The Journal of Supercomputing (2024) 80:12408–12429
https://doi.org/10.1007/s11227-023-05881-1

Detailed parallel social modeling for the analysis of COVID-19 spread

Aymar Cublier Martínez[1] · Jesús Carretero[1] · David E. Singh[1]

Accepted: 23 December 2023 / Published online: 14 February 2024

Abstract
Agent-based epidemiological simulators have been proven to be one of the most successful tools for the analysis of COVID-19 propagation. The ability of these tools to reproduce the behavior and interactions of each single individual leads to accurate and detailed results, which can be used to model fine-grained health-related policies like selective vaccination campaigns or immunity waning. One characteristic of these tools is the large amount of input data and computational resources that they require. This relies on the development of parallel algorithms and methodologies for generating, accessing, and processing large volumes of data from multiple data sources. This work presents a parallel workflow for extending the social modeling of EpiGraph, an agent-based simulator. We have included two novel parallel social generation stages that generate a detailed and realistic social model and one new visualization stage. This work also presents a description of the algorithms used in each stage, different optimization techniques that permit to reduce the application convergence time, and a practical evaluation of large workloads on HPC systems. Results show that this contribution can be efficiently executed in parallel architectures and the results allow to increase the simulation detail level, representing a significant advance in the simulator scenario modeling. As a summary of results, the first contribution of this paper is the development of two models (a spatial and a social one) that assign geographical and socioeconomic indicators to each simulated individual (i.e., agents), reproducing the real social distribution of the city of Madrid. The second contribution presents an improved parallel and distributed algorithm that executes the two aforementioned models using different parallelization strategies and preserving the load balance.

Keywords Computational epidemiology · Workflows · Parallel processing · Visualization

Extended author information available on the last page of the article

1 Introduction

Since the beginning of the COVID-19 pandemic, computational epidemiology applications have been proved as efficient support-making tools for the health authorities. By means of these tools, it has been possible to anticipate the existing conditions of the pandemic (number of active cases, pressure on the health-care system and intensive care, number of expected deaths, etc.) as well as to evaluate in advance the impact of different health-related policies like vaccination strategies and non-pharmaceutical interventions.

In this context, in the last years, we have witnessed the appearance of many different approaches for modeling the COVID-19 propagation. Examples, among others, include methods based on differential equations [1], machine learning techniques [2], and statistical models [3]. One of the major challenges in the epidemiological modeling is to capture all the features related to the propagation in order to provide a more accurate prediction or more refined results. Examples of these features include breaking down the population by ages or collectives (instead of considering a homogeneous population), modeling different COVID-19 variants, considering different habits (for example, use of face mask) based on the individual age and occupation, or defining specific collective-targeted measures (for example, increasing the social distance for elderly people). Note that this increase in the refinement level can be crucial for modeling some scenarios, like for instance, simulating vaccination campaigns in which the population subjected to be vaccinated is prioritized according to the age, or enforcing the use of face mask in certain environments (hospitals, public transports, classrooms, etc.).

The epidemiological simulation based on agents [4, 5] considers the characteristics of each population individual, as well as the interactions with other individuals in the simulated environment. These models are able to incorporate many details to the simulation (both related to the individuals and the infections agents that are considered) which makes them very suitable for performing accurate forecast or modeling disease incidence on specific collectives (like elderly people, for instance). The two main drawbacks of agent-based simulators are that they are computationally intensive—which can be mitigated by means of parallelization employing parallelization techniques—and that they require of high-detailed input data. In this work, we present an extension of EpiGraph, a parallel agent-based simulator which is used to provide decision-making support to the Spanish and European Union Health Authorities [6, 7]. The novel contribution consists of a more detailed refined social model that can provide spatial coordinates and specific social-economic indicators (like educational level, economic incomes and political preferences) to each simulated individual. In addition, we introduce several optimization techniques oriented to reduce the application numerical convergence time while preserving the load balance between the processing components. We also provide a novel visualization interface integrated with Google Maps and a scalability analysis study.

These new features are integrated in the framework as the parallel workflow illustrated in Fig. 1. Note that this workflow only includes the novel components concerning the work presented in this paper. A more detailed description of the complete EpiGraph framework can be found in [8]. The EpiGraph execution workflow

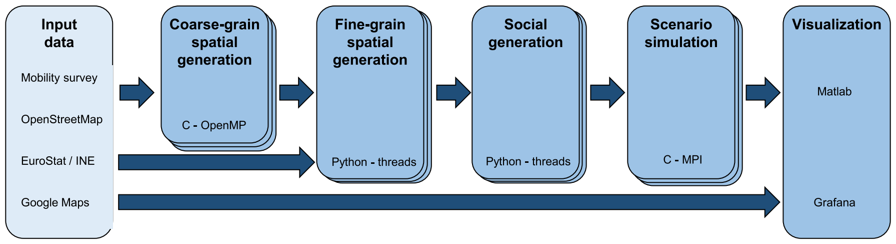


Fig. 1 Epidemiological simulation workflow consisting of five phases: coarse- and fine-grain spatial generation, social generation, scenario simulation and visualization and analysis. The first box on the left corresponds to the input data needed in the workflow. Each phase includes the programming language in which it is implemented

consists of several stages such as spatial generation, social generation, scenario simulation and data visualization and analysis. The scenario generation is responsible for assigning a specific spatial location to each simulated individual. It consists of two stages: the coarse-grain generation that is the costliest part of the stage and the fine-grained generation. In our experiments, we have used Madrid city as use case. Once this stage is completed, the social generation collects specific social indicators based on the individual location in order to create a realistic social model. Then, the scenario simulation is carried out using EpiGraph, an MPI-based parallel application. Finally, the simulation results (number of infections, hospitalizations, etc.) are displayed in the visualization stage. We plan to include real-time mobility information; thus, the whole workflow will have to be executed for every simulator execution in order to update the social model to the current social conditions.

The novel contributions of this work are the coarse- and fine-grain spatial generation algorithms, the social generation, and the visualization stages. The work structure is the following one: Sections 2.1 and 2.2 introduce the spatial model developed in EpiGraph. Section 2.2 provides a description of the social generation stage, and a visualization tool is presented in Sect. 2.4. In Sect. 3, we discuss the improvements made to the optimization algorithm that assigns spatial coordinates to individuals. Then, we show in Sect. 4 the results obtained in this work. Finally, Sects. 6 and 7 present the related work and main conclusions of this paper, respectively.

2 EpiGraph workflow

The goal of this work is to improve and refine the social and spatial models in EpiGraph. As a starting point, in 2018, the Madrid region conducted a mobility survey (Madrid Mobility Survey or MMS from here forward) [9] to study how the population aged over three traveled across the territory. This survey was done using two gathering methods: CAPI (face-to-face interviews with every family member) and CATI (phone interviews with one family member). The sample size was 86,064, with 34,652 CAPI and 50,412 CATI. For setting the sample, CAPI method used the region census of September 2017, and CATI used a database of phone numbers linked with addresses. Among other variables, the questionnaire collects, for each interview, information about the transport area (with an approximate size of 5 blocks) of departure location and arrival destination of each journey, the reason of the journey, and the means of transportation.

Based on this survey, the spatial or mobility model consists of two steps: the coarse-grain and the fine-grain refinements. In the coarse-grained refinement, we focus on big areas of the city—the 21 districts of the city. We calculate the commutes from and to the districts based on the information of the MMS. This allows us to introduce two new variables for each individual: *residence district* and *work district*. We aim to calibrate these two variables using the MMS as reference, i.e., the goal is that the individuals in the simulation produce a transportation pattern (journeys from the residence district to the working district and *vice versa*) as close as possible to the one obtained from the MMS. In the fine-grained refinement, we distribute the individuals within their related districts.

2.1 Coarse-grain spatial generation

In this section, we describe the optimization algorithm that determines the residence and work districts for each individual. This algorithm is written in C and parallelized with OpenMP. We first define some of the procedures and variables that will be used to describe the algorithms.

One key variable for the optimization algorithm is the transport matrix T. In the considered scenario, the matrix has dimension 21×21 (for the 21 districts in Madrid). We define matrix T_r as the matrix extracted from the 2018 MMS. Each entry (i, j) represents the number of travels between residence district i and work district j. We normalize this matrix to ensure that all the columns add up to 1. Furthermore, we define matrix T_s as the matrix obtained at each step of the algorithm by computing the movement from residence and work data for each simulated individual. For this, we count each journey from residence district i to work district j and add it to matrix T_s. We later normalize the columns in order to provide a straight comparison with matrix T_r.[1] The main goal of the optimization algorithm is to minimize the cost function depicted in Eq. 1, where $||\cdot||_F$ is the Euclidean norm for matrices (defined as the square root of the sum of the absolute squares of its elements). The coarse-grain spatial generation consists of an initialization and an optimization phase.

$$C(T_s) = ||T_s - T_r||_F \tag{1}$$

Initialization phase the first stage of this algorithm is the initialization, in which the individuals receive a tentative residence and work district. This stage is executed sequentially due to its reduced execution time and can be decomposed into the three main steps depicted in Algorithm 1. First, for each individual, we calculate the family relationships (lines 1 to 4). Note that each individual can reside alone (no family members) or reside together with other family relatives (up to four members).[2] In a second step, each individual is assigned to a work district (line 5). Finally, each individual, and all the family members, are assigned to a residence district (lines 6 to 8).

[1] Note that the simulated population is smaller than the real one considered in the survey.

[2] The family member distribution as well as other characteristics of the simulated individuals (like age pyramid or occupation) are extracted from EuroStat and other national Statistical databases.

Algorithm 1 Initialization of residence and work districts vectors

```
1: for $i \in [1, n]$ do
2:     $ReadRelationMatrix$
3:     $WriteFamilyMatrix$
4: end for
5: $\widetilde{w} \leftarrow$ randsample$(m, 1, 21, weight)$        /* Assign work districts */
6: for $i \in [1, n]$ do
7:     $r \leftarrow AssignResidenceDistrict$
8: end for
```

Algorithm 2 Coarse-grain spatial generation algorithm

```
1: $H = \bigsqcup_{l=1}^{M} H_l$
2: for $iteration = 1 \rightarrow N$ do
3:     $\sigma \leftarrow$ $n$-$cycle$                          /* Choose random cycle */
4:     #pragma omp parallel num_threads(MAX_THREADS)
5:     parallel do
6:         $r_l \leftarrow r,\ \ w_l \leftarrow w$                 /* Local copy of residence and work */
7:         $D_l \leftarrow (T_s - T_r)_{H_l}$
8:         $(i, j) \leftarrow ComputeMinMaxCoor(D_l)$
9:         $d \leftarrow CostFunction$
10:        #pragma omp parallel for num_threads(NESTED_THREADS)
11:        for $k \in \sigma(1, ..., n)$ do
12:            if $(r_{l,k}, w_{l,k}) = (i, j) \wedge (D_l[i][j] > 0)$ then
13:                for $district \in H$ do
14:                    $r_{l,k} \leftarrow district,\ \ r_{l,F(k)} \leftarrow district$
15:                    $d' \leftarrow CostFunction$
16:                    if $d' < d$ then
17:                        $d \leftarrow d',\ \ Apply\ changes$
18:                    else
19:                        $Undo\ changes$
20:                    end if
21:                end for
22:            else if $(r_{l,k} \neq i) \wedge (w_{l,k} = j) \wedge (D_l[i][j] < 0)$ then
23:                $r_{l,k} \leftarrow i,\ \ r_{l,F(k)} \leftarrow i$
24:                $d' \leftarrow CostFunction$
25:                if $d' < d$ then
26:                    $d \leftarrow d',\ \ Apply\ changes$
27:                else
28:                    $Undo\ changes$
29:                end if
30:            end if
31:        end for
       end parallel
32:    $SortThreadsCost$                        /* Merge results of $d$ best threads */
33:    #pragma omp parallel for
34:    for $k \in \sigma(1, ..., M)$ do
35:        $Choose\ Best\ k^{th}\ Thread$              /* Select up to $d$ (depth) threads to commit */
36:    end for
37:    $DynamicBalance(\tau, D)$                /* Depicted in Algorithm 4 */
38: end for
```

For each simulated city, we have a relation sparse matrix M and a work vector $\widetilde{w}$, respectively of dimensions $n \times n$ and size m, where $n > 0$ is the population in the city considered and $m > 0$ the number of work groups (in our considered use case $n = 3,511,110$ and $m = 772,337$). Each value $m_{i,j}$ in matrix M determines the type of relation (or contact) between individual i and j. Similarly, each scalar $\widetilde{w}_k$ in vector $\widetilde{w}$ indicates the size of work group k. Note that M contains individual contacts related to family, work and leisure activities. In this work, we are only considering journeys from the residence to the work place. Consequently, we need to extract only the family contacts, so when one individual is placed in a certain residence district, all the related family members are also placed in the same district. In order to track the family connections of each individual, we construct an auxiliary matrix F of dimensions $n \times 5$ (5 is the maximum number of family relations per individual in the simulations), where vector F_i contains the individuals having a family relation with individual i.

In a second step, the work vector $\widetilde{w}$ is used to assign a work district randomly to each work group. Then, we generate a work district vector w of size n, the population. Each scalar w_i indicates the working district of individual i. Finally, in the last step, we assign to each individual and all his family members a certain residence district.

Let us assume we have assigned individual i to work district w_i. Let r be the vector containing the residence districts. The assignation to a residence district is done by choosing a random district, but weighting the sample by column C_{w_i} of matrix T_r. In this way, we assure that the distribution of residence districts is as close as possible with the one obtained from the MMS. At the end of the initialization stage, two vectors of size n are generated: w, which lists the working district for each individual and r, which lists the residence district for each individual.

Optimization stage The second stage corresponds to the optimization logic, depicted in Algorithm 2 that includes the OpenMP pragmas. The algorithm decomposes the existing districts between the running threads using nested parallelism. The demographic data (404,173 inhabitants) read from disk was parallelized following a scheme in which each thread access to independent portions of data. It is worth mentioning that the I/O represents a small fraction—smaller than 1%—of the overall execution time. Let H be the set of city districts that is being modeled (e.g., $H = [1, 21]$ for Madrid) and $M > 0$ the number of threads. Then,

$$H = \bigsqcup_{l=1}^{M} H_l, \tag{2}$$

where $(H_l)_{1 \le j \le M}$ is a disjoint partition of H. In the first level (outer parallel section, lines 5 to 31 in Algorithm 2), each thread l works on its own subset H_l of columns of matrix $D = T_s - T_r$ that correspond to different districts on the city. To assure two or more threads do not write to the same arrays r and w, copies r_l and w_l are created for each thread. Then, every thread executes the sequential optimization algorithm on its sub-matrix $D_l = (D)_{H_l}$ and annotates every individual it has optimized (i.e., moved from one district to another in order to minimize the cost function depicted in Eq. 1).[3]

The merging section at the end of every outer iteration collects the results of the threads involved in the execution (see lines 32 to 36). We introduce a parameter for determining the depth of the merging operation. First, we sort the threads according to the improvement achieved in the cost $d_l = ||T_{l,s} - T_r||$. Second, we select as many threads (ranked by smallest cost) as the value of the depth parameter d. Then, we commit the changes introduced by individual assignations done by these threads, prioritizing the ones with minimum cost. Finally, all threads are updated with the same optimized array r before the start of the next iteration. With this procedure, only the changes of the best d threads are committed. The next and final stage of the iteration is to execute the dynamic balance of the threads (see line 37). We will discuss further this procedure in Algorithm 4.

Note that the degree of parallelism of the first level is limited by the number of matrix columns (21 in our scenario). In the second parallel level—that corresponds to the nested parallel section—we parallelize the for loop that iterates over the individuals (see line 11 in Algorithm 2). Then, we use private copies of residence district vector $r_{l,\tau}$ for each nested thread τ for each thread l. Each nested thread iterates over its assigned group of individuals and commits the changes in the global data structure at the end of the loop. In line 12, we check if the individual resides and works in the local extrema coordinates calculated in line 8 and check if these extrema are a maximum. In this case, we move the individual from its residence district to a new one. In the other case (line 22), when the extrema are a minimum and the individual does not reside in the minimum district i, we move the individual to the district that produces the minimum value. Note that given that the number of individuals is large, it is possible to involve many threads in this nested level.

Figure 2a shows an example of the original transport matrix T_r, representing the target distribution to be reached. In Fig. 2b, we show T_s, the matrix produced in the initialization phase. Finally, Fig. 2c shows the optimized matrix T_o, produced by the optimization algorithm. Note that T_r and T_o are similar, the former is obtained from the MMS and the latter from the individual distribution defined in the simulator.

As a summary of this algorithm, firstly each thread T_i is assigned to a partition of the set of districts (line 5, Algorithm 2). Then, each thread creates nested threads to iterate over the individuals (lines 11 to 31). Finally, at the end of every outer iteration, every copy r_i is committed with a depth parameter d to generate r_{opti} (lines 32 to 36). There are three conditions for terminating the optimization process—by exiting the outer parallel loop in line 5:

[3] The 21 districts of Madrid are, following the numerical order in the figure: Centro, Arganzuela, Retiro, Salamanca, Chamartín, Tetuán, Chamberí, Fuencarral-El Pardo, Moncloa-Aravaca, Latina, Carabanchel, Usera, Puente de Vallecas, Moratalaz, Ciudad Lineal, Hortaleza, Villaverde, Villa de Vallecas, Vicálvaro, San Blas-Canillejas, Barajas.

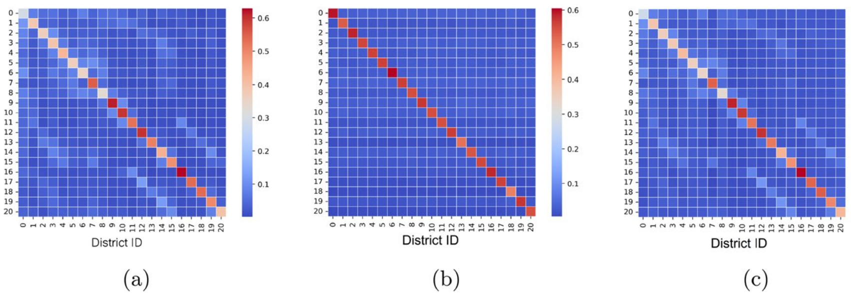


Fig. 2 Comparison between matrices T_r (**a**), T_s (**b**)—not yet optimized with a cost equal to 0.800—, and T_o (**c**)—optimized with a cost equal to 0.201—[2]. The color of each coordinate of the matrix represents the normalized value of each element of each matrix (number of travels from district i to j, normalized by matrix columns)

(1) The number of successful optimizations (changes applied in residence district that reduce the cost) reaches a predefined maximum value of 1000.
(2) The number of unsuccessful optimizations since the last successful one reaches a predefined maximum value of 1000.
(3) The optimization suitability has been evaluated for all individuals in the district.

At the beginning of the execution, there are so many optimization opportunities that the most frequent exit condition is the first one. However, as the algorithm progresses, conditions 2 and 3 become more frequent, as there are less possible optimizations that it is possible to perform. Figure 3 shows a graphical example of the parallel algorithm workflow with $k = 4$ (first level threads), $l = 2$ (second level threads) and $d = 2$ (depth value of 2). In this example, thread T_1 has minimum cost $C(T_{l,s})$ followed by T_4. Therefore, r_{opti} commits changes made by thread T_1 and T_4.

To end this subsection, we prove a result of time complexity of Algorithm 2 that will be useful in the following pages.

Lemma 1 *The time complexity of Algorithm* 2 *is* $\mathcal{O}(n^2)$.

Proof First, notice the parallel for loop iterating over the individuals in line 11. This section of the algorithm is in $\mathcal{O}(n)$. If we show that the number of iterations N depends linearly on the size of the simulation, we will have proved the complexity $\mathcal{O}(n^2)$, as there are two for loops, one inside (line 11) the other (line 2). However, this is also true, because the number N depends on the average cost gain $|d - d'|$. The smaller the gain, the more iterations will be necessary to reach a given threshold cost. Finally, the average cost gain depends itself linearly on the size of the problem, by definition of the cost function in Eq.(1). $\square$

2.2 Fine-grain refinement

The previous stage maps each simulated individual to a certain residence and work district. In the fine-grain refinement, we increase the detail level by adding two more

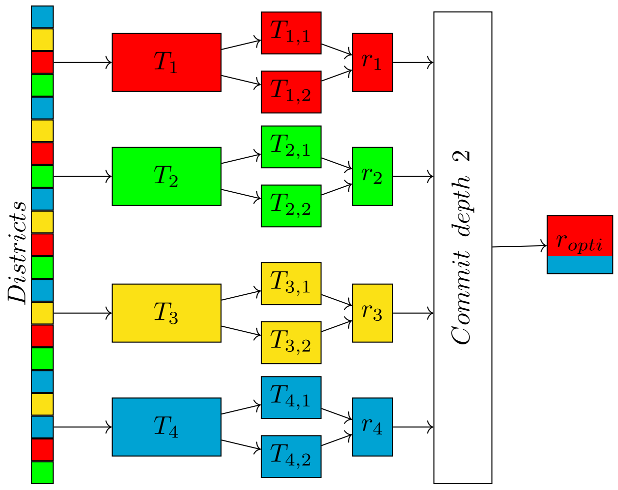


Fig. 3 Parallel algorithm workflow example. We use $k = 4$ main threads, each having $l = 2$ nested threads and a depth value of 2 ($d = 2$). In this example, main threads T_1 and T_4 reach to the first and second best solutions, respectively

variables to each individual: *residence address* and *work address*. Each variable corresponds to a specific address (street-number) in the city. In this stage, this mapping is refined at street-number level using data from the OpenStreetMap project [10]. A JSON file containing information on residence buildings such as street, number, zip code and coordinates is generated from OpenStreetMap.

Algorithm 3 Fine-grain spatial generation algorithm

```
1: for $i \in [1, ..., n]$ do
2:     Collect residence district
3:     Map to a census section
4:     Choose street within the census section
5:     Assign coordinates
6: end for
```

Algorithm 3 shows the pseudocode of the fine-grain spatial generation. For each individual, the related residence district is obtained (line 2) using the results of the previous stage. Then, based on real data of Madrid city [11], we obtain the population distribution vector across census sections[4] of the considered district. In line 3, we choose randomly a census section for the individual by weighting the probabilities by this distribution vector. In this way, the probability of mapping one individual to a certain census section will depend on the section's population (i.e., the census section weight). As input data source, we use [12], which contains information about addresses distributions across census sections. With this information, it is possible to choose a random street of the census section (line 4) for performing the

[4] Census sections are the lowest level units for the collection of statistical information. In a city, a census section has a size of a few housing blocks.

mapping. Finally, the set of coordinates related to the address calculated before are assigned to each individual, using the JSON file generated from OpenStreetMap. This algorithm is parallel implemented in Python scripts.

2.3 Social generation

In this stage, the social model used by EpiGraph is refined using the data from the previous spatial generation stage. We carry out a breakdown of socioeconomic variables by districts for the city of Madrid. Then, using the spatial model to approximate the number of residents in each district, we can refine these variables to a fine-grained geographical level using demographic and socioeconomic data from official sources.

We first focus on income *per capita* by district. Open data on this variable is available on the webpage of the Madrid city council [13–15]. Comparing these data with spatial data generated by the model, we can study correlation between COVID-19 incidence and income per capita. Other interesting indicators to consider are educational level and population by nationality.

The data acquired consists of three socioeconomic variables: income *per capita* (average income earned per person, year 2018), Gini index (measures the income inequality, year 2020) and population density (population per km^2, year 2022), with a spatial resolution of census section. This stage is done by scripts in R that are responsible for loading and parsing the input data and Python that assign the socioeconomic indicators to each individual. Section 2.2 introduces the methodology that assigns to each simulated individual a residence census section, which is performed according to the individual's residence district and the sizes—in terms of population—of the census sections belonging to the district. Then, as both the socioeconomic variables and the individuals are classified by census section, it is possible to map the indicators to the individuals. All this process is embarrassingly parallel (one thread per indicator) and is completed within seconds.

Finally, three different infection risk scale factors of `COVID-19` infection are assigned to each simulated individual. Each risk is relative to a socioeconomic indicator. These values of risk come from the scientific literature [16–18] and establish some degree of correlation between `COVID-19` incidence and socioeconomic variables. Lastly, the global infection risk r_i is computed based on the partial risks following the relation $r_i = r_{i,1} \times r_{i,2} \times r_{i,3}$ and is introduced in the simulator EpiGraph.

2.4 Simulation and visualization

The epidemiological model of EpiGraph has been extended considering the work presented in [19]. This work analyzes the relationship between social disparities and the COVID-19 propagation in Germany, considering different regional indicators like health, socioeconomic status or age structure of the population. Based on this work, the previous social indicators have been used to determine the COVID-19 infection degree. In this way, instead of considering a homogeneous population, now the disease is propagated on a non-homogeneous society, in which the individual characteristics are different depending on the area they reside.

Grafana [20] has been used to interactively visualize the results generated in the simulation stage. The visualization stage displays the spatial location of each individual on a city map using the Google Map API. In this way, it is possible to display, in an interactive fashion, the spatial location of each individual and track how the COVID-19 disease is propagated through the city. Figure 4 shows an example of individual location of the city of Madrid using two different scales. We can observe the spatial location of the individuals, next to each building entrances. It is important to highlight two points. First, the spatial generation only considers residence areas, thus an individual will not be mapped into public areas (parks) or commercial buildings (malls, businesses centers, etc.). Indeed, the motivation of this choice is to later include socio-economic variables for each individual. By mapping them to residence addresses, we will be able to access to more refined statistical data and assign them to each individual. Furthermore, this assumption does not hinder the spreading model, as public areas and commercial buildings are already implemented in the epidemiological model of EpiGraph. Second, the social model considers both the residence and work addresses, so the mobility patterns of the simulated individuals match the city's actual one.

3 Workflow optimization

This section introduces two methodologies aimed to improve the workflow execution performance. The first one is a novel technique that increases the coarse-grain spatial generation convergence rate while maintaining the load balance among the running threads. The second one provides a performance comparison of different policies for processing the complete EpiGraph dataset on a parallel architecture.

3.1 Algorithm convergence acceleration

The coarse-grain spatial generation and the simulation stages are the most time-consuming components in the EpiGraph workflow. We have considered three different distributions of the districts among the threads, cyclic, block and balanced, as shown in Fig. 5 for a set of 16 districts and four main threads. Note that the different size of the districts may lead to load unbalance situations for the block and cyclic distributions. The balanced distribution—Fig. 5c—is similar to the cyclic partitioning, but the districts are firstly sorted by population. In this way, we ensure an even distribution of the districts among the threads according to the number of individuals that each thread has assigned. Note that all these distributions (block, cyclic and unbalanced) are static, i.e., they are not modified during the algorithm execution.

We have evaluated the code performance for these three distributions using Intel® VTune^{TM} Profiler [21] on the Tirant v3 supercomputer in which each node is composed by two sockets of Intel Xeon Sandy Bridge E5-2670 with 8 cores each at 2.6GHz, for a total of 16 cores per node and 32 GB of DDR3 main memory. The performance results show that the block and cyclic distributions (introduced in Sect. 2) exhibit the largest unbalance, which is reduced by means of the balanced distribution. However, given this distribution is also static, when the coarse-grain spatial generation algorithm progresses, the lack of optimization opportunities produces unbalance situations among the main threads.

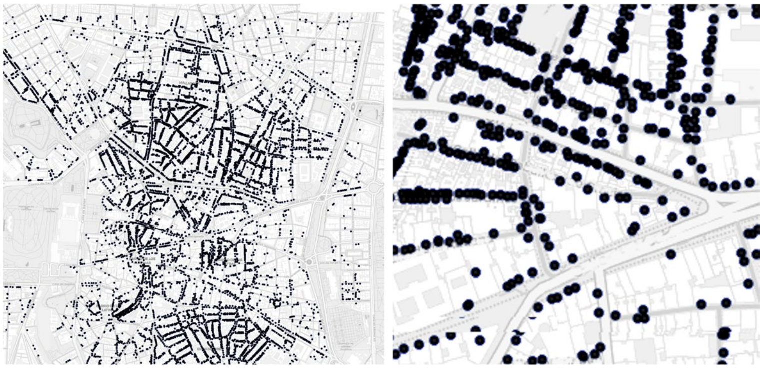


Fig. 4 Central district of Madrid, where each point represents a simulated individual that has been assigned its residential address. Map centered at 40° 27’ 57.7758" N, 31 41′ 21.3828" W

To tackle this problem, we have developed a dynamic algorithm that is able to increase the workload of the threads with the smallest execution time when a certain unbalance condition is reached. We first define the threshold r for measuring the amount of unbalance between the running threads. We denote $t_{\max}$ the time spent by the slower thread and $t_{\min}$ the time spent by the fastest thread during any given iteration, the threshold is defined by the ratio $r := (t_{\max} - t_{\min})/t_{\max}$, so that $0 \leq r \leq 1$. Therefore, if $r = 0$ that means that all threads spent the exact same time inside the given iteration. The load balance algorithm, depicted in Algorithm 4, collects the thread execution time in every iteration (line 1) and stores the value in T (note that the first entry of this vector is T(1)). Then, it calculates the threshold value r using the values of the threads with largest and smallest execution times (line 5). If this ratio is greater than some given threshold $\tau \geq 0$, a certain number of districts (given by the swap depth parameter D) is exchanged between the threads with the smallest execution times (line 8). Figure 5d shows an example of this operation assuming that threads 3 and 4 have the smallest execution time. Note that the districts that are swapped are the smallest ones.

Algorithm 4 Dynamic balance algorithm

```
    Input: $\tau, D$        /* Unbalance threshold and swap depth */
 1: $T = ThreadMonitor()$                    /* Array of thread execution times */
 2: $Sort(T)$
 3: $t_{\max} = T(1)$
 4: $t_{\min} = T(N_{thread})$
 5: $r = (t_{\max} - t_{\min})/t_{\max}$
 6: if $r > \tau$ then
 7:     for $k \in (1, ..., D)$ do
 8:         $ExchangeDistrict(N_{thread}, N_{thread} - 1)$
 9:     end for
10: end if
```

The results of this optimization are twofold. First, the workload is better balanced between the main threads as well as the nested threads. Note that each thread only

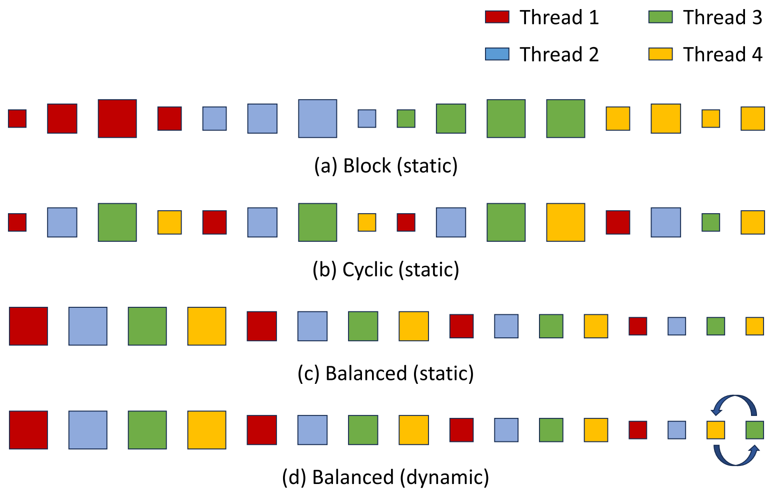


Fig. 5 Different distribution schemes for a set of 16 districts and four main threads: **a** static block distribution, **b** static cyclic distribution, **c** static balanced, and **d** dynamic balanced with one district swap

performs optimizations (i.e., exchange the individual's residence location in order to minimize the cost function) between the districts that it has assigned. Consequently, a district swap between two threads increases the optimization opportunities—and consequently the thread execution time—given that there are new exchange combinations available between the districts assigned to the threads. This strategy has an interesting side implication: the algorithm convergence time is also reduced, given that it can perform a larger number of optimizations. As we show in Sect. 3.2, this permits to speedup the algorithm convergence time.

3.2 Large workflow executions

In this section, we analyze the workflow execution of the preprocessing stages on an HPC platform. In total, EpiGraph dataset includes the 642 largest European cities, with a total population of 189,433,972 individuals. Note that the spatial generation stage works independently for each urban city. Our goal is to estimate the processing time of the complete dataset while running the workflow on multiple compute nodes. We have developed a simulation framework that receives, as input data, the real processing time of three cities with different sizes and the load balance policy that is being used. In our experiments we have used the actual processing time for a small (128,260 inhabitants), medium (404,173 inhabitants) and large (3,511,110 inhabitants) city. This configuration was evaluated with three distribution schemes: static balanced, dynamic balanced and dynamic random. The dynamic random approach is similar to the dynamic balanced one, but it swaps the districts between two randomly selected districts (instead of the ones with the smallest execution time).

Given that the algorithm time complexity is $\mathcal{O}(n^2)$ by Lemma 1, where n is the city population, we use a polynomial fitting of degree 2 of the form $ax^2 + bx + c$, with $a > 0$ for interpolating the estimated execution times of the rest of the cities in the city's dataset. Note that we obtain three different polynomials (one for each distribution scheme), each one with a correlation coefficient R equal to 1, which validates our assumption of algorithm complexity. In this way, it is possible to estimate the execution time of a certain city, knowing its population size and the distribution scheme that is being used. By means of simulation, we have computed the required time needed to run all 642 simulations on parallel architecture considering a different number of compute nodes and workload distributions. The next section describes the results that we have obtained.

4 Evaluation

In this section, we present a practical evaluation obtained on an Intel Xeon Gold 6212U processor with 48 cores. We have used the gcc 7.5.0 compiler version in a platform with Ubuntu 18.04. All tests were conducted on three different population scenario such ass small, medium and large. In the small scenario, we simulate 128,260 individuals. In the medium one, we simulate 404,173 individuals. Finally, in the large scenario, we simulate a population of 3,511,110 individuals. The performance evaluation is focused on the coarse-grain spatial generation algorithm that is the most time-consuming part of the workflow (besides the simulation stage that is not considered in this work). For instance, an execution of the coarse-grain spatial generation algorithm lasts thousands of seconds, while the rest of the stages (fine-grain spatial generation, social generation, and visualization) last a few seconds.

Let us introduce a notation for describing the different parameters used in the experiments. Notation $k - d - l$ indicates that the test is done with k main threads, with depth parameter d and with l nested threads. (We omit the l if there is no nested parallelism.) Given that performance results for cyclic and balanced modes are similar as we see in Fig. 6, we chose for simplicity the cyclic partitioning in the rest of the tests that are presented.

In a second set of experiments, we study which of the partition configurations gives better results. We valuate this by measuring the time needed to reach a predefined threshold cost. We evaluate different numbers of nested threads for each configuration.

In the next step, we analyze the numerical convergence of different configurations. For this test, we choose to fix each one of the dimensions: k, the number of main threads, and l, the number of nested threads. We measure the evolution of the cost function C in time. These results were obtained using the medium-sized simulated population (404,173 individuals). Figure 7a evaluates different scenarios, fixing the dimension of nested threads and setting $l = 1$. We observe a big improvement in convergence from $k = 1$ main threads to $k = 4$, and between $k = 4$ and $k = 8$. However, after increasing the number of main threads greater than 8, we observe that the improvement is negligible. Figure 7b ranges the number of nested threads

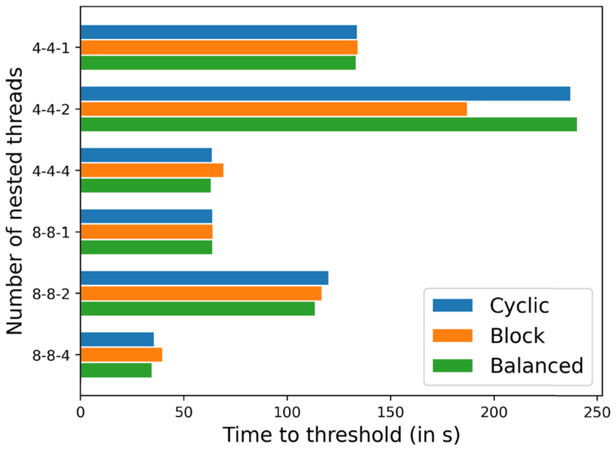


Fig. 6 Time to reach threshold cost of $t = 0.20$ between different configurations of nested threads, for $k = 4, d = 4$ and $k = 8, d = 8$

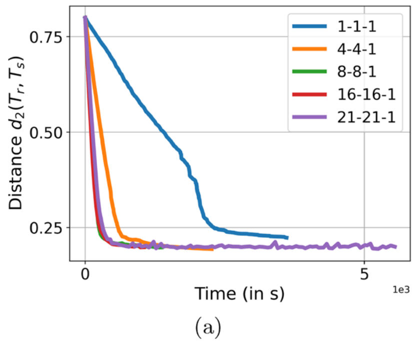
when the number of main threads is fixed ($k = 2$). We observe that it is possible to scale in this dimension, reducing the convergence time when the number of nested threads is increased.

Figure 8 evaluates the impact of the depth parameter on execution time. Fixing the number of main threads and nested threads, we analyze how the time to reach a threshold cost change when modifying the value of parameter d. In this figure, we set $k = 8$ and $l = 1$. We set $d = 1, 2, 4, 6, 8$. As expected, we observe that the best execution times are achieved with maximum depth parameter ($d = k$). Note that between $d = 1$ and $d = 8$, there is an improvement factor 6. Even by increasing the depth from 1 to 2, we reduce the execution time by half.

Figure 9 shows the time to reach a threshold cost for five different configurations, each one chose to utilize all the cores available (48 in total), except 21-21-2 configuration because there cannot be more main threads than the 21 districts.

According to the previous insight, we have set the depth parameter to the maximum value because it is the one that achieved the best performance. We have grouped the results by size of the scenario. The simulation having the maximum number of main threads, i.e., 21-21-2, seem to perform less efficiently than those with a balanced number of threads (4-4-12, 8-8-6 and 12-12-4). Furthermore, the configuration 2-2-24 seems to provide the best results.

Regarding the improvement of the coarse-grain spatial generation stage introduced in Sect. 3.1, we have compared for the $8 - 8 - 2$ configuration three different district distributions, namely static balanced, dynamic balanced, and dynamic random in which two random threads are chosen for swapping D of their assigned districts. This evaluation was carried on Tirant v3 supercomputer, in which each node has two Intel Xeon Sandy Bridge E5-2670 processors with 16 cores per node. Figure 10a to 10c compares these results for three problem sizes. In the experiments, we vary the dynamic balanced mode between **static balance**, **dynamic balance**, and **dynamic random**. The dynamic balance is the faster one, followed by the dynamic



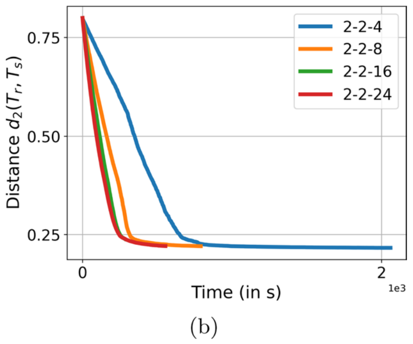


Fig. 7 Cost evolution in time. In **a** fixing the number of nested threads to 1, configurations 1-1-1, 4-4-1, 8-8-1, 16-16-1 and 21-21-1. In **b** fixing the number of main threads to 2, configurations 2-2-4, 2-28, 2-2-16 and 2-2-24

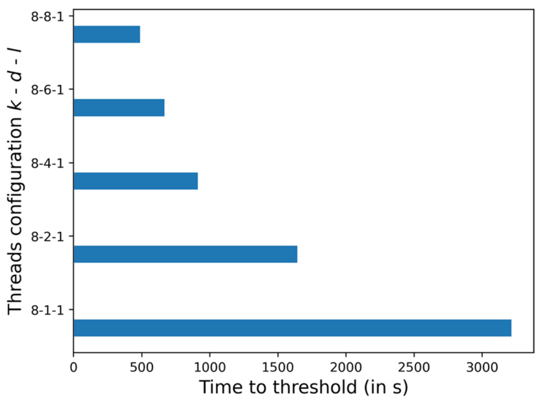


Fig. 8 Time to reach threshold cost $t = 0.25$ for different depth configurations: 8-1-1, 8-2-1, 8-4-1, 8-6-1 and 8-8-1

random. No balance at all (i.e., static balance) gives the poorer results. Different sized simulations, with (a) small 128,260 inhabitants, (b) medium 404,173 inhabitants, and (c) large 3,511,110 inhabitants. In (d), convergence time to threshold cost $t = 0.20$ for the total 642 European cities that are sorted in an ascending, descending or random order for a dynamic balanced data distribution.

Note that the dynamic approaches improve the convergence of the algorithm (cost vs time) better than the static counterpart. As expected, the dynamic balanced distribution performs better than the random one, especially for small problem sizes. The dynamic balanced distribution performs 16.8% more optimizations than the static version and 23.1% more optimizations than the dynamic random. Moreover, for the large size problems, the algorithm convergence is much better for the dynamic versions.

We have simulated the execution cost of large workflows consisting of 642 cities on Tirant v3 supercomputer (see Sect. 3.2 for details). Figure 10d shows the results considering a dynamic balanced data distribution, different number of compute nodes and three workflow execution orders such as ascending,

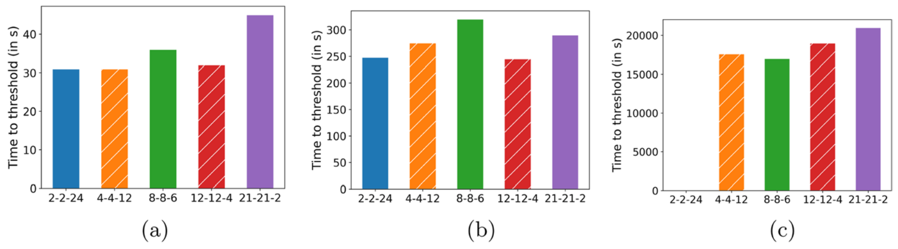


Fig. 9 Time to reach threshold cost $t = 0.25$ in configurations 2-2-24, 4-4-12, 8-8-6, 12-12-4 and 21-21-2. From left to right, with **a** small 128,260 inhabitants, **b** medium 404,173 inhabitants, and **c** large 3,511,110 inhabitants

descending and random, where the cities are sorted in ascending, descending and random order, respectively. Note that when this ordering is set, the different cities are processed by the available compute nodes following a FIFO scheme.

In Fig. 10d, we observe that the descending order produces the smallest processing times, and the differences with the other orders are more important when a reduced number of compute nodes is used. The reason is that with the descending order, the largest cities are processed first. According to the algorithm time complexity—which is quadratic-related to the population size—these cities represent the most time-consuming part of the workflow. With the descending ordering, the workload can be balanced better using the medium and small cities. Furthermore, when the number of nodes increases, each node will have assigned fewer cities, and the algorithm convergence time will be limited by the time required to compute the largest cities.

5 Discussion

In this section, we discuss the main results of this paper, including the strengths and limitations of the work. Finally, we discuss how to overcome these limitations.

The development of a spatial and a social model for EpiGraph represents a major improvement in the quality and precision of the epidemiological simulations. A coarse-grained spatial approach (district detail level) using real transportation data (see Sect. 2.1) allows us to construct a more refined model using a parallel algorithm that assigns random sampled addresses to a large number of agents (see Sect. 2.2). Furthermore, a social model is constructed using the geographical information available from the spatial model (see Sect. 2.3), which are then represented on a map with spatial visualizations tool, such as Grafana (see Sect. 2.4). Second, the refinement of the parallel algorithm is the second main contribution of this paper. By studying in detail the base parallel algorithm, we are able to develop a more optimized version of the algorithm that reduces the execution time of the original proposal, as seen in Sect. 3 and Sect. 4. A strength related to this optimization process is the possibility to extend this work

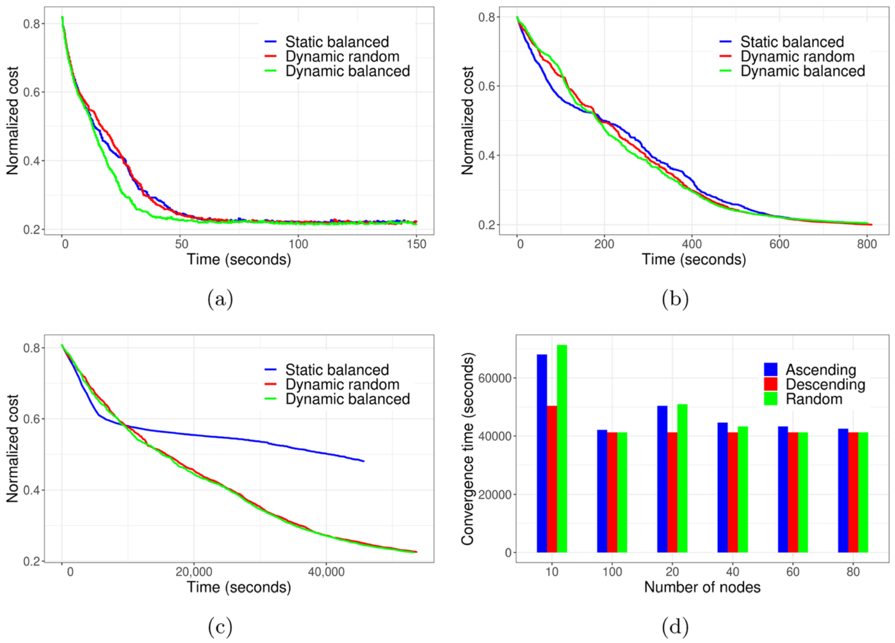


Fig. 10 Cost evolution in time, with fixed configuration 8-8-2

to distributed memory system, as the developed algorithms are well-suited for both shared and distributed memory platforms.

Regarding the work limitations, the social model can be further improved by combining the information of all the three models (epidemiological, spatial and social) considered in the simulation. This will allow to enrich the simulation results with cross-model indicators. For instance, having access to epidemiological data at district level will allow to determine if local district-level lockdowns (like the ones that occur in Spain in 2020) are applied. Using this information, it would be possible to refine the social model, given that some of the individuals residing in the affected districts will not be allowed to go to work. Another limitation is the integration with other data sources more recent than the mobility survey. One example is to use traffic conditions (measured in real time) for modeling the individual mobility. Furthermore, as discussed in Sect. 3.2, large-scale workflow executions have not yet been evaluated on distributed compute nodes. With some minor modifications, the algorithms presented in this paper can be adapted to these platforms. Regarding the implementation of EpiGraph, the simulator has not been currently integrated with OpenMP. We also think that a hybrid parallel model (based on the combination of MPI and OpenMP) would increase the simulator performance. Finally, we plan to provide EpiGraph with malleable capabilities that permit to dynamically increase or decrease the number of processes according to the computational intensity of the program phase that is being executed.

6 Related work

Agent-based model EpiGraph [22] is a fully distributed simulator for influenza and COVID-19 diseases. One of the distinguishing features of EpiGraph is that it relies on realistic data for both individuals and their interaction patterns, which are extracted by scaling from existing social networks and contact matrices.

This section discusses other works on epidemic models, similar or related to EpiGraph. Reiner et al. [23] use a deterministic SEIR framework to model the propagation of the virus and the effect of NPIs (social distancing mandates and mask use) until Spring of 2021. Some of their limitations are the absence of age structure and the assumption of a well-mixed population. Kerr et al. [24] include demographic information about age structure and population size. Different from our work, contacts are not based on existing patterns; scalability issues are partly sidestepped by dynamic scaling. Vaccines are modeled by adjusting individuals' susceptibility to infection and probability of developing symptoms after being infected; both modifications affect the overall probability of progressing to severe disease and death. However, some features we consider in EpiGraph (such as vaccine effectiveness across variants) are not currently implemented in Covasim. Modeling social mixing is crucial for obtaining realistic simulations. Other research [25–27] considers different ways to refine social interactions. In EpiGraph, social mixing is modeled using the Facebook and Enron contact networks and individual contact matrices.

Bubar et al. [28] compare five age-stratified prioritization strategies in terms of cumulative incidence, mortality, and years of life lost. Some limitations have to do with using pre-pandemic contact matrices, not incorporating NPIs, and only considering variation in disease severity and risk by age-although contact rates, and thus infection potential, vary greatly not only by occupation and age. Results show, such as in our work, that people aged 60 years and older should be prioritized to minimize deaths. Matrajt et al. [29] use a mathematical model paired with optimization algorithms to determine the optimal use of vaccine for different combinations of vaccine effectiveness and number of doses available under a wide variety of scenarios; the optimal allocation strategies were computed using age as the sole risk factor. This work obtains similar conclusions as our work, that is, for low vaccine effectiveness, the best option for reducing deaths is to allocate vaccines to older age-groups first.

Models such as SIMID-SCM [30] consider the evolution of the COVID-19 epidemic in Belgium with a deterministic age-structured extended compartmental model. This model integrates social contact data and is fitted on hospitalizations' data (admission and discharge), on the daily number of COVID-19 deaths (with a distinction between general population and nursing home related deaths) and results from serological studies, with a sensitivity analysis based on a Bayesian approach.

Other agent-based model, closer to EpiGraph, such as SwissTPH-OpenCOVID [4] is a stochastic, discrete-time, individual-based transmission model of SARS-CoV-2 infection and COVID-19 disease. The model simulates viral transmission between infectious and susceptible individuals that come in contact through an age-structured, small-world network.

MOCOS international research group has developed in [5] an agent-based model on the basis of a continuous time stochastic micro-simulation. All relevant duration times like incubation time, time till hospitalization, and time till testing are sampled from distributions based on empirical data.

7 Conclusions

In this work, we present a novel parallel execution workflow for extending the social modeling and visualization capabilities of EpiGraph simulator. This includes novel spatial generation algorithms that determine the residence and work location of each simulated individual with a high-detail level. This information is used by a social generation stage to assign to each individual different social-economic indicators related to the place of residence. These indicators are subsequently used to determine a major or minor risk of transmission of the disease. Note that all these algorithms are executed in parallel. This work also introduces a visualization stage that permits to display, in an interactive way, the location of each individual and graphically evaluate propagation of the COVID-19 disease. Several optimization techniques are also presented, that aim to reduce the algorithm execution time and to enhance the processing of large workloads consisting of hundreds of cities.

This work provides a comprehensive performance analysis of the most time-consuming stage of the workflow—the coarse-grain spatial generation—proving that it is possible to efficiently execute it in parallel under different configurations. According to the evaluation, a nested parallelism with a balanced distribution of the main and nested threads in combination with a maximum depth parameter leads to the best performance. In addition, the use of dynamic data redistribution schemes reduce the algorithm convergence times and improve the load balance. The results obtained in this work are used for improving the workflow execution in production environments.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Declarations

Ethical approval Not applicable.

Funding This work has been partially funded by the European High-Performance Computing Joint Undertaking (JU) under the ADMIRE project (grant agreement No 956748). This work was also supported by the project grant BCV-2022-2-0010 of the Spanish Supercomputing Network (RES). The role of all study sponsors was limited to financial support. Funding for APC: Universidad Carlos III de Madrid (Agreement CRUE-Madroño 2024).

Availability of data and materials Data sets generated during the current study are available from the corresponding author on reasonable request. The data for population per census sections, income *per capita*, Gini index and density are available at the *Instituto Nacional de Estadística* web page: https://ine.es.

References

1. Beira MJ, Sebastião PJ (2021) A differential equations model-fitting analysis of COVID-19 epidemiological data to explain multi-wave dynamics. Sci Rep 11(1):1–13
2. Alballa N, Al-Turaiki I (2021) Machine learning approaches in COVID-19 diagnosis, mortality, and severity risk prediction: a review. Inf Med Unlocked 24:100564
3. Ojo O, García-Agundez A, Girault B, Hernández H, Cabana E, García-García A, Arabshahi P, Baquero C, Casari P, Ferreira EJ et al (2020) Coronasurveys: using surveys with indirect reporting to estimate the incidence and evolution of epidemics. arXiv preprint arXiv:2005.12783
4. Shattock AJ, Le Rutte EA, Dünner RP, Sen S, Kelly SL, Chitnis N, Penny MA (2022) Impact of vaccination and non-pharmaceutical interventions on SARS-CoV-2 dynamics in Switzerland. Epidemics 38:100535
5. Adamik B, Bawiec M, Bezborodov V, Bock W, Bodych M, Burgard JP, Goetz T, Krueger T, Migalska A, Pabjan B et al (2020) Mitigation and herd immunity strategy for COVID-19 is likely to fail. MedRxiv
6. Singh DE, Olmedo Luceron C, Limia Sanchez A, Guzman Merino M, Duran Gonzalez C, Delgado-Sanz C, Gomez-Barroso D, Carretero J, Marinescu MC (2022) Evaluation of vaccination strategies for the metropolitan area of madrid via agent-based simulation. BMJ Open. https://doi.org/10.1136/bmjopen-2022-065937
7. Sherratt K, Gruson H, Johnson H, Niehus R, Prasse B, Sandman F, Deuschel J, Wolffram D, Abbott S, Ullrich A et al (2022) Predictive performance of multi-model ensemble forecasts of COVID-19 across European nations. medRxiv
8. Guzmán-Merino M, Durán C, Marinescu MC, Delgado-Sanz C, Gomez-Barroso D, Carretero J, Singh DE (2021) Data management in epigraph COVID-19 epidemic simulator. In: Euro-Par 2021: Parallel Processing Workshops: Euro-Par 2021 International Workshops, Lisbon, Portugal, August 30-31, 2021, Revised Selected Papers, pp 267–278. Springer, Berlin, Heidelberg. 10.1007/978-3-031-06156-1_22
9. de Transportes de Madrid C (2018) Encuesta Domiciliaria de Movilidad de la Comunidad de Madrid 2018. Accessed 26 November 2022
10. OpenStreetMap Foundation (2022) OpenStreetMap. https://www.openstreetmap.org. Accessed 26 November 2022
11. Madrid City Council (2022) Population by district, census sections. Accessed 10 December 2022
12. Madrid City Council (2022) Streets by district, census sections. Accessed 10 December 2022
13. Madrid City Council (2023) Per capita income 2013. Accessed 28 November 2022
14. Madrid City Council (2022) Educational level 2022. Accessed 29 November 2022
15. Madrid City Council (2022) Population by nationality 2022. Accessed 29 November 2022
16. Malmusi D, Pasarín MI, Marí-Dell'Olmo M, Artazcoz L, Diez E, Tolosa S, Rodríguez-Sanz M, Pérez G, Peña-Gallardo C, Borrell C (2022) Multi-level policy responses to tackle socioeconomic inequalities in the incidence of COVID-19 in a European urban area. Int J Equity Health 21(1):28

17. Marí-Dell'Olmo M, Gotsens M, Pasarín MI, Rodríguez-Sanz M, Artazcoz L, Garcia de Olalla P, Rius C, Borrell C (2021) Socioeconomic inequalities in COVID-19 in a European urban area: two waves, two patterns. Int J Environ Res Public Health 18(3):1256
18. Gangemi S, Billeci L, Tonacci A (2020) Rich at risk: socio-economic drivers of COVID-19 pandemic spread. Clin Mol Allergy 18(1):1–3
19. Doblhammer G, Reinke C, Kreft D (2022) Social disparities in the first wave of COVID-19 incidence rates in Germany: a county-scale explainable machine learning approach. BMJ Open. https://doi.org/10.1136/bmjopen-2021-049852
20. Grafana Labs (2022) Grafana: the open observability platform. https://grafana.com/. Accessed 26 November 2022
21. Intel (2023) Fix Performance Bottlenecks with Intel® VTune™ Profiler. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html#gs.viumkz. Accessed 26 September 2023
22. Martín G, Singh DE, Marinescu M-C, Carretero J (2015) Towards efficient large scale epidemiological simulations in epigraph. Parallel Comput 42:88–102
23. Reiner RC et al (2021) Modeling COVID-19 scenarios for the united states. Nat Med 27(1):94–105
24. Kerr CC, Stuart RM, Mistry D, Abeysuriya RG, Rosenfeld K, Hart GR, Núñez RC, Cohen JA, Selvaraj P, Hagedorn B et al (2021) Covasim: an agent-based model of COVID-19 dynamics and interventions. PLoS Comput Biol 17(7):1009149
25. Rockett RJ, Arnott A, Lam C, Sadsad R, Timms V, Gray K-A, Eden J-S, Chang S, Gall M, Draper J et al (2020) Revealing COVID-19 transmission in Australia by SARS-COV-2 genome sequencing and agent-based modeling. Nat Med 26(9):1398–1404
26. Hinch R, Probert WJ, Nurtay A, Kendall M, Wymant C, Hall M, Lythgoe K, Bulas Cruz A, Zhao L, Stewart A et al (2021) Openabm-COVID19-an agent-based model for non-pharmaceutical interventions against COVID-19 including contact tracing. PLoS Comput Biol 17(7):1009146
27. Aleta A, Martin-Corral D, Piontti A, Ajelli M, Litvinova M, Chinazzi M et al (2021) Modeling the impact of social distancing, testing, contact tracing and household quarantine on second-wave scenarios of the COVID-19 pandemic. (2020). Publisher Full Text
28. Bubar KM, Reinholt K, Kissler SM, Lipsitch M, Cobey S, Grad YH, Larremore DB (2021) Model-informed COVID-19 vaccine prioritization strategies by age and serostatus. Science 371(6532):916–921
29. Matrajt L, Eaton J, Leung T, Brown ER (2021) Vaccine optimization for COVID-19: Who to vaccinate first? Sci Adv 7(6):1374
30. Franco N (2021) COVID-19 Belgium: Extended SEIR-QD model with nursing homes and long-term scenarios-based forecasts. Epidemics 37:100490

Authors and Affiliations

Aymar Cublier Martínez[1] · **Jesús Carretero**[1] · **David E. Singh**[1]

✉ David E. Singh
dexposit@inf.uc3m.es

Aymar Cublier Martínez
acublier@pa.uc3m.es

Jesús Carretero
jcarrete@inf.uc3m.es

[1] Department of Computer Science, Universidad Carlos III de Madrid, Leganés, Spain