        raise NotImplementedError()

    def as_list(self):
        return self._cache.as_list()

    def _invalidate_statistics(self):
        '''
        default implementation. if _valid_statistics is something other than
        a boolean, reimplement!
        '''
        self._valid_statistics = False

    def ish(self, *args, **kwargs):
        raise NotImplementedError()

    def _change(self, x):
        '''
        override to add incremental updating functionality
        '''
        pass

    def _prepare_data(self):
        s = '_prepare_data() not implemented for ' + self.__class__.__name__
        raise NotImplementedError(s)

    def __iter__(self):
        return iter(self._cache)

    def contents(self):
        return self._cache.as_list()


def statistic(f):
    '''
    decorator for log functions that return statistics about contents.
    if _valid_statistics is False, generate valid stats before calling
    the wrapped function.
    '''
    @functools.wraps(f)
    def wrapper(*args, **kwargs):
        self = args[0]
        if not self._valid_statistics:
            self._prepare_data()
        return f(*args, **kwargs)

    return wrapper
```

```
from __future__ import division

from log import Log
from witschey import base
from witschey import basic_stats


class NumberLog(Log):

    def __init__(self, *args, **kwargs):
        super(NumberLog, self).__init__(*args, **kwargs)

        self._invalidate_statistics()

    @property
    def hi(self):
        return self._cache[-1]  # assumes SortedList implementation

    @property
    def lo(self):
        return self._cache[0]  # assumes SortedList implementation

    def _invalidate_statistics(self):
        self._cached_mean, self._cached_median = None, None
        self._cached_sd, self._cached_iqr = None, None

        super(NumberLog, self)._invalidate_statistics()

    def norm(self, x):
        "normalize the argument with respect to maximum and minimum"
        if self.hi == self.lo:
            raise ValueError('hi and lo of {} are equal'.format(self.__name__))
        return basic_stats.norm(x, self.lo, self.hi)

    def _prepare_data(self):
        if not self._valid_statistics:
            pass
        self._valid_statistics = True

    def _generate_report(self):
        return base.memo(median=self.median(), iqr=self.iqr(),
                         lo=self.lo, hi=self.hi)

    def ish(self, f=0.1):
        """return a num likely to be similar to/representative of
        nums in the distribution"""
        return self.any() + f*(self.any() - self.any())

    def median(self):
        if self._cached_median is not None:
            return self._cached_median
        self._cached_median = basic_stats.median(self._cache)
        return self._cached_median

    def mean(self):
        if self._cached_mean is not None:
            return self._cached_mean
        self._cached_mean = basic_stats.mean(self._cache)
        return self._cached_mean

    def standard_deviation(self):
        if self._cached_sd is not None:
            return self._cached_sd
        self._cached_sd = basic_stats.standard_deviation(
            self._cache, mean=self.mean())
        return self._cached_sd

    def iqr(self):
        if self._cached_iqr is not None:
            return self._cached_iqr
        self._cached_iqr = basic_stats.iqr(self._cache)
        return self._cached_iqr
```



```
    def xtile(self, *args, **kw):
        return basic_stats.xtile(self._cache, *args, **kw)

    def value_at_proportion(self, p):
        return basic_stats.value_at_proportion(p, self._cache)

    def better(self, log2):
        if log2 is None:
            return ValueError
        if not self._cache or not log2._cache:
            return False
        if self.median() < log2.median():
            return True
        if self.iqr() < log2.iqr():
            return True
        return False
```

```
from model import Model, ModelIO, ModelInputException
from independent_variable import IndependentVariable
from schaffer import Schaffer
from kursawe import Kursawe
from fonseca import Fonseca
from zdt1 import ZDT1
from zdt3 import ZDT3
from viennet3 import Viennet3
from dtlz7 import DTLZ7
from schwefel import Schwefel
from osyczka import Osyczka

__all__ = [Model, IndependentVariable, ModelIO, ModelInputException,
           Schaffer, Kursawe, Fonseca,
           ZDT1, ZDT3, Viennet3,
           DTLZ7, Schwefel, Osyczka]
```



```
from __future__ import division, print_function

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from abc import ABCMeta
from collections import namedtuple
from random import sample

from witschey import basic_stats

ModelIO = namedtuple('ModelIO', ('xs', 'ys', 'energy'))


class Model(object):
    # allows us to get all subclasses with __subclasses__()
    __metaclass__ = ABCMeta

    def __init__(self, independents=None, dependents=None,
                 energy_min=None, energy_max=None):
        if independents is None or dependents is None:
            raise ValueError

        self.xs = independents
        self.ys = dependents
        self.energy_max = energy_max
        self.energy_min = energy_min

    def normalize(self, x):
        return basic_stats.norm(x, self.energy_max, self.energy_min)

    def random_input_vector(self):
        return tuple(x() for x in self.xs)

    def __call__(self, xs, io=False):
        for i, x in enumerate(xs):
            if not self.xs[i].lo <= x <= self.xs[i].hi:
                raise ModelInputException

        ys = tuple(y(xs) for y in self.ys)
        energy = sum(ys)

        if self.energy_min is None or self.energy_min > energy:
            self.energy_min = energy

        if self.energy_max is None or energy > self.energy_max:
            self.energy_max = energy

        if io:
            return ModelIO(xs, ys, energy)

        return ys

    def energy(self, ys, norm=False):
        rv = sum(ys)
        return self.normalize(rv) if norm else rv

    def compute_model_io(self, xs):
        """
        Return a ModelIO namedtuple containing the input provided as the
        argument, the output values for each function, and the energy of that
        output.

        Since this evaluates the model on its input, this method may raise a
        ModelInputException.
        """
        ys = self(xs)
        return ModelIO(xs, ys, self.energy(ys))

    def random_model_io(self):
        """
        Generate a random input for this model, then run the model
        """
```

```
            while True:
                try:
                    return self.compute_model_io(self.random_input_vector())
                except ModelInputException:
                    pass

        def random_replace(self, xs, n=1):
            """
            Returns a tuple identical to xs, except in n positions, where the
            value has been replaced with a value randomly generated by the
            appropriate independent variable.

            >>> from independent_variable import IndependentVariable as IV
            >>> import random
            >>> random.seed(1)
            >>> ivs = tuple(IV(0, 10) for _ in range(3))
            >>> m = Model(independents=ivs, dependents=())
            >>> m.random_replace((5, 5, 5))
            (8.474337369372327, 5, 5)
            >>> m.random_replace((5, 5, 5), 2)
            (4.954350870919409, 5, 4.494910647887381)
            """
            replace_indices = sample(tuple(range(len(xs))), n)
            return tuple(self.xs[i]() if i in replace_indices else x
                         for i, x in enumerate(xs))


    class ModelInputException(Exception):
        pass
```



```
    from __future__ import division, print_function
    # all adapted from Dr. Tim Menzies' model code:
    # https://github.com/timm/sbse14/blob/master/models.py

    import random


    class IndependentVariable(object):
        """
        An independent variable for a model.

        >>> iv = IndependentVariable(0, 10)
        >>> iv.lo, iv.hi
        (0, 10)

        Call an independent variable object to generating random variables within
        its range:

        >>> random.seed(1); iv(), iv(), iv()
        (1.3436424411240122, 8.474337369372327, 7.6377461897661405)

        Provides a 'clip' method to return a variable clipped within the bounds
        of the variable:

        >>> iv.clip(10.5), iv.clip(-100), iv.clip(4.2)
        (10, 0, 4.2)

        The optional third argument to __init__ specifies the type of the
        IndependentVariable. Valid values are 'float' and 'int', and the default
        is 'float'.

        >>> iv = IndependentVariable(0, 10, int)
        >>> iv(), iv(), iv()
        (2, 5, 4)
        """

        def __init__(self, lo, hi, gen_type=float):
            self._lo = lo
            self._hi = hi
            self._type = gen_type

            if self._type == float:
                self._get = random.uniform
            elif self._type == int:
                self._get = random.randint

        def __call__(self):
            return self._get(self.lo, self.hi)

        def clip(self, x):
            """
            Clip the input number within the bounds of the independent variable.
            """
            return max(self.lo, min(self.hi, x))

        @property
        def lo(self):
            """
            Return the lower bound on values for this independent variable.
            Read-only.
            """
            return self._lo

        @property
        def hi(self):
            """
            Return the upper bound on values for this independent variable.
            Read-only.
            """
            return self._hi

        @property
        def type(self):
```

```
        """
        Return the type of this independent variable.
        Read-only.
        """
        return self._type
```



```
# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
import math

from model import Model
from independent_variable import IndependentVariable as IV  # noqa


class DTLZ7(Model):
    def __init__(self, ivs=30, dvs=20):

        # dynamically generate these suckers
        # h/t http://stackoverflow.com/a/13184536/3408454
        generated_fs = []
        for x in xrange(1, dvs):
            f = lambda xs: xs[x]
            f.__name__ = 'f{}'.format(x)
            generated_fs.append(f)

        def g(xs):
            # avoid divide by 0 errors
            denom = abs(xs[-1]) or .0001
            return 1 + (9 / denom) * sum(xs)

        def h(xs, fs=generated_fs, g=g):
            s = 0
            for f in fs:
                fxs = f(xs)
                a = fxs / (1 + g(xs))
                b = 1 + math.sin(3 * math.pi * fxs)
                s += a * b

            return dvs - s

        def final_f(xs):
            return (1 + g(xs)) * h(xs)
        final_f.__name__ = 'f{}'.format(dvs)

        fs = tuple(generated_fs + [final_f])

        independents = tuple(IV(lo=0, hi=1) for _ in xrange(ivs))
        super(DTLZ7, self).__init__(independents=independents, dependents=fs)
```

```
# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
import math

from model import Model
from independent_variable import IndependentVariable as IV  # noqa
from witschey.base import memo_sqrt


class Fonseca(Model):
    def __init__(self, ivs=3):
        ivs = tuple(IV(lo=-4, hi=4) for _ in xrange(ivs))

        def f1(xs):
            e = sum((x - (1 / memo_sqrt(i+1))) ** 2 for i, x in enumerate(xs))
            return 1 - math.exp(-e)

        def f2(xs):
            e = sum((x + (1 / memo_sqrt(i+1))) ** 2 for i, x in enumerate(xs))
            return 1 - math.exp(-e)

        super(Fonseca, self).__init__(independents=ivs, dependents=(f1, f2))
```



```
# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
import math

from model import Model
from independent_variable import IndependentVariable as IV  # noqa


class Kursawe(Model):
    def __init__(self, ivs=3, a=0.8, b=3):
        ivs = tuple(IV(lo=-5, hi=5) for _ in xrange(ivs - 1))
        self.a = a
        self.b = b

        def f1(xs):
            rv = 0
            for i in xrange(len(xs) - 1):
                exponent = (-0.2) * math.sqrt(xs[i] ** 2 + xs[i+1] ** 2)
                rv += -10 * math.exp(exponent)
            return rv

        def f2(xs):
            f = lambda x: (math.fabs(x)**self.a) + (5 * math.sin(x)**self.b)
            return sum(f(x) for x in xs)

        super(Kursawe, self).__init__(independents=ivs, dependents=(f1, f2))
```

```
# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from model import Model
from independent_variable import IndependentVariable as IV  # noqa


class Schaffer(Model):

    def __init__(self, ivs=1):
        independents = tuple(IV(lo=-10 ** 5, hi=10 ** 5) for _ in xrange(ivs))

        # use def instead of lambdas so the functions keep their __name__s
        def f1(xs):
            return sum(x ** 2 for x in xs)

        def f2(xs):
            return sum((x - 2) ** 2 for x in xs)

        super(Schaffer, self).__init__(
            independents=independents, dependents=(f1, f2))
```



```
# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
import math

from model import Model
from independent_variable import IndependentVariable as IV  # noqa


class Viennet3(Model):

    def __init__(self):

        def f1(xs):
            xs_2_sum = xs[0] ** 2 + xs[1] ** 2
            return (0.5 * xs_2_sum) + math.sin(xs_2_sum)

        def f2(xs):
            x_1 = xs[0]
            x_2 = xs[1]

            a = ((3 * x_1 - 2 * x_2 + 4) ** 2) / 8
            b = ((x_1 + x_2 + 1) ** 2) / 27

            return a + b + 15

        def f3(xs):
            x_1sq = xs[0] ** 2
            x_2sq = xs[1] ** 2

            a = 1 / (x_1sq + x_2sq + 1)
            b = 1.1 * math.exp(-x_1sq - x_2sq)

            return a - b

        ivs = (IV(lo=-3, hi=3), IV(lo=-3, hi=3))

        super(Viennet3, self).__init__(
            independents=ivs, dependents=(f1, f2, f3))
```

```
# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
import math

from model import Model
from independent_variable import IndependentVariable as IV  # noqa


class ZDT1(Model):
    def __init__(self, ivs=30):

        def g(xs):
            return 1 + 9 * sum(xs[1:]) / (len(xs) - 1)

        def f1(xs):
            return xs[0]

        def f2(xs):
            gxs = g(xs)
            return gxs * (1 - math.sqrt(xs[0] / gxs))

        ivs = tuple(IV(lo=0, hi=1) for _ in xrange(30))
        super(ZDT1, self).__init__(independents=ivs, dependents=(f1, f2, g))
```



```
# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division

import math

from model import Model
from independent_variable import IndependentVariable as IV  # noqa
from witschey.base import memo_sqrt


class ZDT3(Model):

    def __init__(self, ivs=30):

        def g(xs):
            return 1 + 9 * sum(xs[1:]) / (len(xs) - 1)

        def f1(xs):
            return xs[0]

        def f2(xs):
            gxs = g(xs)
            a = 1 - memo_sqrt(xs[0] / gxs) - (xs[0] / gxs)
            a *= math.sin(10 * math.pi * xs[0])
            return gxs * a

        ivs = tuple(IV(lo=0, hi=1) for _ in xrange(30))

        super(ZDT3, self).__init__(independents=ivs, dependents=(f1, f2, g))
```

```
from __future__ import division, print_function

import inspect

from model import Model, ModelInputException
from independent_variable import IndependentVariable as IV  # noqa


def _lambda_string_strip(s):
    return inspect.getsource(s).split('lambda x: ')[1][:-2]


class Osyczka(Model):

    def __init__(self):
        self.checks = []
        self.checks.append(lambda x: x[0] + x[1] - 2 >= 0)
        self.checks.append(lambda x: 6 - x[0] - x[1] >= 0)
        self.checks.append(lambda x: 2 - x[1] + x[0] >= 0)
        self.checks.append(lambda x: 2 - x[0] + 3 * x[1] >= 0)
        self.checks.append(lambda x: 4 - (x[2] - 3) ** 2 - x[3] >= 0)
        self.checks.append(lambda x: (x[4] - 3) ** 2 + x[5] - 4 >= 0)

        independents = tuple(IV(lo=lo, hi=hi) for lo, hi in
                             ((0, 10), (0, 10), (1, 5),
                              (0, 6), (1, 5), (0, 10)))

        def f1(xs):
            return sum(((-25 * (xs[0] - 2) ** 2),
                        (- (xs[1] - 2) ** 2),
                        (- (xs[2] - 1) ** 2),
                        (- (xs[3] - 4) ** 2),
                        (- (xs[4] - 1) ** 2)))

        def f2(xs):
            return sum(x ** 2 for x in xs)

        super(Osyczka, self).__init__(independents=independents,
                                      dependents=(f1, f2))

    def random_input_vector(self):
        while True:
            candidate = super(Osyczka, self).random_input_vector()
            try:
                self._fail_on_constraint_violations(candidate, no_msg=True)
                return candidate
            except ModelInputException:
                pass

    def valid_input(self, xs):
        try:
            self._fail_on_constraint_violations(xs, no_msg=True)
        except ModelInputException:
            return False
        return True

    def _fail_on_constraint_violations(self, xs, no_msg=False):
        msgs = []
        for check in self.checks:
            if not check(xs):
                if no_msg:
                    raise ModelInputException(xs)
                msgs.append(_lambda_string_strip(check))

        if msgs:
            err = "{} failed on input {}:".format(self.__class__.__name__, xs)
            if len(msgs) == 1:
                err += ' {}'.format(msgs[0])
            else:
                pre = '\n\tviolated constraint: '
                err += pre + pre.join(msgs)
            raise ModelInputException(err)
```



```
    def __call__(self, xs, io=False):
        self._fail_on_constraint_violations(xs)
        return super(Osyczka, self).__call__(xs, io=io)
```

```
from __future__ import division, unicode_literals

from datetime import datetime
import abc
from types import NoneType
from collections import namedtuple, OrderedDict

from witschey.base import memo
from witschey.models import Model
from witschey.config import CONFIG


class Searcher(object):
    # allows us to get all subclasses with __subclasses__()
    __metaclass__ = abc.ABCMeta

    def __new__(cls, *args, **kwargs):
        # construct our object
        future_self = super(Searcher, cls).__new__(cls, *args, **kwargs)

        name = cls.__name__
        # initialize a dict with searcher's name
        # and the initialization time
        d = dict(searcher=name, initialized=datetime.now())

        # if there are global options for this class or its bases in CONFIG
        for k in [k.__name__ for k in cls.__bases__] + [name]:
            if hasattr(CONFIG, k):
                # add them to the dict
                d.update(getattr(CONFIG, k).__dict__)

        # then, add the kwargs to the constructor call to the dict.
        # NB: this happens after adding options from The, so
        #     call-specific options override the globals
        d.update(kwargs)

        # set our spec with the contents of the dict
        future_self.spec = memo(**d)

        return future_self

    def __init__(self, model, *args, **kw):
        self.model = model()

    def run(*args, **kwargs):
        raise NotImplementedError()


class SearcherConfig(object):

    def __init__(self, searcher=None, model=None, **kwargs):
        self.searcher, self.model = searcher, model
        self._kw_dict = kwargs

    def get_searcher(self, searcher=None, model=None, **kwargs):
        s = searcher or self.searcher
        m = model or self.model
        kw = self._kw_dict.copy().update(kwargs) or {}
        return s(m, **kw)

    @property
    def searcher(self):
        return self._searcher

    @searcher.setter
    def searcher(self, value):
        if isinstance(value, NoneType) or issubclass(value, Searcher):
            self._searcher = value
        else:
            raise TypeError('{} is not a Searcher or None'.format(value))

    @property
    def model(self):
```



```
        return self._model

    @model.setter
    def model(self, value):
        if isinstance(value, NoneType) or issubclass(value, Model):
            self._model = value
        else:
            raise TypeError('{} is not a Model or None'.format(value))

    def update(self, searcher=None, model=None, **kwargs):
        if searcher is not None:
            self.searcher = searcher
        if model is not None:
            self.model = model
        self._kw_dict.update(kwargs)

    def as_dict(self):
        "returns a OrderedDict with the searcher and model first"
        return OrderedDict(searcher=self._searcher,
                           model=self._model, **self._kw_dict)

    def __repr__(self):
        kw_string = ', '.join('{0}={1}'.format(k, v)
                              for k, v in self.as_dict().iteritems())
        return '{0}({1})'.format(self.__class__.__name__, kw_string)


SearchReport = namedtuple('SearchReport',
                          ['best', 'best_era', 'evaluations', 'searcher',
                           'spec', 'report'])
```

```
from __future__ import division, print_function

import random
import math

from searcher import Searcher, SearchReport
from witschey.base import NullObject, StringBuilder
from witschey.log import NumberLog
from witschey.models import ModelInputException


class SimulatedAnnealer(Searcher):
    """
    A searcher that works by mostly-dumb stochastic search that starts with
    lots of random jumps, then makes fewer random jumps, simulating a cooling
    process. See http://en.wikipedia.org/wiki/Simulated_annealing and
    https://github.com/timm/sbse14/wiki/sa for more information.
    """

    def __init__(self, *args, **kwargs):
        super(SimulatedAnnealer, self).__init__(*args, **kwargs)
        self._current = self.model.random_model_io()
        self._best = self._current  # assumes current is immutable
        self._lives = 4
        self._best_era = None
        self._current_era_energies = NumberLog(max_size=None)

    def run(self, text_report=True):
        """
        Run the SimulatedAnnealer on the model specified at object
        instantiation time.
        """
        self._report = StringBuilder() if text_report else NullObject()
        evals = None

        for k in range(self.spec.iterations):
            if self._lives <= 0 and self.spec.terminate_early:
                evals = k
                break
            self._update(k / self.spec.iterations)
            if k % self.spec.era_length == 0 and k != 0:
                self._end_era()

        if evals is None:
            evals = self.spec.iterations
        return SearchReport(best=self._best.energy, evaluations=evals,
                            best_era=self._best_era, spec=self.spec,
                            searcher=self.__class__, report=self._report)

    def _mutate(self, xs):
        return tuple(xs[i] if random.random() < self.spec.p_mutation else v
                     for i, v in enumerate(self.model.random_input_vector()))

    def _get_neighbor(self, model_io):
        neighbor = None
        while neighbor is None:
            gen = self._mutate(model_io.xs)
            try:
                neighbor = self.model(tuple(gen), io=True)
            except ModelInputException:
                pass

        return neighbor

    def _end_era(self):
        self._report += ('\n', '{: .2}'.format(self._best.energy), ' ')
        if not self._best_era:
            self._best_era = self._current_era_energies

        try:
            improved = self._current_era_energies.better(
                self._prev_era_energies)
        except AttributeError:
            improved = False
        if improved:
            self._best_era = self._current_era_energies
        else:
            self._lives -= 1

        self._prev_era_energies = self._current_era_energies
        self._current_era_energies = NumberLog(max_size=None)

    def _update(self, temperature):
        """update the state of the annealer"""
        # generate new neighbor
        neighbor = self._get_neighbor(self._current)
        self._current_era_energies += neighbor.energy

        # compare neighbor and update best
        if neighbor.energy < self._best.energy:
            self._best, self._current = neighbor, neighbor
            self._report += '!'

        if neighbor.energy < self._current.energy:
            self._current = neighbor
            self._report += '+'
        else:
            # if neighbor is worse than current, we still jump there sometimes
            cnorm = self.model.normalize(self._current.energy)
            nnorm = self.model.normalize(neighbor.energy)
            # occasionally jump to neighbor, even if it's a bad idea
            if self._good_idea(cnorm, nnorm, temperature) < random.random():
                self._current = neighbor
                self._report += '?'

        self._report += '.'

    def _good_idea(self, old, new, temp):
        """
        sets the threshold we compare to to decide whether to jump

        returns e^-((new-old)/temp)
        """
        numerator = new - old

        if not 0 <= numerator <= 1:
            numerator = old - new
        try:
            exponent = numerator / temp
        except ZeroDivisionError:
            return 0
        rv = math.exp(-exponent)
        if rv > 1:
            raise ValueError('p returning greater than one',
                             rv, old, new, temp)
        return rv * self.spec.cooling_factor
```

```
from __future__ import division

import random

from searcher import Searcher, SearchReport
from witschey import base
from witschey.base import tuple_replace, StringBuilder, NullObject
from witschey.log import NumberLog
from witschey.models import ModelInputException


class MaxWalkSat(Searcher):

    def _local_search_xs(self, bottom, top, n=10):
        '''divide the space from bottom to top into n partitions, then
        randomly sample within each partition'''
        chunk_length = (top - bottom) / n

        for i in range(n):
            i = (i * chunk_length) + bottom
            yield random.uniform(i, i + chunk_length)

    def _update(self, improvement_char, dimension=None, value=None):
        '''calculate the next value from the model and update state as
        necessary'''
        # check for invalid input
        if value is not None and dimension is None:
            err = 'cannot call _update with specified value but no dimension'
            raise ValueError(err)

        if dimension is None:
            dimension = base.random_index(self._current.xs)

        if value is None:
            # get random value if no value input
            value = self.model.xs[dimension]()

        updated = False
        while not updated:
            new_xs = tuple_replace(self._current.xs, dimension, value)
            try:
                self._current = self.model(new_xs, io=True)
                updated = True
            except ModelInputException:
                value = self.model.xs[dimension]()

        self._evals += 1
        self._current_era += self._current.energy

        # compare to previous best and update as necessary
        if self._current.energy < self._best.energy:
            self._best = self._current
            self._report += improvement_char
        else:
            self._report += '.'

        # end-of-era bookkeeping
        if self._evals % self.spec.era_length == 0:
            self._end_era()

    def _end_era(self):
        self._report += ('\n{: .2}'.format(self._best.energy), ' ')

        # _prev_era won't exist in era 0, so account for that case
        try:
            improved = self._current_era.better(self._prev_era)
        except AttributeError:
            improved = False
        self._prev_era = self._current_era

        # track best_era
        if improved or self._best_era is None:
            self._best_era = self._current_era
```



```
        else:
            self._lives -= 1

        if self._lives <= 0:
            self._terminate = True
        else:
            self._current_era = NumberLog()

    def run(self, text_report=True):
        '''run MaxWalkSat on self.model'''

        # current ModelIO to evaluate and mutate
        self._current = self.model.random_model_io()
        self._best = self._current
        # initialize and update log variables to track values by era
        self._current_era = NumberLog()
        self._current_era += self._current.energy
        self._best_era = None
        # bookkeeping variables
        self._evals = 0
        self._lives = 4
        self._report = StringBuilder() if text_report else NullObject()
        self._terminate = False

        while self._evals < self.spec.iterations and not self._terminate:
            # get the generator for a random independent variable

            if self.spec.p_mutation > random.random():
                # if not searching a dimension, mutate randomly
                self._update('+')
            else:
                # if doing a local search, choose a dimension
                dimension = base.random_index(self._current.xs)
                search_iv = self.model.xs[dimension]
                # then try points all along the dimension
                lo, hi = search_iv.lo, search_iv.hi
                for j in self._local_search_xs(lo, hi, 10):
                    self._update('|', dimension=dimension, value=j)

        return SearchReport(best=self._best.energy,
                            best_era=self._best_era,
                            evaluations=self._evals,
                            searcher=self.__class__,
                            spec=self.spec,
                            report=self._report)
```

```
from __future__ import division, print_function

from itertools import chain, combinations, cycle, izip, tee
import random
from collections import Iterable

from witschey import base
from searcher import Searcher, SearchReport
from witschey.log import NumberLog
from witschey.models import ModelInputException

# adapted from Chris Theisen's code
#     his code provided the shell that I worked in and styled to my liking
# Structure from:
# www.cleveralgorithms.com/nature-inspired/evolution/genetic_algorithm.html


def _random_crossover_points(n, length):
    # get n random valid crossover points for a sequence of len length
    r = list(xrange(1, length - 1))
    if len(r) <= length:
        return r
    xovers = sorted(random.sample(xrange(1, length - 1), n))
    return xovers


def _crossover_at(seq1, seq2, xovers):
    # takes two sequences and a single crossover point or a list of points
    if not isinstance(xovers, Iterable):
        xovers = [xovers]
    cycle_seq = cycle((seq1, seq2))

    # iter. of start and stop points for sections
    xovers = chain((None,), xovers, (None,))
    parent_point_zip = izip(cycle_seq, base.pairs(xovers))

    segments = tuple(parent[start_stop[0]:start_stop[1]]
                     for parent, start_stop in parent_point_zip)

    return tuple(chain(*segments))


class GeneticAlgorithm(Searcher):
    """
    A searcher that searches the input space by modeling a population of
    organisms that 'breed', are selected for their good qualities, and
    mutate slightly from generation to generation.

    For more information, see https://github.com/timm/sbse14/wiki/Ga and
    http://en.wikipedia.org/wiki/Genetic_algorithm.
    """

    def _mutate(self, child):
        i = base.random_index(child)
        return base.tuple_replace(child, i, self.model.xs[i]())

    def _crossover(self, parent1, parent2, xovers=None):
        if len(parent1) != len(parent2):
            raise ValueError('parents must be same length to breed')
        if len(parent1) == 1:
            return random.choice((parent1, parent2))
        if xovers is None:
            xovers = self.spec.crossovers

        x_pts = _random_crossover_points(xovers, len(parent1))

        return _crossover_at(parent1, parent2, x_pts)

    def _select_parents(self):
        """
        Return an iterator with 2 copies of each pair of parents in the
        population
        """
```



```
        return chain(*tee(combinations(self._population, 2)))

    def _breed_next_generation(self):
        children = []
        for parent1, parent2 in self._select_parents():
            failures = 0
            child = None
            while child is None:
                xs = self._crossover(parent1.xs, parent2.xs)
                if random.random() < self.spec.p_mutation or failures > 0:
                    # mutate more if the parents don't work well together
                    for _ in range(max(failures + 1, len(xs))):
                        xs = self._mutate(xs)
                try:
                    child = self.model(xs, io=True)
                except ModelInputException:
                    failures += 1
            children.append(child)
        self._evals += len(children)
        return tuple(children[:self.spec.population_size])

    def run(self, text_report=True):
        init_xs = tuple(self.model.random_input_vector()
                        for _ in xrange(self.spec.population_size))
        get_energy = lambda x: x.energy
        best_era = None

        report = base.StringBuilder() if text_report else base.NullObject()

        self._population = tuple(self.model.compute_model_io(xs)
                                 for xs in init_xs)

        best = min(self._population, key=get_energy)

        self._evals, lives = 0, 4

        for gen in xrange(self.spec.iterations):
            if self._evals > self.spec.iterations or lives <= 0:
                break

            prev_best_energy = best.energy

            self._population = self._breed_next_generation()

            best_in_generation = min(self._population, key=get_energy)
            best = min(best, best_in_generation, key=get_energy)

            report += str(best.energy)
            report += ('+' if x.energy < prev_best_energy else '.'
                       for x in self._population)
            report += '\n'

            energies = NumberLog(inits=(c.energy for c in self._population))
            try:
                improved = energies.better(prev_energies)
            except NameError:
                improved = False
            prev_energies = energies  # noqa: flake8 doesn't catch use above

            if improved:
                best_era = energies
            else:
                lives -= 1

        if best_era is None:
            best_era = energies

        return SearchReport(best=best.energy,
                            best_era=best_era,
                            evaluations=self._evals,
                            searcher=self.__class__,
                            spec=self.spec,
                            report=None)
```

```
from __future__ import division, print_function

import random
from itertools import izip

from searcher import Searcher, SearchReport
from witschey.base import memo_sqrt as sqrt
from witschey.log import NumberLog
from witschey.models import ModelInputException


def _random_scaled_velocity(a, b, scale=.1):
    magnitude = max(a, b) - min(a, b)
    return random.uniform(-magnitude, magnitude) * scale


class ParticleSwarmOptimizer(Searcher):
    """
    A searcher that models a "flock" of individuals roaming the search space.
    Individuals make decisions about where to go next based both on their
    own experience and on the experience of the whole group. For more
    information, see https://github.com/timm/sbse14/wiki/pso#details and
    http://en.wikipedia.org/wiki/Particle_swarm_optimization
    """

    def __init__(self, *args, **kwargs):
        super(ParticleSwarmOptimizer, self).__init__(*args, **kwargs)

        self._flock = tuple(self._new_particle()
                            for _ in range(self.spec.population_size))
        self._evals = len(self._flock)
        self._current_flock_energies = NumberLog(p.energy
                                                 for p in self._flock)
        self._best = min(self._flock, key=lambda x: x.energy)
        self._lives = 4

    def _new_particle(self):
        return Particle(self.model, self.spec.phi1, self.spec.phi2)

    def _update(self):
        self._prev_flock_energies = self._current_flock_energies

        for p in self._flock:
            p._update(self._best)
        self._evals += len(self._flock)
        self._current_flock_energies = NumberLog(p.energy
                                                 for p in self._flock)

        self._best = min(self._best, *self._flock, key=lambda x: x.energy)
        if self._current_flock_energies.better(self._prev_flock_energies):
            self._best_flock = self._flock
        else:
            self._lives -= 1

    def run(self, text_report=False):
        for i in range(self.spec.generations):
            self._update()
            if self._lives <= 0 or self._evals >= self.spec.iterations:
                break

        best_flock_energies = NumberLog(p.energy for p in self._best_flock)
        return SearchReport(best=self._best,
                            best_era=best_flock_energies,
                            evaluations=self._evals,
                            searcher=self.__class__,
                            spec=self.spec,
                            report=None)


class Particle(object):
    """
    A particle in the "flock".
    """
```



```
    def __init__(self, model, phi1, phi2):
        self._model = model
        self._current = model.random_model_io()
        self._best = self._current
        self._phi1, self._phi2 = phi1, phi2

        # calculate constriction factor
        phi = phi1 + phi2
        self._k = 2 / abs(2 - phi - sqrt(phi * phi) - 4 * phi)

        # initialize velocities
        self._velocity = tuple(_random_scaled_velocity(iv.lo, iv.hi)
                               for iv in model.xs)

    @property
    def energy(self):
        return self._current.energy

    def _compute_new_velocity(self, local_best):
        to_local = tuple(a - b
                         for a, b in izip(local_best.xs, self._current.xs))
        to_personal = tuple(a - b
                            for a, b in izip(self._best.xs, self._current.xs))
        v = tuple((self._k * (v + self._phi1 * loc + self._phi2 * pers)
                   for v, loc, pers in izip(
                       self._velocity, to_local, to_personal)))
        return v

    def _update(self, local_best):
        """
        Given local_best, the best value seen by this particle's neighbors,
        find the particle's new velocity.
        """
        init_loc = self._current.xs
        init_vel = self._velocity
        candidate_xs = tuple(p + v for p, v in izip(init_loc, init_vel))

        updated = False
        while not updated:
            try:
                self._current = self._model.compute_model_io(candidate_xs)
                updated = True
            except ModelInputException:
                candidate_xs = self._model.random_replace(candidate_xs)

        self._velocity = self._compute_new_velocity(self._best)
```

```
from __future__ import division


import random
from itertools import repeat, izip

from log import NumberLog
import texttable
from basic_stats import xtile
from witschey import basic_stats
from witschey.base import memo_sqrt


def a12(xs, ys):
    gt, eq = 0, 0
    for x in xs:
        for y in ys:
            if x > y:
                gt += 1
            if x == y:
                eq += 1

    return (gt + eq / 2) / (len(xs) * len(ys))


def a12_fast(xs, ys):
    """
    Non-parametric statistical test. Answers the question: "If you pick a
    random x from xs, and a random y from ys, what's the probability that
    x will be greater than y?"
    """

    xs_i = izip(sorted(xs, reverse=True), repeat(0))
    ys_i = izip(sorted(ys, reverse=True), xrange(len(ys), 0, -1))

    gt, eq = 0, 0

    cs, ds = xs_i, ys_i
    c, d = cs.next(), ds.next()
    while True:
        try:
            while d[0] < c[0]:
                gt += d[1]
                # gt += 1
                print(d)
                d = ds.next()
            else:
                if d[0] == c[0]:
                    eq += 1
                    d = ds.next()
                else:
                    cs, ds = ds, cs
        except StopIteration:
            break

    gt += sum(1 for _ in xs_i)

    print('gt: {}\t eq: {}'.format(gt, eq))
    return (gt + eq / 2) / (len(xs) * len(ys))


def test_statistic(y, z):
    """Checks if two means are different, tempered
     by the sample size of 'y' and 'z'"""
    delta = z.mean() - y.mean()
    sd_y = y.standard_deviation()
    sd_z = z.standard_deviation()

    if sd_y + sd_z:
        delta /= memo_sqrt(sd_y / len(y) + sd_z / len(z))

    return delta
```



```
def bootstrap(y0, z0, conf=0.01, b=1000):
    """
    The bootstrap hypothesis test from p220 to 223 of Efron's book 'An
    introduction to the boostrap.

    Simple way to describe: "If you randomly generate 1000 similar datasets,
    is a likely to be significantly different to b?"
    """
    y, z = NumberLog(y0), NumberLog(z0)
    x = NumberLog(inits=(y, z))
    observed_mean_difference = test_statistic(y, z)
    yhat = tuple(y1 - y.mean() + x.mean() for y1 in y.contents())
    zhat = tuple(z1 - z.mean() + x.mean() for z1 in z.contents())
    bigger = 0
    for i in range(b):
        # sample with replacement for yhat and zhat
        swr_yhat = (random.choice(yhat) for _ in yhat)
        swr_zhat = (random.choice(zhat) for _ in zhat)
        sampled_mean_difference = test_statistic(
            NumberLog(swr_yhat, max_size=None),
            NumberLog(swr_zhat, max_size=None))
        if sampled_mean_difference > observed_mean_difference:
            bigger += 1
    return bigger / b < conf


def different(xs, ys):
    """
    Quick test to see if 2 things are different. A12 is a reasonable first
    approximation, and fast, and if it gets past A12, run the slower, more
    authoritative, bootstrap.
    """
    return a12(xs, ys) and bootstrap(xs, ys)


def scottknott(data, max_rank_size=3, epsilon=0.01):
    """
    Recursively split data, maximizing delta of the expected value of the
    mean before and after the splits. Reject splits with under max_rank_size
    items.
    """
    flat_data = [x for log in data for x in log.contents()]
    data_mean = basic_stats.mean(flat_data)

    def recurse_and_rank(parts, rank=0):
        "Split, then recurse_and_rank on each part."

        cut = min_mu(parts, data_mean, len(flat_data), max_rank_size, epsilon)
        if cut:
            # if cut, rank "right" higher than "left"
            rank = recurse_and_rank(parts[:cut], rank) + 1
            rank = recurse_and_rank(parts[cut:], rank)
        else:
            # if no cut, then all get same rank
            for part in parts:
                part.rank = rank
        return rank

    recurse_and_rank(sorted(data, key=lambda x: x.median()))

    return data


def min_mu(parts, data_mean, data_size, max_rank_size, epsilon):
    """Find a cut in the parts that maximizes the expected value of the
    difference in the mean before and after the cut. Reject splits that are
    insignificantly different or that generate very small subsets.
    """
    cut = None
    max_delta = 0
    mrs = max_rank_size
    for i, left, right in left_right(parts, epsilon):
```

```
            if len(parts[:i]) >= mrs and len(parts[i:]) >= mrs:
                delta = len(left) / data_size * (data_mean - left.mean()) ** 2
                delta += len(right) / data_size * (data_mean - right.mean()) ** 2

                if abs(delta) > max_delta and different(parts[i-1], parts[i]):
                    max_delta, cut = abs(delta), i
        return cut


    def left_right(parts, epsilon=0.01):
        """For each item in 'parts', yield the splitting index, everything to the
        beginning (including the item) and everything to the end.
        """
        for i in range(1, len(parts)):
            if parts[i].median() - parts[i - 1].median() > epsilon:
                left = NumberLog((p for p in parts[:i]), max_size=None)
                right = NumberLog((p for p in parts[i:]), max_size=None)
                yield i, left, right


    def rdiv_report(data):
        """
        Generate a tabular report on the data. Assumes data is in lists, where the
        first element of each list is its name.
        """
        # wrap each line in a NumberLog
        data = map(lambda xs: NumberLog(label=xs[0], inits=xs[1:], max_size=None),
                   data)

        # sort by rank & median within each rank
        # sorting is stable, so sort by median first, then rank
        ranked = sorted((x for x in scottknott(data, max_rank_size=1)),
                        key=lambda y: y.median())
        ranked = tuple(sorted(ranked, key=lambda y: y.rank))

        # get high and low values for entire dataset
        lo = min(log.lo for log in data)
        hi = max(log.hi for log in data)

        # generate column names
        rows = [['rank', 'name', 'med', 'iqr', '',
                 '10%', '30%', '50%', '70%', '90%']]

        # generate rows
        for x in ranked:
            # each row starts with 'rank label, median, iqr'
            next_row = [x.rank + 1]
            next_row.append(x.label + ',')
            next_row.append('{0:0.2},'.format(float(x.median())))
            next_row.append('{0:0.2}'.format(float(x.iqr())))

            # get xtile: '( -* | -- ) ##, ##, ##, ##, ##'
            xtile_out = xtile(x.contents(), lo=lo, hi=hi, width=30, as_list=True)
            # xtile is displayed as the whisker plot, then comma-separated values
            row_xtile = [xtile_out[0]]
            # don't use 'join', we want each to be its own list element
            row_xtile.extend(map(lambda x: x + ',', xtile_out[1:-1]))
            row_xtile.append(xtile_out[-1])

            next_row.extend(row_xtile)
            rows.append(next_row)

        table = texttable.Texttable(200)
        table.set_precision(2)
        table.set_cols_dtype(['t', 't', 't', 't', 't', 't', 't', 't', 't', 't', 't'])
        table.set_cols_align(['r', 'l', 'l', 'r', 'c', 'r', 'r', 'r', 'r', 'r', 'r'])
        table.set_deco(texttable.Texttable.HEADER)
        table.add_rows(rows)
        return table.draw()
```