Abstract

Keywords: calibration profile, model evaluation, agent based modelling

A new method to evaluate simulation models: The Pattern Probability Space Estimation (PPSE) algorithm

July 19, 2021

1 The Pattern Probability Space Estimation (PPSE) algorithm

1.1 Objective

The PPSE method is inspired from the evolutionary algorithm. It extends the PSE method (citation). The PSE method discovers all the pattern that could be produced by a model given some uncertainty on its parameters. In PSE the pattern space is defined as discrete space containing descriptors of the dynamics of the simulation. Each combination values of these descriptors is considered a distinct pattern.

PSE discovers all the possible patterns, however it does not provide any additional information on these pattern apart from there existence. PPSE extends on that to compute a likelihood for each pattern. This likelihood is the defined as the likelihood that the model produce this pattern given that the input parameters are drawn from a uniform distribution.

1.2 Naive algorithm

Lets consider a simulation model and a function that computes patterns from the dynamics produced by the model. The model has some free parameters. For the sake of simplicity we consider here that the input parameters of the model are defined on $[0, 1]$.

Listing 1: Model and pattern signatures

```
1  type Dynamic = ...
2  type Pattern = Vector[Int]
3
4  class Model(compute: Vector[Double] => Dynamic, dimension: Int)
5
6  def simulation(model: Model): Dynamic = ...
7  def pattern(dynamic: Dynamic): Pattern = ...
```

The naive algorithm to compute the likelihood of the pattern is to uniformly sample the parameter space and to measure the number of time a given pattern has been reached.

Listing 2: Naive PPSE

```
1
2  def naivePPSE(model: Model, pattern: Dynamic => Pattern, random:
       Random, samples: Int): Map[Pattern, Double] =
3    var patterns = Map[Pattern, Double]()
4
5    for
6      i <- 0 to samples
7    dor
8      val x = Vector.fill(model.dimension)(random.nextDouble())
9      val p = pattern(model.compute(x))
10     patterns = patterns.getOrElse(p, 0) + 1
11
12   val total = patterns.values.sum
13   patterns.mapValues(x => x / total)
```

This algorithm computes an approximation of the probability to reach a pattern given a uniform sampling of the parameter space. This method is however highly inefficient for model in which a large variablity of the dynamics is generated by a very small faction of the input space. This is generally the case in complex systems models. As we have shown in the PSE paper sampling uniformly is, for instance, very inefficient to discovers new patterns in the case of the Reynolds Flocking model.

The idea behind the PPSE algorithm is to evolutionally bias the sampling of the parameter space towards the part generating rare pattern, while keeping the enough knoledge on this bias to be de-bias the results and obtain unbiased estimator of the pattern likelihoods.

1.3 Description of the algorithm

The algorithm is composed of the following steps:

Initialization. We construct an initial set of parameter vectors.

Step 0. We first generate a random set X_0 of parameter vectors, that is $X_0 = \{\bar{a}^1, \ldots, \bar{a}^M\}$, with $\bar{a}^j = (a_1^j, \ldots, a_n^j)$ and $a_i^j \in A_i$. The number M may be much higher than l_k.

Approximation loop. We construct X_{N+1} from X_N.

Step 1. We construct the sets $S_i = \{\bar{a} \in X_N : \mathfrak{Cat}(\bar{a}) = C_i\}$ for $i = 1, \ldots, l_k$ (we may get $S_i = \emptyset$).

Step 2. Here we apply elitism, that is we select only the best parameter vector in each category: We construct the sets $E_i = \{\bar{a} \in S_i : f(\bar{a}) \leq f(\bar{a}')\ \forall \bar{a}' \in S_i\}$ for $i = 1, \ldots, l_k$, and $H = \bigcup_{i=1}^{l_k} E_i$. In practice all the sets E_i will be either empty (if $S_i = \emptyset$) or composed by only one parameter vector; if there exist more more than one that are minimal in S_i, then we can randomly choose a fixed number of them to compose E_i.

Step 3. Generate H^* from H by randomly modifying the parameter vectors in H. We can for instance generate each vector in H^* by applying a gaussian variation to a vector in H.

Step 4. Define the set X_{N+1} as the union of H and H^*. Check if a predefined stopping criterion is met (number of iterations, stabilization of the solutions, etc.): Go to Step 5 if it is, iterate Steps 1 to 4 if is not.

Ending. When the stopping criterion is met, we can output our approximation to the function $h_k(x)$.

Step 5. If $\{\bar{a}^1, \ldots, \bar{a}^m\}$ is the last H computed, we define our approximation to $h_k(x)$ as the set of 2-tuples $\{(\bar{a}_k^1, f(\bar{a}^1)), \ldots, (\bar{a}_k^m, f(\bar{a}^m))\}$. Let us note that by construction $m \leq l_k$, but in practice, when the number of iterations is high enough (see section 1.4), we get $m = l_k$.

1.4 Validity

1.5 Heuristic improvement

1.6 Generic implementation

2 Application the SimpopLocal geographical model

2.1 A model to simulate the emergence of structured urban settlement system

2.2 Agents, attributes, and mechanisms of the model

2.3 Objective function

2.4 Results

Conclusion