Fast SimEDaPE: Simulation Estimation by Data Patterns Exploration

Francisco Wallison Rocha[1], **Emilio Francesquini**[2], **Daniel Cordeiro**[1]

[1] Escola de Artes, Ciências e Humanidades – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

{wallison.rocha, daniel.cordeiro}@usp.br

[2]Centro de Matemática, Computação e Cognição – Universidade Federal do ABC (UFABC)
Santo André – SP – Brasil

e.francesquini@ufabc.edu.br

Abstract. *In the context of smart cities, solving problems such as pollution, congestion, and public transport, regularly faced by large cities like São Paulo, is not trivial. To tackle those problems researchers often rely on simulations. An example of a smart city simulator is InterSCSimulator, which simulates urban traffic. However, this simulator has limitations regarding its performance in large scale scenarios. SimEDaPE, a technique used to improve simulation performance based on the recurrence of patterns from previous simulations, was proposed in this context. SimEDaPE is still under active development and as such has some performance bottlenecks in some stages, such as the temporal mapping stage. In this work, we propose an improvement to this step of SimEDaPE using optimized libraries (written in C instead of Python), and parallelism. As a result, we obtained a considerable relative performance of 156x, running on 8 cores compared to the reference sequential implementation.*

1. Introduction

In the context of smart cities, urban mobility is one of the most important aspects both for citizens and government entities. Urban mobility is directly related to problems such as pollution, heavy urban traffic and insufficient public transportation. Currently, large cities such as São Paulo, Paris, New York, among others struggle to deal with these problems [Santana et al. 2017].

Solving these problems often require innovative solutions. Before deploying these solutions, however, they must be evaluated. A technique of great value in smart cities is simulation. Simulation allows testing these solutions at a fraction of the cost that would be needed to field test them. The InterSCSimulator is an urban traffic simulator used in this context. It simulates city traffic for a set of trips (origin-destination) including cars, bicycles, buses, subways, and pedestrians [Santana et al. 2017]. However, the InteSCSimulator has some performance limitations for big scenarios, such as the ones presented by a city as big as São Paulo. Indeed, the use of this simulator in large-scale conurbation scenarios can be challenging due to the number of computational resources required for running more extensive simulations. The Simulation Estimation by Data Patterns Exploration (SimEDaPE) was proposed to improve the InterSCSimulator performance [Rocha et al. 2021].

SimEDaPE aims to find and exploit the large-scale behavior of urban traffic flows of simulation. After a single complete execution of the simulation, execution-representative patterns (called *simulation points*) are identified and reused in future executions. These Simulation Points are represented by vehicle traffic flow in the streets (represented by time series). Simulation Points are then selected using a clustering algorithm. The algorithm identifies the patterns and which Simulation Points represent parts of the simulation. In addition to clustering, SimEDaPE also performs a mapping step between the simulation points and remaining time series of the cluster for later use in estimating new similar simulations. With the pre-processing steps completed, it is possible to estimate a new partial simulation similar to the first one, using the latest simulation points corresponding to those extracted in the pre-processing with the mapping obtained in the pre-processing step. Thus, without running a complete simulation, SimEDaPE can speed up the execution of the simulation as a whole [Rocha et al. 2021].

Although SimEDaPE allows for a faster execution (as compared to the regular InterSCity simulator), the clustering and mapping steps can still be a bottleneck for the simulation of large scale scenarios. This is due to some implementation details such as language compiler/interpreter among others. In this work we show how we were able to obtain an important performance gain in the mapping step (which is the most time-consuming). For that, we employed multithreading and faster implementations of the mapping step using Cython.

2. SimEDaPE: Simulation Estimation by Data Patterns Exploration

One of the goals of smart city simulations is to extract metrics from the simulation to understand the behavior of a city according to a given scenario. We can use different metrics for traffic simulations such as average vehicle speed, distance driven, or percentage of occupation of roads by vehicles. These metrics can be extracted from the vehicle flows in the streets represented through time series.

InterSCSimulator is one such smart city simulator. It simulates the urban traffic of a city like Sao Paulo. However, due to the large amount of computational power required to run large scenarios in large cities like Sao Paulo, regular simulators can be challenging to use. The Simulation of Estimation by Exploration of Data Patterns (SimEDaPE) was proposed [Rocha et al. 2021] as a possible solution to this problem. SimEDaPE uses pattern recurrence, obtained from previous simulations, to estimate new similar simulations without running them completely. In SimEDaPE, behaviors are represented by time series. The time series is formed by road vehicles' entry and exit events in a specific time interval. Due to the use of time series to represent the behavior of the simulation, for greater precision, SimEDaPE uses the K-shape algorithm to compare time series [Paparrizos and Gravano 2015]. In addition to performing clustering to identify patterns and select a representative (the simulation point) for each cluster, SimEDaPE uses a Dynamic Time Warping (DTW) [Berndt and Clifford 1994] implementation of the Dtaidistance[1] library (written entirely in Python) to perform a mapping between the simulation points and the other time series in the cluster for later use in estimating new simulations.

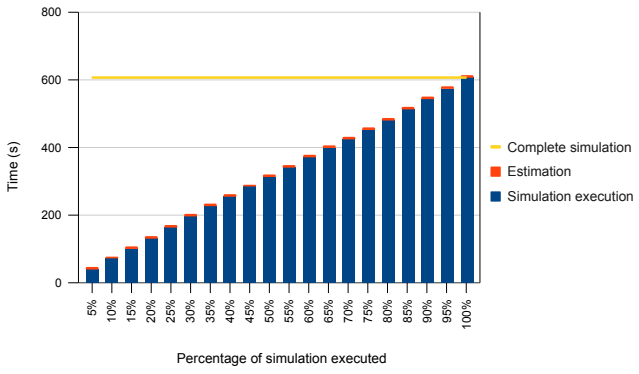
Then, after performing clustering using K-shape with a proper distance to com-

[1]https://pypi.org/project/dtaidistance/

pare time series, Shape Based Distance (SBD), the next step is to determine the simulation points. In SimEDaPE, the simulation points are represented by the centroids of each simulation cluster, similar to what has been done for hardware simulations [Hamerly et al. 2005]. The centroid in the cluster is the element (time series) with the shortest distance to the other time series of the cluster. SimEDaPE allows you to run partial simulations with a maximum error bellow 30% for metrics like average speed [Rocha et al. 2021]. Finally, to estimate a time series from the simulation point, the warping path is calculated using DTW. The warping path (WP) is the element-to-element association with the shortest distances between two series. WP is used to map the simulation points in the time series of each cluster. In the new simulations to be estimated, this same WP is used to temporally distort the simulation point in each time series [Rocha et al. 2021]. However, the warping path calculation for the mapping and warping of simulation points in the time series has complexity of $O(m^2)$, where m is the number of points in the time series, making it one of the most computationally expensive steps of the whole process.

Figure 1 shows the execution time for each step of SimEDaPE. In blue the time for partial execution of the scenario, in orange the time to estimate the rest of the simulation, in yellow the time to execute the complete scenario. In addition to the steps shown in the Figure 1 , two additional pre-processing steps taking $159.31s$ for the clustering step and $9511s$ for the mapping or warping path calculation step. These two pre-processing steps, however, only need to be run once, since all future similar simulations can reuse the pre-processed data. Therefore, from this perspective, we see that the mapping calculation time for this experiment corresponds to 92.54% of the total time spent in the worst case (run a full simulation) of SimEDaPE. In view of the above, our work proposes using an optimized library version written using Cython instead of the pure Python implementation. Furthermore, to take full advantage of the machine's resources, this work also leverages parallelism to relative performance this step.

Figure 1. Time spent on each step of the SimEDaPE application.



3. Improvements and Experimental Results

Here we show three approaches to improving the mapping step implemented internally in Python using the dtaidistance library. All three approaches start from using the dtaidistance library using Cython. The first is a sequential approach. The second and third are parallel approaches using the Joblib library, with the difference that the second uses disk cache saving of the information during processing. In the parallel approaches, each process was responsible for calculating a complete cluster.

To determine the performance of each of the approaches compared to the current implementation, we performed four experiments to calculate the warping path (mapping). This experiment was done using a dataset with 864 time series of size 3403 data points, distributed into 64 clusters. This dataset was obtained from a simulation of 150000 trips in a region of 36 roads. The machine used to run the experiment has 64 GB RAM and an Intel Core i7-9700K CPU 3.60 GHz octa-core processor, and for parallel implementations 8 cores were used. Table 1 shows the current implementation time and the time for the 3 new implementations using dtaidistance using Cython, one sequential and two parallel (one using cache-to-disk save and one without). In addition, the speedups of the new approaches compared to the current one are shown.

Table 1. Relative performance compared to current implementation.

Implementation	**Time**	**Relative Performance**
Current	$2.353s$	-
Cython Sequential	$46s$	51x
Cython Parallel (disk cache saving)	$65s$	36x
Cython Parallel	$15s$	156x

4. Conclusions

This work showed three implementations to improve the performance of the mapping step of the SimEDaPE technique. Improving this step is crucial because it takes time to run not-so-large scenarios. In addition, the objective of SimEDaPE is to work with large volumes of data generated by scenarios in large cities such as the city of Sao Paulo. The results presented for the scenario used are promising for the three cases. However, it is worth mentioning the processing time of the implementation using Cython with Joblib without the use of in-memory cache. This implementation has a fantastic time of $15s$ using 8 cores and with speedup of 156x compared to the reference sequential implementation. For future work, the objective is to test these implementations in scenarios such as the city of São Paulo with more than 4 million trips.

References

Berndt, D. J. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, page 359–370, Seattle, WA. AAAI Press.

Hamerly, G., Perelman, E., Lau, J., and Calder, B. (2005). Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism*, 7(4):1–28.

Paparrizos, J. and Gravano, L. (2015). K-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 1855–1870, New York, NY, USA. Association for Computing Machinery.

Rocha, F. W., Fukuda, J. C., Francesquini, E., and Cordeiro, D. (2021). Accelerating smart city simulations. *Latin America High Performance Computing Conference*. To publish.

Santana, E. F. Z., Lago, N., Kon, F., and Milojicic, D. S. (2017). InterSCSimulator: Large-scale traffic simulation in smart cities using erlang. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 211–227. Springer.