SDS Library

Andrew Owen Martin [a.martin@gold.ac.uk]

18th December 2017

1 Introduction

Stochastic Diffusion Search (SDS) is a swarm intelligence algorithm. It has many interesting features, most significantly its simplicity, and the speed of its performance over suitable problems.

A suitable problem is search problem where the value of each solution can be estimated from the value of a number of quick and simple tests. The quicker and simpler the tests, the better. SDS is very tolerant of noise, so it tends to be more suitable in real-world problems rather than theoretical problems.

An intuitive example is locating the string `'hello'` in a larger text, each location in the text is a potential solution and can be evaluated by the results of these five tests:

1. There is an `'h'` at the chosen location
2. There is an `'e'` one letter along from the chosen location
3. There is an `'l'` two letters along from the chosen location
4. There is an `'l'` three letters along from the chosen location
5. There is an `'o'` four letters along from the chosen location

To define an SDS using this library you largely just need to define a function which takes no input and returns a random hypothesis (a potential solution), and define a list of functions each of which take a hypothesis and return the result of a quick and simple test.

To define an SDS to perform the example task you'd therefore need something like these functions.

1.1 Random Hypothesis Function

This function takes a random generator which can be an instance of the Random class, or a reference to the random module, or anything that performs `randint`, and returns an integer which represents a hypothesised location of the model in the larger text.

```
larger_text = 'xxxhelxxxelloxxxxxxhelloxxxxx'

model = 'hello'

def random_hypothesis(random_obj):

    return random_obj.randint(0, len(larger_text)-len(model))
```

Of course the `larger_text` variable can be much larger than the tiny string used here, by using a `file` object and the `seek` method, text files representing entire genomes have been successfully searched with SDS.

1.2 List of microtest functions

This defines `microtests`, a list of functions which each take a hypothesis generated by `random_hypothesis` and returns the result of a simple test.

```
microtests = [
    lambda hyp: larger_text[hyp] == 'h',
    lambda hyp: larger_text[hyp+1] == 'e',
    lambda hyp: larger_text[hyp+2] == 'l',
    lambda hyp: larger_text[hyp+3] == 'l',
    lambda hyp: larger_text[hyp+4] == 'o',
]
```

These functions have been defined manually, but Python provides a number of ways to automatically define functions for more complex tasks. Even this example could have been produced by looping over a single function which returns test functions like the ones defined here.

1.3 Initialising a swarm

With the random hypothesis function defined and the list of microtests defined all that is required is to define a swarm, a swarm is simply a list of `Agents` and `Agents` are simple data structures which each maintain a single hypothesis and a boolean state variable which defines whether or not they are `active`.

```
import sds

agent_count = 1000

swarm = sds.Agent.initialise(agent_count=agent_count)
```

After this code runs `swarm` will be a list of 1000 inactive agents with their hypothesis uninitialised.

1.4 Choosing a Diffusion function

The SDS library implements three diffusion functions, Passive, Context-Free and Context-Sensitive. You should experiment with their different behaviours. When you need to choose a diffusion function use one of `sds.passive_diffusion`, `sds.context_free_diffusion`, or `sds.context_sensitive_diffusion`. For now stick with Passive Diffusion.

1.5 Running SDS

The entry point for running SDS is the `run` function, the required arguments are as follows:

`swarm` Defined in the third section.

`microtests` Defined in the second section.

`random_hypothesis_function` Defined in the first section.

`max_iterations` The max number of iterations of SDS to run. `None` means loop until halted.

`diffusion_function` The diffusion function to employ.

`random` A random generator. (This will likely be made optional in later versions)

The `run` function returns its results as a `collections.Counter` of clusters, which details where the active agents have decided to congregate.

```
clusters = sds.run(
    swarm,
    microtests,
    random_hypothesis,
    max_iterations=100,
    diffusion_function=sds.passive_diffusion,
    random,)

print(clusters.most_common(1))
```

There are other functions available but these are the main ones.

1.6 A full runnable example

The combination of `make_microtest` function and the `microtests` list comprehension is equivalent to the five manually defined functions in the Section "List of microtest functions".

In the `search_space` of this example are four intances of four of the five letter in the word `'hello'` appearing in sequence, the locations marked with a `|` in the code. Play around with the `search_space` variable, and see how the results change.

```
import random
import sds

search_space = "xxhellxelloxhexhelxoxxxhxlloxxx"
#                 |   |        |       |

model = "hello"

def random_hyp(rnd):
    return rnd.randint(0,len(search_space)-len(model))

def make_microtest(offset):
return lambda hyp: search_space[hyp+offset] == model[offset]

microtests = [
    make_microtest(offset) for offset in range(len(model))
]

swarm = sds.Agent.initialise(agent_count=1000)

clusters = sds.run(
swarm=swarm
microtests=microtests,
random_hypothesis_function=random_hyp,
max_iterations=300,
diffusion_function=sds.passive_diffusion,
random=random.Random(),
report_iterations=10,
)

print(clusters.most_common())
```

Running this script should produce something similar to the following

```
   0 Activity: 0.184. 23:  32, 6:  29, 15:  28, 2:  22, 12:  20, 3:  14, 5:  11, 14:   7, 24:   7
  10 Activity: 0.763. 6: 216, 2: 208, 23: 171, 15: 164, 1:   2, 3:   1, 24:   1
  20 Activity: 0.766. 2: 232, 23: 200, 6: 183, 15: 148, 5:   1, 12:   1, 14:   1
  30 Activity: 0.780. 2: 223, 6: 213, 23: 187, 15: 152, 3:   2, 12:   2, 22:   1
  40 Activity: 0.745. 6: 185, 23: 185, 2: 184, 15: 184, 14:   2, 1:   1, 3:   1, 22:   1, 24:   1
  50 Activity: 0.768. 15: 213, 2: 198, 6: 197, 23: 157, 12:   2, 5:   1
  60 Activity: 0.772. 2: 213, 15: 205, 6: 179, 23: 169, 12:   2, 1:   1, 22:   1, 7:   1, 24:   1
  70 Activity: 0.777. 15: 230, 2: 218, 6: 179, 23: 146, 1:   2, 24:   1, 12:   1
  80 Activity: 0.768. 15: 240, 2: 199, 23: 178, 6: 146, 22:   1, 24:   1, 7:   1, 12:   1, 14:
  90 Activity: 0.748. 15: 272, 2: 191, 23: 143, 6: 134, 1:   3, 3:   2, 7:   1, 12:   1, 14:   1
 100 Activity: 0.780. 15: 285, 2: 199, 23: 164, 6: 124, 24:   2, 14:   2, 3:   1, 22:   1, 7:   1
 110 Activity: 0.766. 15: 280, 2: 193, 23: 145, 6: 139, 12:   3, 1:   2, 3:   2, 5:   1, 22:   1
 120 Activity: 0.757. 15: 273, 23: 169, 6: 163, 2: 146, 14:   3, 3:   1, 5:   1, 22:   1
 130 Activity: 0.766. 15: 282, 23: 172, 6: 163, 2: 146, 24:   1, 7:   1, 12:   1
```

```
140 Activity: 0.724. 15: 262, 6: 159, 23: 156, 2: 144, 12:   2, 7:   1
150 Activity: 0.741. 15: 269, 23: 171, 2: 164, 6: 129, 1:   3, 12:   3, 22:   1, 14:   1
160 Activity: 0.775. 15: 260, 2: 180, 23: 169, 6: 161, 12:   3, 22:   1, 5:   1
170 Activity: 0.754. 15: 251, 2: 186, 23: 176, 6: 134, 12:   3, 1:   1, 3:   1, 22:   1, 7:   1
180 Activity: 0.750. 15: 234, 23: 186, 2: 174, 6: 154, 7:   1, 24:   1
190 Activity: 0.751. 15: 251, 23: 181, 2: 159, 6: 155, 12:   2, 1:   1, 5:   1, 24:   1
200 Activity: 0.746. 15: 241, 23: 192, 6: 175, 2: 135, 14:   2, 12:   1
210 Activity: 0.749. 15: 214, 6: 204, 23: 181, 2: 146, 1:   1, 3:   1, 24:   1, 7:   1
220 Activity: 0.763. 6: 207, 15: 196, 23: 191, 2: 164, 24:   2, 5:   1, 12:   1, 14:   1
230 Activity: 0.756. 2: 192, 6: 192, 23: 182, 15: 182, 14:   4, 12:   2, 22:   1, 7:   1
240 Activity: 0.750. 2: 222, 6: 181, 23: 173, 15: 170, 12:   2, 3:   1, 7:   1
250 Activity: 0.771. 2: 206, 23: 200, 6: 184, 15: 178, 1:   1, 22:   1, 7:   1
260 Activity: 0.753. 2: 194, 23: 192, 15: 187, 6: 174, 7:   2, 12:   2, 22:   1, 14:   1
270 Activity: 0.744. 15: 214, 23: 185, 2: 179, 6: 154, 5:   5, 1:   2, 12:   2, 24:   1, 22:   1
280 Activity: 0.767. 15: 214, 6: 192, 2: 176, 23: 174, 14:   3, 5:   2, 22:   2, 12:   2, 7:   1
290 Activity: 0.772. 2: 214, 15: 188, 6: 184, 23: 181, 1:   2, 3:   1, 5:   1, 14:   1


[(2, 195), (15, 191), (23, 190), (6, 186), (22, 2), (3, 1), (5, 1), (24, 1), (12, 1)]
```

Ignoring the activity lines for now, each of the tuples in the list at the bottom represent a cluster, the left number is the location of the cluster and the right numeber is the number of active agents at that location, the list is ordered with the largest clusters first, so `(2, 195)` means 195 agents are at location 2, and being the first tuple in the list means this must be the largest cluster.