Performance Evaluation and Applications

2022-2023

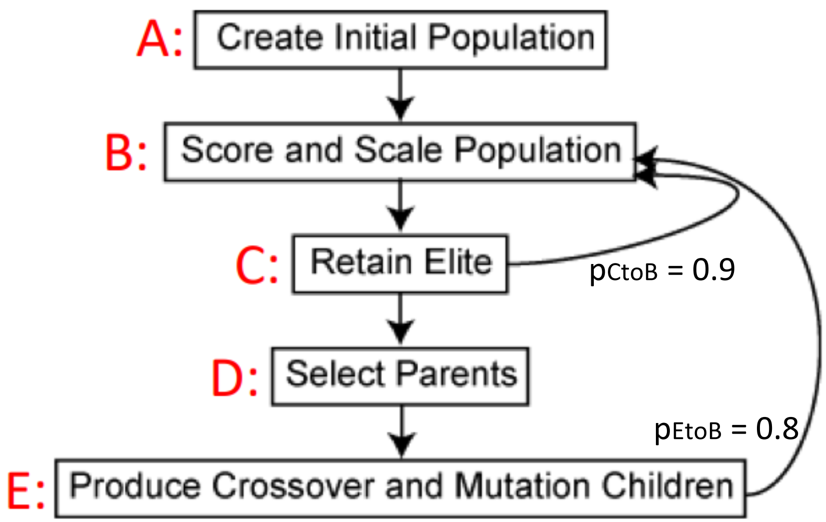
Professor Marco Gribaudo

PROJECT TYPE A

Elisa Servidio

10544789

The problem:
Performance of a Genetic Algorithm



Runtime of execution of the stages:

TraceA-A	TraceA-B	TraceA-C	TraceA-D	TraceA-E
0.71087795	0.11064513	0.04675949	0.25993050	0.21122156
0.13224217	9.73066332	0.27461941	0.19036620	1.38127096
0.23755462	17.59158162	0.03235590	0.22025383	0.43908806
0.30897996	5.72131182	0.03841262	0.01742415	0.25656726
0.57796399	1.30353606	0.06856267	0.09234362	0.02876801
0.32314768	7.02870473	0.06892380	0.41048447	0.06188163
0.18991331	12.62322785	0.50940406	0.08319836	0.73593947
1.70765743	3.31312406	0.13868625	0.07541423	0.17154645
0.00150048	1.75164841	0.13747957	0.23707276	0.94890661
0.17232313	14.78476658	0.04880492	0.05553179	0.07704857
0.20178501	5.72940229	0.29947072	0.08170527	0.26884314
0.65458747	9.41396964	0.00754012	0.26257350	0.31984218
0.64865542	9.89084876	0.01091985	0.06112535	0.09634073
0.83854821	7.78355214	0.00213362	0.12316024	0.73047864

- **multi-core machine**
- stages B and E can be **fully parallelized**

The problem:
Performance of a Genetic Algorithm

*What is the **minimum number of cores** required to provide an average of more than 30 solutions per second?*

*Which would be the **average utilization** of the CPU in this case?*

The solution:
PHASE 1 – Data preparation

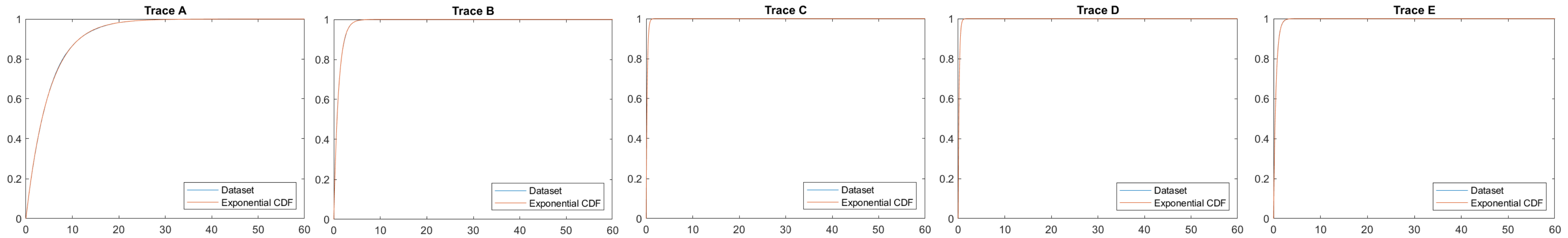
Data fitting

- Analysis of coefficient of variation *cv*

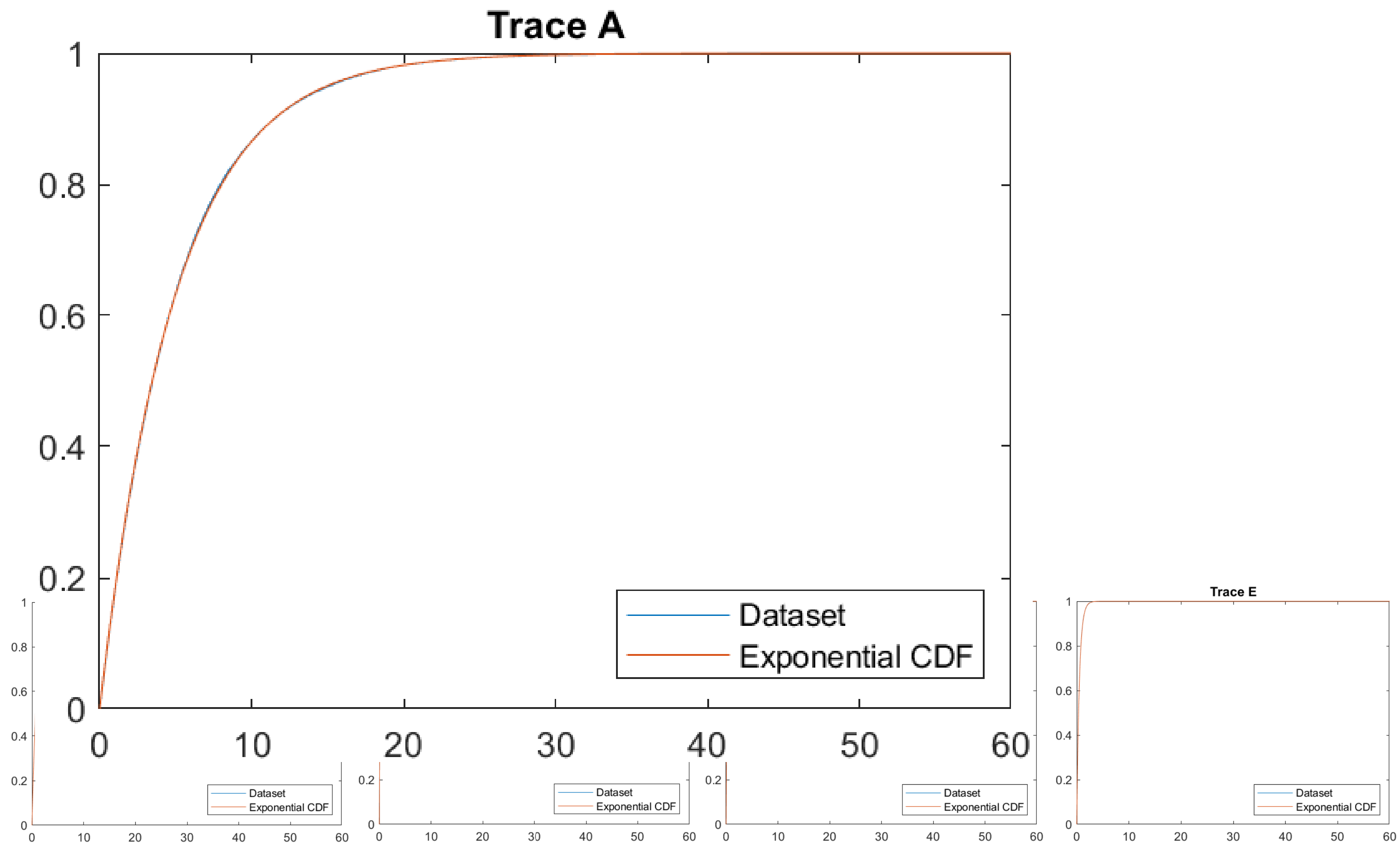
```
Check if cv almost equal to 1:
    1.0007    1.0004    1.0056    1.0019    1.0016
```

EXPONENTIAL DISTRIBUTION

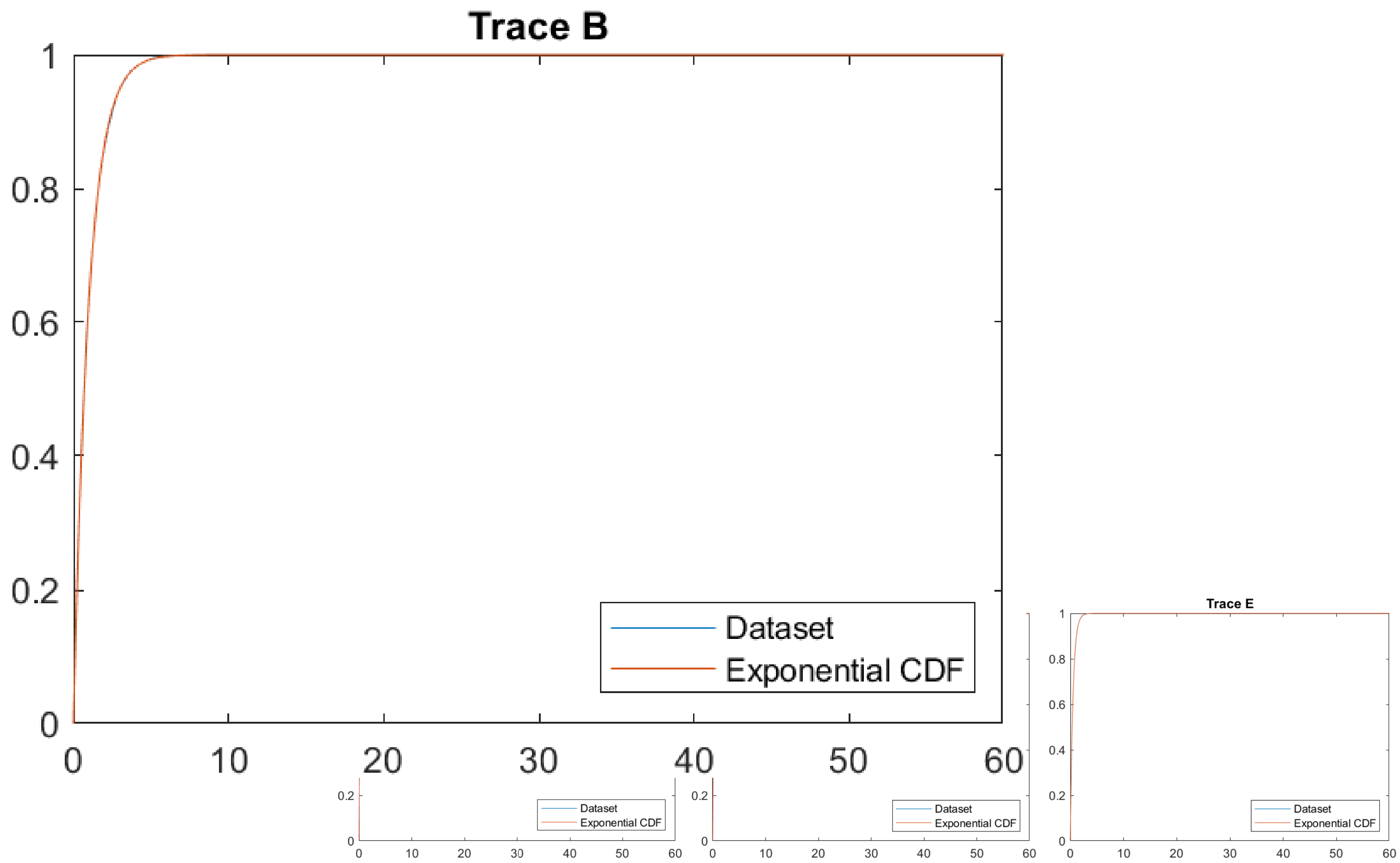
- Computation of *lambda* parameters
- Generation of exponential distribution fitting the data



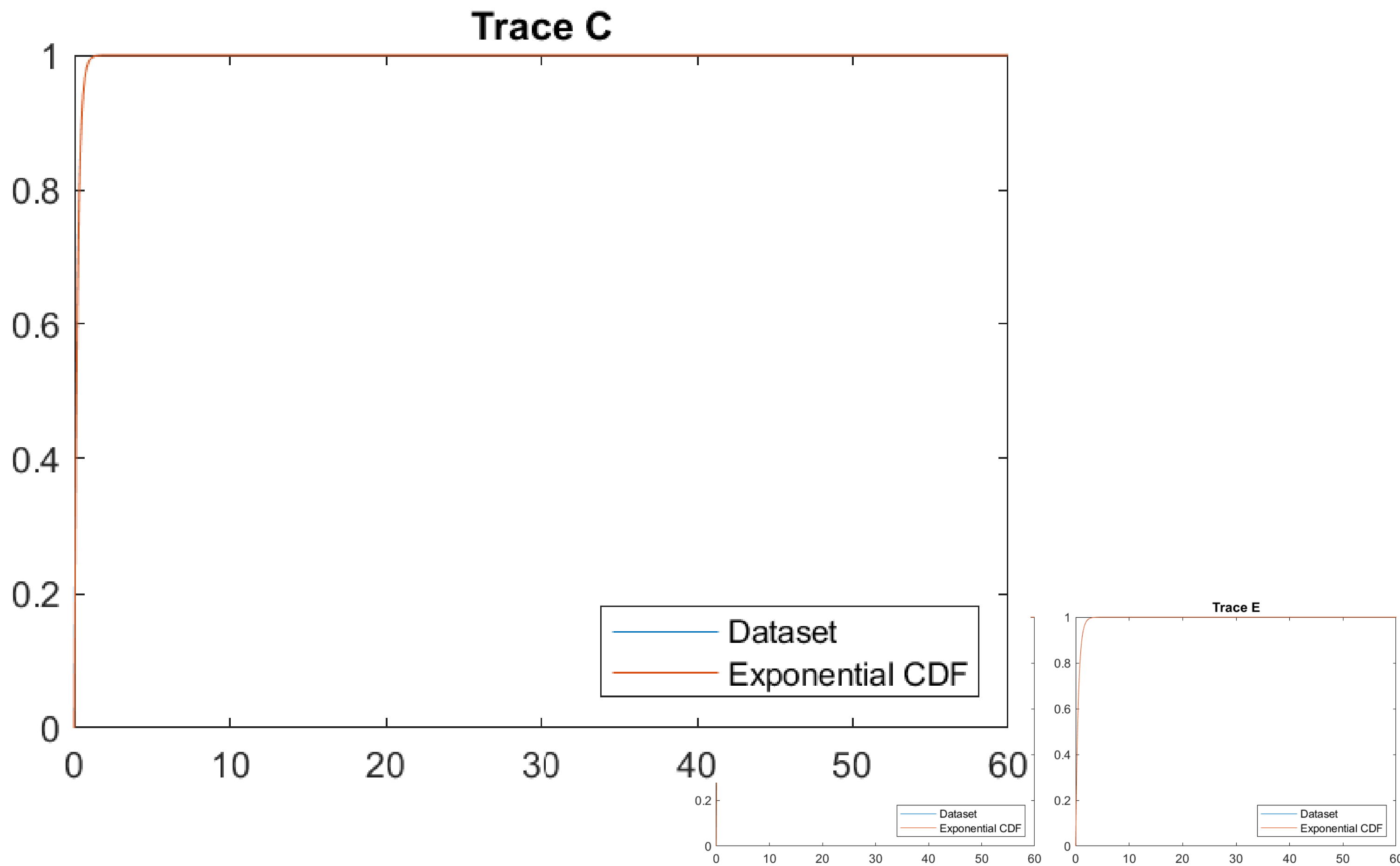
The solution:
PHASE 1 – Data preparation



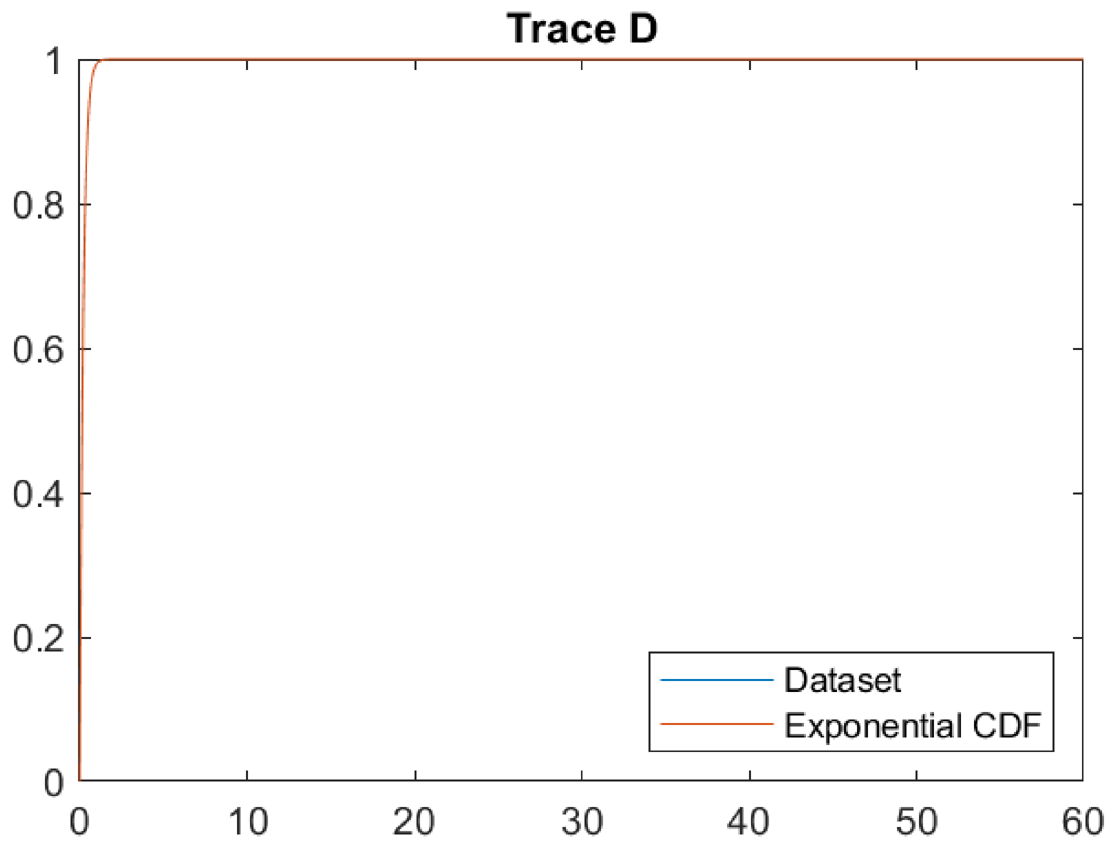
The solution:
PHASE 1 – Data preparation

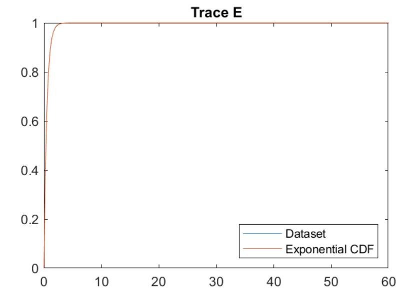


The solution:
PHASE 1 – Data preparation

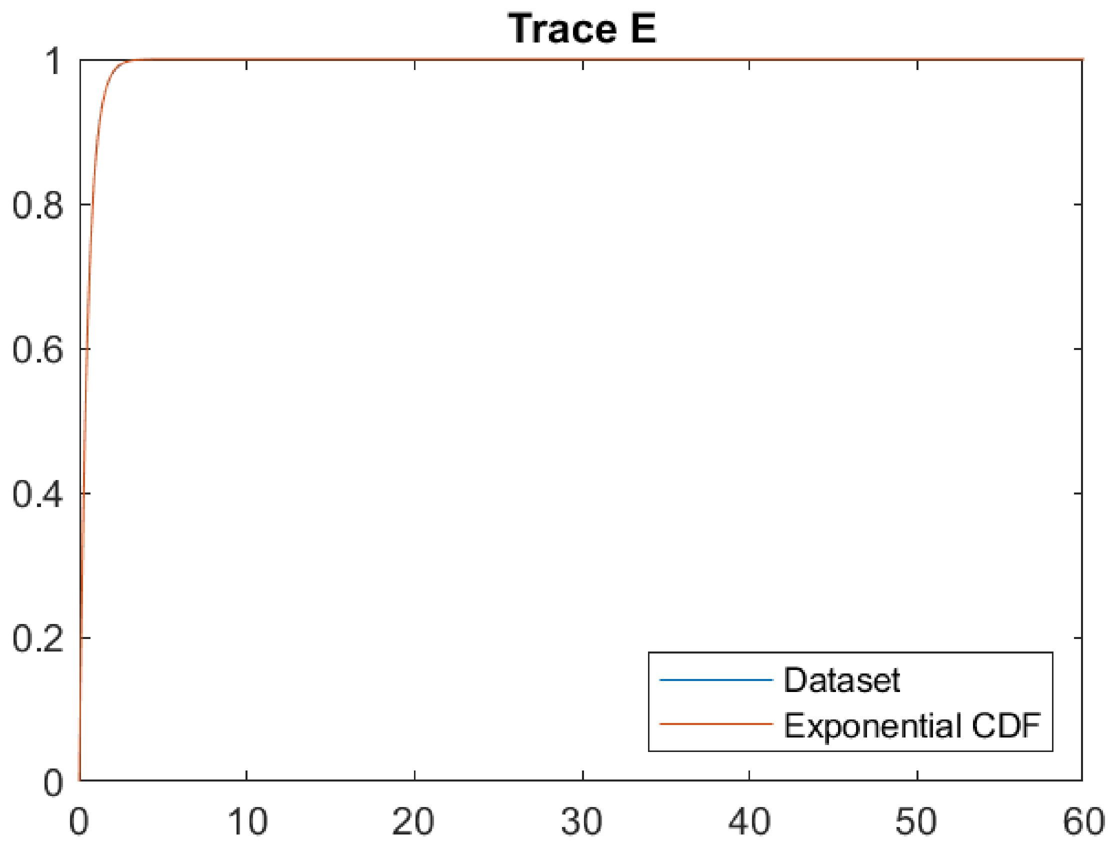


The solution:
PHASE 1 – Data preparation





The solution:
PHASE 1 – Data preparation



The solution:
PHASE 2 – State Machine based model

Generation of high-level algorithm to produce a trace

- Creation of *State Machine based model* of the system
- Generation of the corresponding *trace*
- *Simulation* of the system to retrieve the *number of cores* required to achieve a throughput ≥ 30

The solution:
PHASE 2 – State Machine based model

Throughput computation

- Analysis of throughput with different number of cores

```
%first simulation is computed to retrieve the number of cores corresponding
%to throughput >= 30
core = 1; %number of cores
throughput = 0; %system throughput
while throughput < 30
    res = state_machine_model(core,lambda);
    throughput = res(2);
    core = res(1);
     if (throughput < 30)
        core = core + 1;
    end
end
```

- Completion variable *sol* is updated in the code that handles final state E

```
if s == 5   %state E: produce crossover and mutation children
    dt = (-log(rand())/lambda(1,5))/ core;
    multiCoreBusyTime = multiCoreBusyTime + dt*core;
    r = rand();
    if r <= 0.8
        ns = 2; %next state B
    else
        ns = 1; %next state A
        sol = sol + 1; %one solution completed
    end
end
```

run time perfectly divided by the number of cores

The solution:

PHASE 2 – State Machine based model

Results

```
Minimum number of cores required to provide an average of more than 30 solutions per second:
     4
```

Throughput and utilization depend on *dt*, randomly generated from exponential distribution

```
dt = -log(rand())/lambda(1,1);
dt = (-log(rand())/lambda(1,2))/ core;
dt = -log(rand())/lambda(1,3);
dt = -log(rand())/lambda(1,4);
dt = (-log(rand())/lambda(1,5))/ core;
```

CONFIDENCE INTERVALS

The solution:
PHASE 3 – Throughput

- The system is simulated again to compute values of throughput given the number of cores just retrieved

- Computation of *throughput*
  - *Performance counters are updated in the code that handles each state*

```
K = 100;
for k=1:K
    res = state_machine_model(core,lambda);
    Throughput = res(2);
    X_value(k,1) = Throughput;
    singleCoreBusyTime = res(3);
    multiCoreBusyTime = res(4);
    multiCoreTime = res(5);
    Utilization_value(k,1) = (singleCoreBusyTime + multiCoreBusyTime)/multiCoreTime;
end
```

The solution:
PHASE 3 – Throughput

- Computation of 95% confidence interval of *throughput*
  - percentile of the standard normal distribution set to *1.96*

```
X_min = mean(X_value) - d_gamma * sqrt(var(X_value)/K);
X_max = mean(X_value) + d_gamma * sqrt(var(X_value)/K);
```

The solution:
PHASE 3 – Throughput

Results

```
Throughput confidence interval (95%) with 4 cores:
[34.3107,34.3737]
```

The solution:
PHASE 4 – Average Utilization

Computation of the average utilization considering 4 cores

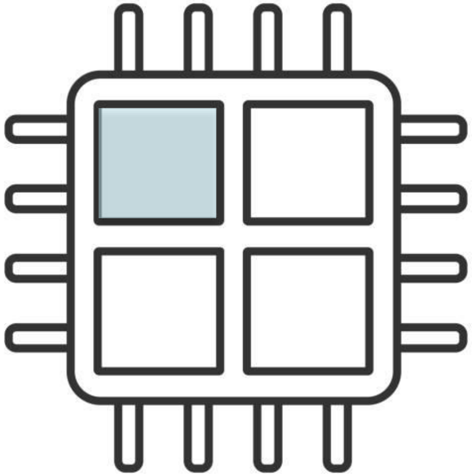
- $Utilization = \frac{Busy\ time}{time}$
- At each state, $Busy\ time$ and $time$ are computed
- Time is computed considering all cores

```
            end
            s = ns; %update current state
            t = t + dt;
            multiCoreTime = multiCoreTime + dt*core;
        end
        solution = sol/(t/1000); %solutions per seconds (throughput) (t from ms to s)
    end
```

The solution:
PHASE 4 – Average Utilization

- For non-parallelized stages A, C, D
  - *singleCoreBusyTime*: busy time of only one core

```
if s == 1   %state A: create initial population
    dt = -log(rand())/lambda(1,1);
    ns = 2; %next state B
    singleCoreBusyTime = singleCoreBusyTime + dt;
end
```

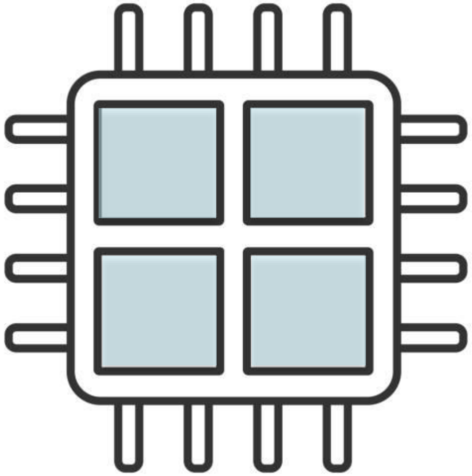


Only one core is busy *dt*

The solution:
PHASE 4 – Average Utilization

- For parallelized stages B,E
  - *multiCoreBusyTime*: busy time distributed among all cores

```
if s == 2   %state B: score and scale population
    dt = (-log(rand())/lambda(1,2))/ core;
    ns = 3; %next state C
    multiCoreBusyTime = multiCoreBusyTime + dt*core;
end
```



All cores are busy *dt*

The solution:
PHASE 4 – Average Utilization

- Given the number of cores, through the simulation of the system values of *utilization* are computed

```
K = 100;
for k=1:K
    res = state_machine_model(core,lambda);
    Throughput = res(2);
    X_value(k,1) = Throughput;
    singleCoreBusyTime = res(3);
    multiCoreBusyTime = res(4);
    multiCoreTime = res(5);
    Utilization_value(k,1) = (singleCoreBusyTime + multiCoreBusyTime)/multiCoreTime;
end
```

The solution:

PHASE 4 – Average Utilization

- Computation of 95% confidence interval of *utilization*
  - percentile of the standard normal distribution set to *1.96*

```
U_min = mean(Utilization_value) - d_gamma * sqrt(var(Utilization_value)/K);
U_max = mean(Utilization_value) + d_gamma * sqrt(var(Utilization_value)/K);
```

The solution:
PHASE 4 – Average Utilization

Results

```
Average utilization confidence interval (95%) of the CPU:
[0.58825,0.58845]
```

POLITECNICO
MILANO 1863