Future Generation Computer Systems 157 (2024) 226–236

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A scalable multi-density clustering approach to detect city hotspots in a smart city

Eugenio Cesario [a,*], Paolo Lindia [a], Andrea Vinci [b]

[a] University of Calabria, Rende (CS), Italy
[b] ICAR-CNR, Rende (CS), Italy

ARTICLE INFO

Keywords:
Multi density-based clustering
Parallel data mining
Smart city

ABSTRACT

In the field of Smart City applications, the analysis of urban data to detect city hotspots, i.e., regions where urban events (such as pollution peaks, virus infections, traffic spikes, and crimes) occur at a higher density than in the rest of the dataset, is becoming a common task. The detection of such hotspots can serve as a valuable organizational technique for framing detailed information about a metropolitan area, providing high-level spatial knowledge for planners, scientists, and policymakers. From the algorithmic viewpoint, classic density-based clustering algorithms are very effective in discovering hotspots characterized by homogeneous density; however, their application on multi-density data can produce inaccurate results. For such a reason, since metropolitan cities are characterized by areas with significantly variable densities, multi-density clustering approaches are more effective in discovering city hotspots. Moreover, the growing volumes of data collected in urban environments require the development of parallel approaches, in order to take advantage of scalable executions offered by Edge and Cloud environments. This paper describes the design and implementation of a parallel multi-density clustering algorithm aimed at analyzing high volumes of urban data in an efficient way. The experimental evaluation shows that the proposed parallel clustering approach takes out encouraging advantages in terms of execution time, speedup, and efficiency.

1. Introduction

In recent years, Edge Computing has emerged as a major computing paradigm in several contexts, such as multimedia streaming, autonomous vehicles, cognitive buildings, and Smart Cities [1–3]. In this paradigm, data is stored and elaborated as close as possible to the sources of information to improve response times and save bandwidth. For instance, in Smart Cities, several edge computing nodes can be deployed in different city areas to collect and elaborate information, e.g., on traffic [1], drainage network [2], and power grids [3], in order to fast provide knowledge about the city, optimize energy consumption, or perform local control on infrastructures [4,5].

The integration of Edge and Cloud Computing paradigms is leveraging the development of several solutions, which are shown to be particularly useful in the field of distributed machine learning methods: computation and data are spread between the edge and the cloud in order to produce efficient, scalable, and accurate prediction models [3,4,6–8]. In fact, interconnected edge nodes provide a first layer of computation, which can be exploited to filter and aggregate data, which is then stored and analyzed in a remote cloud layer. This approach aims to reduce network and latency congestion, as each edge node computes its own local data, by reducing the exchange of information with the other nodes and the cloud as much as possible. Such a locality-based principle can proficiently reduce communication network congestion, and improve the computational efficiency and scalability of the applications [9]. However, such heterogeneous computing infrastructures require novel approaches to enable distributed and parallel machine learning analysis that can leverage both edge computing and cloud resources.

In the field of Smart City applications, the detection of *city hotspots* is becoming a more and more popular task [10,11]. Given the availability of geo-referenced data, urban hotspots may be thought of as homogeneous zones in spatial data, each with its own characteristics, such as density, distribution, and boundaries. The detection of different hotspots in a city is useful to extract specific spatial knowledge in large metropolitan regions, studying each hotspot individually, or discovering relationships between distinct hotspots. Such knowledge can provide high-level summaries for spatial datasets, which is valuable information to support the work of scientists, city planners, and policymakers [12,13]. For example, information collected through sensor networks can be utilized by city managers to gain valuable insights into

* Corresponding author.
E-mail addresses: eugenio.cesario@unical.it (E. Cesario), paolo.lindia@dimes.unical.it (P. Lindia), andrea.vinci@icar.cnr.it (A. Vinci).

https://doi.org/10.1016/j.future.2024.03.042
Received 26 September 2023; Received in revised form 1 March 2024; Accepted 26 March 2024
Available online 28 March 2024

the condition of urban infrastructures, enabling prompt or predictive maintenance actions on roads, water and power delivery systems [14]. As another instance, in environmental analysis scientists typically divide a city into zones depending on environmental variables such as pollutant density or atmospheric conditions. In epidemic analysis, the detection of hotspots is useful to forecast spreading trends [15,16] in cities and countries. In crime analysis, police departments are interested in identifying locations with similar crime behaviors, so as to characterize the urban territory and forecast crime trends [17]. This can be exploited to efficiently deploy officers over the urban territory and to better govern the municipality in terms of public safety. However, several studies warn against the so-called pre-crime effect [18,19], thus policymakers and administrators should take advantage of these capabilities avoiding discrimination, ethnic profiling, the neglection of paramount principles such as the presumption of innocence, and pre-emptive security policies.

Among various techniques for spatial analysis, density-based clustering algorithms have been proven highly effective in identifying urban hotspots within a city. Usually, these algorithms have the capability to automatically detect the number of clusters, as well as to identify clusters of arbitrary shapes and sizes, which are useful properties to effectively discover spatial hotspots in urban territories [10,20,21]. In particular, since the density of population, traffic, or events in cities can largely vary from one area to another area (thus showing multi-density distributions of points), several studies show that multi-density-based approaches outperform classic algorithms to discover urban hotspots in metropolitan cities [10,11,22,23].

The multi-density distribution of population in metropolitan cities is also quantitatively confirmed by the coefficient of variation cv, representing the relative standard deviation of urban population density. Specifically, for a given city, cv is formally defined as $cv = \sigma/\mu$, where σ represents the standard deviation of the population density within the city, and μ is the average population density. Thus, the coefficient of variation is a dimensionless measure estimating the variation in population density within a city. A higher cv value indicates a more evident non-uniformity in the population density of a city. On the basis of computed cv values, a recent study presented in [20] highlights the considerable variability in densities within metropolitan cities, highlighting the appropriateness to apply multi-density detection algorithms for the accurate detection of city hotspots.

Working on such a research activity, we recently developed a multi-density clustering algorithm to discover spatial hotspots in urban environments [20]. In particular, the algorithm has been specifically designed to analyze data generated in metropolitan cities, whose density data distributions are heavily characterized by high variability, making de facto classic density-based clustering detect inaccurate city hotspots. More in detail, the algorithm proposed in [20] overtakes this issue through a preliminary computation of density variations among data points, and by performing a preliminary partitioning of the data into several density level sets (each one characterized by homogeneous density distributions). However, although the algorithm demonstrates good results in terms of clustering accuracy, some steps of the approach are computationally intensive and require parallelization to leverage the scalable execution offered by an Edge–Cloud environment. This arises from two main reasons: firstly, the computational cost of processing large data volumes, and secondly, the geographical distribution of computing nodes in urban Edge–Cloud infrastructures. In fact, considering the increasing data volumes collected in urban environments, the analysis of such large datasets on a single machine could take a very long processing time to get results; consequently, employing parallel data mining approaches becomes essential to enhance computational efficiency and scalability. Furthermore, the widespread availability of nodes in Edge–Cloud infrastructures provides access to a distributed pool of computing resources on the Edge, offering an effective means to run parallel and distributed algorithms.

This paper describes the design and implementation of a parallel multi-density clustering algorithm, capable of discovering spatial hotspots from urban data and suitable for deployment and run in an edge–cloud environment. Specifically, the proposed solution (described by a workflow formalism) is the parallelization of CHD (City Hotspot Detector), i.e., a sequential multi-density clustering approach we proposed in [20], aimed at analyzing a high volume of data in an efficient way. Briefly, the CHD algorithm is composed of several steps, as follows. First, the neighborhood density for each point is estimated by calculating its K-nearest neighbors' reachability distance. Then, the points are sorted w.r.t. their estimated density, and the density variation between each consecutive couple of points in the ordered list is computed; eventually, a rolling mean operator is applied to smooth density variation values, thus making variations more stationary. On the basis of such smoothed density variations, the points are partitioned into several density level sets (DLSs), each one characterized by homogeneous density distributions. Then, each density level set is analyzed by a specific density-based clustering algorithm instance (whose input parameters are automatically detected on the basis of the specific DLS density), to detect clusters in the data partition. The final result of the algorithm consists of a set of spatial clusters, each one representing an urban hotspot. Some of the aforementioned steps are very intensive from a computational viewpoint, particularly when the analysis involves large data volumes. For such a reason, we propose a scalable solution of CHD whose the k-nearest distance computation and the multiple DB-SCAN executions (which are the most time-consuming and critical steps) are parallelized, with the aim to achieve higher performances and improve the scalability of the approach.

In order to show a concrete scenario to which such an approach can be applied, we will present as a case study the analysis performed on a real-world dataset collecting geo-referenced crime data of Chicago, and we plot the detected crime hotspots on a map. Also, a scalability analysis has been performed on large synthetic datasets (up to eight million instances), to deal with high orders of magnitude under different settings and with respect to several data sizes. The experimental evaluation demonstrates that the parallel implementation produces good results in terms of execution time, efficiency, and speed-up.

The rest of the paper is organized as follows. Section 2 outlines related work in the area of machine learning based on edge–cloud computing for smart city applications, and reports a review of the most important approaches and applications in the multi-density spatial clustering literature. Section 3 presents the problem formulation and the parallel multi-density clustering algorithm proposed in the paper together with the designed workflow. Section 4 describes the results achieved on a real-world case study. Section 5 provides the experimental evaluation of the approach, in terms of execution time, efficiency, and scalability. Finally, Section 6 concludes the paper and plans future research works.

2. Related work

Several works have been proposed in the literature to show how edge-based machine learning can provide useful algorithms in several large-scale smart environments, such as Smart Power Grid, Smart Buildings, and Smart Cities [4–6]. In particular, in the field of Smart City applications, the detection of city hotspots has become one of the most relevant issues in urban environments [10,24–26]. With the aim of providing a summary of the most representative research works in this field, firstly we will review here some edge-based machine learning applications in Smart environments (Section 2.1), and then we will describe the most important approaches in the density-based spatial clustering literature (Section 2.2).

2.1. Edge computing and machine learning.

Machine learning based on edge computing for Smart Cities is a hot topic that has been covered in several works and applied in

real case studies [4–6,8]. In [4] a distributed near real-time traffic forecasting application for edge computing environments is proposed, based on a data distribution algorithm and a traffic forecasting model for floating car data, to be deployed in a city-wide fog computing infrastructure. The validation showed that the approach running on an edge environment achieves higher efficiency than other cloud-specific approaches in terms of computation distribution and reliability, especially when connectivity issues may occur. In another recent representative paper [5] a review of existing edge-based applications is given, in the contexts of Smart Power Grid, Smart Power Management, and Smart Waste Management. Ref. [27] presents a review study on the most widely used approaches for moving computation at the edge of the network, in order to make machine learning algorithms take advantage of computing capabilities at the edge server level and at the device level. In particular, they focus on operational aspects, including data compression techniques, tools, frameworks, and hardware used in concrete intelligent edge systems.

2.2. City hotspots detection.

The detection of city hotspots, i.e., urban areas in which events of interest occur with high density, is a common task when analyzing urban datasets [10,24–26]. For this purpose, several sequential and parallel density-based approaches have been proposed in the literature. Most of them are based on classic techniques, i.e., the well-known DBSCAN and OPTICS algorithms [28–31], while others rely on multi-density clustering techniques [10,20,22,23]. In this section, we will briefly review some of the most representative research work in both areas, also detailing those approaches that can be suitable for parallel execution leveraging distributed edge–cloud computing nodes.

2.2.1. Classic density-based approaches.

The DBSCAN algorithm [28] is based on the concepts of core-points and density-reachability. In this algorithm, a core-point is defined as a point with at least a specific number of neighboring points, denoted as $minPts$, within a certain radius ϵ around it. A point is considered reachable from a core point if it falls within its radius ϵ (directly reachable) or if a path of directly reachable core points connecting the core point to the specified point exists. DBSCAN forms clusters by grouping data points that are connected to each other, while points not connected to any core-points are labeled as outliers.

The OPTICS algorithm [29] (Ordering Points To Identify the Clustering Structure) aims at creating an ordering of the data points by reachability distance, given a $minPts$ parameter. Such ordering is exploited to make a reachability plot, containing information about the density-based clustering structure, which supports the analyst in choosing the best ϵ parameter. The same ordered structure is exploited to fast produce the same results of the DBSCAN clustering, after fixing the ϵ parameter. Compared to DBSCAN, the data points ordering task requires more resources in terms of computational power, but the detection of a cluster is much faster.

Considering the algorithms described above, their parallel implementations have been proposed in [30] (DBSCAN-MR) and [31] (RP-DBSCAN). Both approaches are based on data partitioning and map-reduce patterns, and their efficiency has been assessed through a set of experiments on large datasets. Furthermore, in [32] authors propose a parallel approach based on DBSCAN to perform two-dimensional data clustering, by partitioning the points on the basis of grid framing, and using several procedures (Delaunay triangulation, unit-spherical emptiness checking using line separation, and bichromatic closest pairs) to detect connectivity among core points. Also, they perform a wide range of studies on synthetic and real-world datasets with various parameters. In [33] authors propose "*HY-DBSCAN*", which is a scalable DBSCAN solution aimed at overtaking several computational issues of the sequential implementation. In particular, the load balancing is improved by exploiting a modified kd-tree to distribute points across blocks; also, during local clustering, grid-based spatial indexing is used for region queries; and, finally, for cluster merging, the authors propose a distributed Rem's Union-Find algorithm that resolves communication deadlocks and converges to the correct result in a finite number of iterations.

In [34] authors propose PDSDBSCAN, i.e., a parallel disjoint-set based implementation of DBSCAN, which exploits graph algorithmic concepts and disjoint-set data structures to break the sequential data access order exhibited by classic DBSCAN executions. The main idea of the approach is that each node runs a sequential DBSCAN instance on its local data, to perform parallel computations of local clusters without requiring communication among the nodes. In particular, clusters are built by employing a tree-based bottom-up approach and modeled as a tree-based structure. Then, local clusters (i.e., trees) are merged to obtain the final clusters.

Another parallel version of DBSCAN is P-DBSCAN [35], which exploits the PR-tree (Priority R-tree) as a spatial index data structure. The algorithm first splits the input dataset into several parts; then, each computational node builds a local PR-tree and executes its clustering task on its local data independently from the other running nodes; finally, the sub-results are aggregated into one final model. Authors declare that P-DBSCAN achieves higher efficiency than sequential solutions because the PR-Tree achieves higher computational efficiency in query answering than the R*- and R-tree indexes [35].

Ref. [36] analyzes the computational bottlenecks of DBSCAN and its inherent sequential control flow dependency at the point of the recursive expansion. To overtake such limitations, authors propose HPDBSCAN, a parallel approach of density-based clustering employing three major techniques in order to break the sequentiality of DBSCAN, i.e., (i)a computation split heuristic for domain decomposition, (ii) a data index preprocessing step and (iii) a rule-based cluster merging scheme. The experimental evaluation of HPDBSCAN, performed as an OpenMP/MPI hybrid application [36], shows good results in terms of computation time and memory consumption.

A parallel version of DBSCAN based on SPARK is proposed in [37]. Authors first recognize that the large amount of distance computations between each node pairs, which is required to detect the core points and their neighbors, is the main computational bottleneck of the whole algorithm. To overtake it, authors implement parallelization mechanisms on Spark, and exploit an optimized partition method based on RTree, and therefore perform several local DBSCAN runs parallelly on data partitions and merge the local results in a final clustering model.

2.2.2. Multi density-based approaches.

The afore-described approaches are all based on the DB-SCAN algorithm, thus their application on multi-density datasets may not provide proper results, as they are not suitable for discriminating clusters on the basis of different densities. Several approaches have been proposed in the literature to overcome this issue. In [10] the VDBSCAN algorithm (Varied Density Based Spatial Clustering of Applications with Noise) is proposed, aiming at detecting clusters having different densities. VDBSCAN computes an ordered list of k-dist values for each object, where each k-dist element is the minimum distance so that k points are included in the object's neighborhood. This ordered set can be visualized and exploited to detect the sharp changes of the k-dist values corresponding to a list of several radius values, in order to generate clusters with different densities.

The approach proposed in [23], namely K-DBSCAN, evaluates the density of each data point (similarly to KDDCLUS), identifies k sets having similar densities, and associates the same ϵ value to the elements of the same set, thus producing k ϵ-values. These are exploited to execute a variation of the DBSCAN, which takes into account different ϵ values for each data point, thus detecting varied-density clusters. However, similarly to the k-means, the approach is very dependent on a proper setting of the parameter k, which is crucial to specify the number of the different ϵ values to be evaluated.

Algorithm 1 The CHD Algorithm [20].

Require:
D: urban event dataset;
k : an integer value to compute the k-neighborhood density;
ω : a coefficient for the density variation threshold;
s : smoothing window size;
Ensure:
$UH = \{uh_1, uh_2, \ldots, uh_H\}$: a set of H city hotspots;
1: $kDistList \leftarrow$ ComputeKDist(D, K)
2: $denVarList \leftarrow$ ComputeDensityVariation($kDistList$)
3: $smoothDenVarList \leftarrow$ MovingAverageFiltering($denVarList$, s)
4: $\tau \leftarrow \mu(smoothDenVarList) + \omega \cdot \sigma(smoothDenVarList)$;
5: $densityLevelSetsList \leftarrow$ PartitionDensityVariation(D,$smoothDenVarList$,τ);
6: epsList $\leftarrow$ ComputeEpsValues($densityLevelSetsList$);
7: **for** each ϵ_i *in* $epsList$ **do**
8: $DLS_i \leftarrow densityLevelSet[i]$
9: $(uh_{\epsilon_i}) \leftarrow$ DBSCAN(DLS_i, ϵ_i, k)
10: $UH \leftarrow UH \cup uh_{\epsilon_i}$
11: **end for**
12: **return** (UH)

Ref. [22] presents an algorithm to perform big data clustering with varied density, using a Hadoop platform running MapReduce. The main idea of this research study is the use of a local density detection approach to find each point's density. This has been implemented by integrating the MR-VDBSCAN approach [38] in the architecture proposed in [22], to detect clusters with varying densities. Also, authors rely on the application of a load balancing strategy to deal with large-scale datasets, in order to achieve efficient speed-up and scale-up for skewed big data.

In [39], authors proposed an adaptive Multi-density DBSCAN algorithm (AMD-DBSCAN) with an improved parameter adaptation method to search for multiple parameters pairs,i.e., Eps and MinPts, that allows the model to detect clusters with different density levels. The proposed approach is a novel exploration of k_{dist} value, which is the kth nearest neighbor of a point, to search for multiple parameter pairs matching the distribution of the dataset. Once the lists of candidate Eps and MinPts have been obtained, DBSCAN is executed on the different subsets of data.

The work presented in [17] describes CHD (City Hotspot Detector), an approach based on multi-density clustering, which has been specifically designed to discover urban hotspots in a city. In particular, to deal with the high density variability among different areas of urban environments, a moving average filtering technique is exploited to smooth out density fluctuations and highlight main trends. An extensive comparative analysis of CHD with other state-of-art density-based clustering algorithms, on both state-of-the-art and real-world datasets, has been reported in [11,20], showing that it detects higher quality city hotspots than other classic density-based approaches proposed in the literature. In [40] a parallel solution of CHD has been sketched, to achieve higher performance in terms of scalability. Some preliminary tests, conducted up to a low number of parallel nodes, have shown encouraging results in terms of execution time and speedup.

2.3. New material with respect to the conference paper.

The current paper largely extends the work presented in [40] and provides several original contributions with respect to the previous one, as summarized in the following. Section 3 has been extended (w.r.t. the conference paper) and the algorithm workflow description has been enhanced, by providing more details about the steps to improve the performance of the approach. Section 4 reports as a case study the detection of crime hotspots in the whole city of Chicago, to show the usefulness of the algorithm in a real-world application, which is an original contribution of this paper and has not been published elsewhere. Section 5 has been largely enhanced in several parts. First, the experimental testbed has been extended from 4 up to 32 nodes, to perform a scalability analysis on a higher number of servers. Then, tests have been performed on higher data volumes, by extending the size of input data from one-hundred thousand to eight million instances, in order to carry out a scalability performance analysis on a very large dataset. Finally, an extensive experimental analysis has been devoted to investigating how the algorithm performance, in terms of execution time, speed-up, and efficiency is related to the number of computing nodes and the dataset size. Figs. 5, 6, 7, 8 and 9 are completely original figures of this paper and have not been published elsewhere. Finally, Sections 1 and 2 have been extended and modified in several parts, and the whole paper has been carefully restructured.

3. Multi-density urban hotspots detection: Sequential and parallel approaches

This section first presents the City Hotspot Detector (CHD) algorithm proposed in the paper [20] by discussing the problem formulation and summarizing the related sequential algorithm. Then, we illustrate a parallel version of CHD, based on the distributed computation of the most computing intensive tasks on parallel running nodes.

3.1. Problem formulation

Let D be a dataset collecting spatial urban data instances, $D = \{d_1, d_2, \ldots, d_{|D|}\}$, where each d_i is a data tuple described by $\langle latitude, longitude \rangle$, where $latitude$ and $longitude$ are the coordinates of the place the event occurs. For our purpose, urban events can be traffic spikes, crimes, viral infections, pollution peaks, etc., that is, any event that happens and is localized in urban areas. The goal of the analysis is to discover a set UH of urban hotspots, $UH = \{uh_1, uh_2, \ldots, uh_H\}$, where an urban hotspot uh_h is a spatial contiguous area characterized by events having higher density with respect to other areas in the city.

3.2. A sequential approach

The meta-code of the sequential approach for discovering multi-density urban hotspots is reported in Algorithm 1. The algorithm receives in input the dataset D, the integers k and s, and the real number ω. Specifically, k is an integer value to tune the k-neighborhood density, ω is a tuning coefficient to compute the density variation threshold, s is a smoothing parameter exploited for smoothing density

variation levels (as explained later in the section). The algorithm computes and returns a set of urban hotspots $UH = \{uh_1, \ldots, uh_H\}$, where each hotspot can be characterized by a different density w.r.t. other hotspots.

The algorithm begins by computing, for each point d_i, the *k-nearest neighbor distance* of d_i, given a certain k. This is performed by the ComputeKDist(D, k) method, which computes the distance between each $d_i \in D$ and its kth-nearest neighbor points (line 1). The method returns the $k-distList$ data structure, i.e., a list storing in ascending order the *k-nearest neighbor distance* of each point $d_i \in D$. It is worth noting that the *k-nearest neighbor distance* value of a certain point d_i is proportionally related to the point density: the higher such a distance, the lower the density of points around d_i [41]. As this step is completed, the Compute-Density-Variation($k - distList$) method computes the *density variation list* of each point d_i with respect to the next point d_{i+1} in the sorted $k - distList$ and returns the *density variation list* (line 2). The resulting density variation is then smoothed by applying a moving average filter on a window size s (line 3), in order to make more stable the density variation and to properly highlight peeks and deeps. On the basis of the computed smoothed density variation list, the density variation threshold τ is computed (line 4). Then, the PartitionDensityVariation method builds a list of *density level sets* (line 5), where a *density level set* consists of a partition of data points that are consecutive in the density variation list and that have a density variation lower than τ. Thus, the points are divided into several density level sets (DLS)s, each one characterized by a density distribution. At this point, the Compute-Eps-Values method computes the ϵ values, by evaluating the level-turning line for each density level set (line 6). Such values are stored and returned in the $\epsilon - list$, i.e., a list of ϵ values that are estimated as the best values with respect to the different densities in the data [41]. Finally, each density level set DLS_i is processed by the DBScan clustering algorithm with parameters (ϵ_i, k), and the discovered clusters UH_{ϵ_i} are added to the final cluster set (lines 7–11). All non-marked points are recognized as noise. The final result UH consists of a set of spatial clusters, each one representing an event-dense urban hotspot, detected by different ϵ-value settings (i.e., by different densities).

3.3. A parallel approach

The sequential approach for discovering multi-density urban hotspots described in the previous section is a sequence of concatenated steps, some very intensive from a computational viewpoint. However, some steps can be naturally parallelized in order to achieve higher performances. In particular, as it will be better shown in the experimental evaluation section (Section 5), both the *k-nearest distance computation* (line 1) and the *multiple DB-SCAN executions* (lines 7–11) steps are the most time-consuming and critical ones. For such a reason, our effort consists in the parallelization of these two steps, which has been done by implementing a *Single Program Multiple Data (SPMD) parallelism* pattern [42,43]. More precisely, in SPMD parallelism, the large-scale data to be processed is split among the n available processors, each executing in parallel the same computation (or algorithm) over a designated subset of the input dataset. The final result is obtained as a combination of the local models outputs produced by the n algorithms, as shown in Fig. 1. In our case, we exploited the SPMD paradigm to parallelize both the *k-nearest distance computation* and the *multiple DB-SCAN executions* steps. In the first one, the original urban dataset is horizontally split into several subsets, then the subsets are distributed to the computational nodes to parallelly compute the kth-nearest neighbors of the points in each subset. Instead, for the DBSCAN executions, the input data points are preliminarily partitioned into several density level sets (DLSs), then several DBSCAN instances are run on each specific partition in parallel to detect the clusters from each DLS; afterwards, the final clustering model is obtained by joining the clusters detected by each local computation.

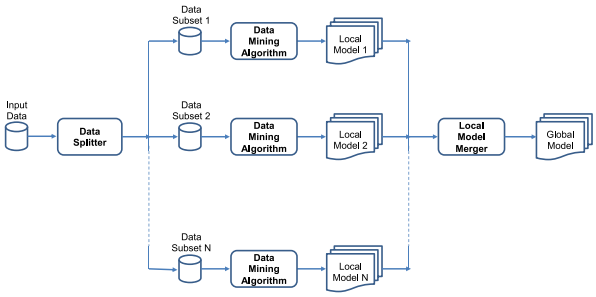


Fig. 1. SPMD parallelism pattern.

Now, in order to have a clear view of the whole parallel approach we propose in this paper, Fig. 2 shows it by exploiting the workflow formalism, i.e., a graph in which nodes represent data sources, algorithms, and outputs, and edges represent execution dependencies among nodes. The original dataset D contains urban data instances (represented in the previously described format) of events that occurred in an urban environment. In particular, let us suppose that the original dataset is composed of N instances, each one represented by $\langle latitude, longitude \rangle$-couple. The workflow comprises six steps (see Fig. 2), each one described in the following.

Step 1 — Data Mapping. The original urban dataset is partitioned by the *Data Mapper* in M horizontal partitions $D_1, \ldots, D_M$, whose size (i.e., number of instances) is balanced among them, and thus each consisting of $|D_i| \cong \frac{1}{M}|D|$ data points, for $i = 1, \ldots, M$. The partitions are then distributed to the computational nodes to compute the partials k-nearest-neighbors of each data point (see Step 2), assuring that intra-partition and inter-partition comparisons are performed for all data points, and hence ensuring that each couple of partition $\langle D_i, D_j \rangle$ with $i \geq j$ is processed at least once by a computing node. Thus, each computing node elaborates a subset $S \in \mathcal{S}$ of the partitions $D_1, \ldots, D_M$, where $s \in S$ is a data point, and $|S| = N$ (N is the number of available computing nodes). It is worth noticing that this data partitioning and mapping is an additional step with respect to the sequential case (where no splitting step is contemplated), and it is aimed at improving the scalability of the whole approach.

Step 2 — Partial K-nearest-neighbors computing. This step is aimed at computing the kth-nearest neighbor of each point in each subset S of partitions distributed to nodes, given a certain k parameter. Specifically, for each data point x in the subset S, the set knn_{xS} is computed, such that $knn_{xS} \subseteq S$, $|knn_{xS}| = k$, and $\forall x' \in S \backslash knn_{xS}$, $dist(x, x') \geq \max_{x'' \in S} dist(x, x'')$. In the workflow, this is done by running N k-nearest-neighbor computations, each one taking in input one of the S subsets built at the previous step. The result consists of a set $\{knn_1, \ldots, knn_N\}$, where each knn_i contains the *k-nearest neighbors* of each $x \in S_i$ with respect to S_i.

Step 3.1 — K-distance List Computation. This step aims to compute the global k-distList for each $d_i \in D$, by merging the information computed in the previous step. First, for each $d_i \in D$, a set of knn_{d_i} is computed by searching the k-nearest neighbors of d_i in all the sets knn_S where $d_i \in S$. Then, the k-distance of each d_i is computed as $\max_{x' \in knn_{d_i}} dist(d_i, x')$. In this way, the final result is the list storing the *k-nearest neighbor distance* of each point $d_i \in D$.

Step 3.2 — Smoothed Density Variation Evaluator. This step is aimed at performing the computations defined on line 3 of Algorithm 1, in order to make more stable the density variation and to properly highlights peeks and deeps. The result is the smoothed density variation list, described in Section 3.2.

Step 3.3 — Density Level Sets Synthetizer. This step is aimed at performing two sequential tasks. First, the density variation threshold τ is computed on the basis of the computed smoothed density

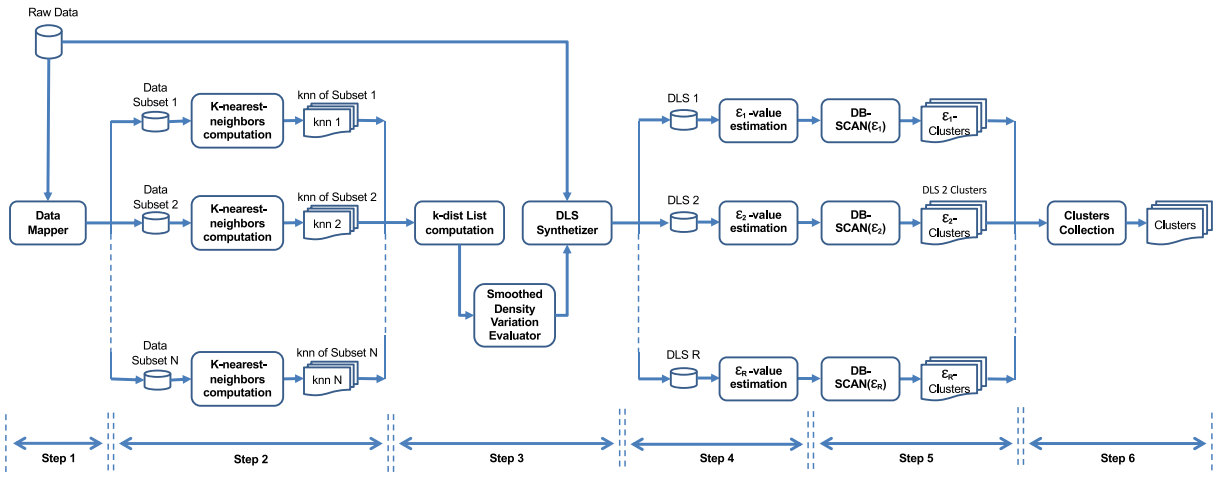
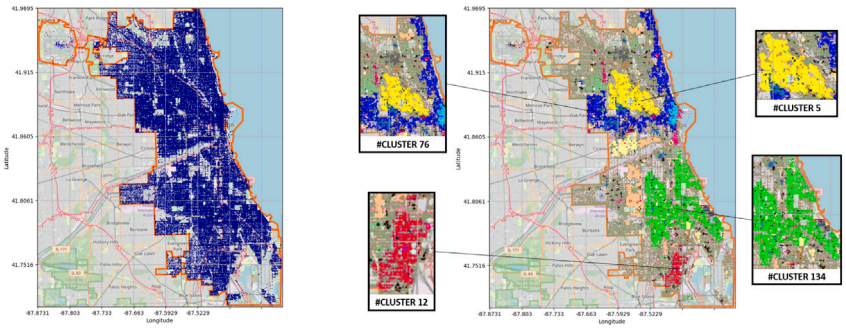


Fig. 2. Multi-density algorithm workflow.



(a) Geo-localized crimes (b) Crime hotspots

Fig. 3. Geo-localized crime events and detected crime hotspots in Chicago (2001–2022).

variation values. As a consequence of that, a list of the *density level sets* $DLS = \{DLS_1, \ldots, DLS_R\}$, i.e., collections of data points whose density variations are lower than τ, is built.

Step 4 — Eps-list Computation. In this step, for each density level set DLS_i, an ϵ_i value is estimated. According to [41], for a certain density level set DLS_i, its corresponding ϵ_i is chosen by considering the maximum $k-distance$ of points belonging to DLS_i.

Step 5 — Several DB-Scan Executions. For each ϵ_i (computed at the previous step), a DBSCAN instance is executed on the corresponding density level set DLS_i. These runs are performed in parallel, by exploiting the available N nodes. The parallelization of this step is crucial to the whole scalability of the approach, because it is the most relevant time consuming stage of the entire process. More precisely, this step is performed in parallel on the available N computing nodes, where each DLS_i is analyzed by an instance of DBSCAN(ϵ_i, min_pts) and produces a clustering model UH_{ϵ_i}. Each clustering model is a set of clusters/dense regions detected on the specific density level set.

Step 6 — Clusters Collection. All discovered clusters $\{UH_{\epsilon_1}, \ldots, UH_{\epsilon_R}\}$ are added to the final cluster set $UH = \cup_{i=1\ldots R} UH_i$, where all non-marked points are recognized as noise in the final clustering model. The final result consists in a set of spatial clusters, each one representing an event-dense urban hotspot, detected by different ϵ-value settings (i.e., by different densities).

4. Application of CHD on a real case-study: detecting crime hotspots in Chicago

To show a real-world application of the algorithm described above, we show as a case study the analysis performed to detect crime hotspots in the whole area of Chicago. The analysis has been performed on the *'Crimes - 2001 to present'* dataset, available on the *Chicago Data Portal*,[1] which is a real-life collection of instances describing criminal events that occurred in Chicago from 2001 to the present. Each instance corresponds to a geo-localized crime event, described by longitude and latitude attributes. From the whole dataset, we selected the crime events occurred in the city from 2001 to 2022, totally amounting to about six million instances. The considered city boundaries and the collected geo-localized crime events are shown in Fig. 3(a). The area consists of several urban districts, each characterized by different densities of population, economic activities, crimes, etc. (thus an interesting task for multi-density hotspot analysis).

The execution of the algorithm on the aforementioned dataset has produced a set of spatial clusters, each one representing an urban hotspot, as shown in Fig. 3(b). Interestingly, this image shows how crime events are clustered on the basis of a density criteria; the four most significant hotspots are zoomed-in on the left and rights sides of Fig. 3(b), clearly recognizable in several parts of the area (colored in green, red, blue and yellow). Many other hotspots are detected, representing areas having minor crime-densities w.r.t. the highlighted ones, or local high-density crime zones surrounded by low-density ones. Such results have been achieved by fixing $\omega = -0.27$, $k = 64$, $s = 5000$, which have been assessed to best suit our application scenario. The analysis of how the parameter setting affects the results quality has been deeply studied in [11] and is out of the scope of this paper.

In order to make the experimental execution workflow more clear, Fig. 4 sketches the steps of the approach, by graphically showing the

[1] Source: https://data.cityofchicago.org/.

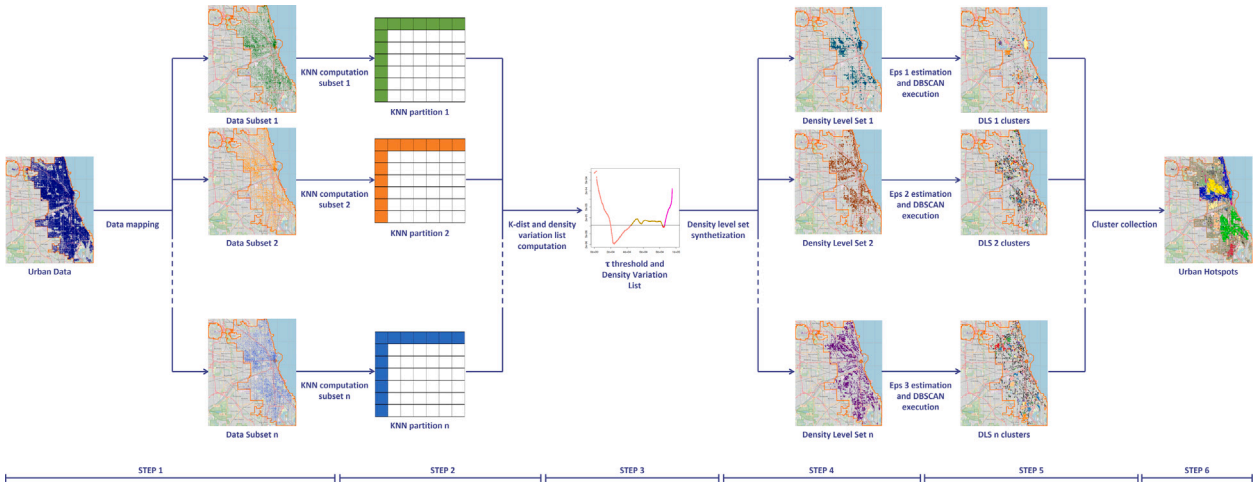


Fig. 4. Experiment Workflow.

data involved during the running steps. The input geo-referenced data is split in several subsets (step 1), to compute the partial k-nearest neighbors of all points (step 2). Then, the k-distance and density variation lists are computed and, on the basis of the τ value, the points are partitioned on several density level sets (step 3). For each density level set, an ϵ value is estimated (step 4) and a DBSCAN instance is executed, to discover the clusters in each DLS (step 5). Finally, all discovered clusters are collected and aggregated in a final global clustering model (step 6).

5. Experimental evaluation and results

To evaluate the scalability of the multi-density clustering approach described in Section 3, we conducted a comprehensive series of experiments by testing the parallel approach in various scenarios. We have tested the parallel approach on synthetic datasets to deal with higher orders of magnitude, which are not achievable on the real dataset due to its limited cardinality, under different settings and with respect to several data sizes (up to eight million events).

In the rest of the section, we will first describe the experimental setting, the synthetic dataset exploited to run the tests, and the performance metrics exploited to assess the performance of the approach (Section 5.1). Then, we will present the results of the evaluation investigating the execution time, the efficiency, and the scalability when varying the number of nodes and data volumes (Section 5.2).

5.1. Experimental setting and performance metrics

The approach described in the previous section has been developed in Python and leveraging the `scikit-learn` library. Experiments were performed on a cluster, located at the University of Calabria (UNICAL), composed of thirteen nodes each having four AMD Opteron(TM) 6376 processors (16 CORE, 2.3 GHz) and sixteen 16 GB RAM modules (256 GB RAM). The total number of cores is 832 (on 52 processors), and the total RAM size is 3328 GB.

In particular, model parallelization has been implemented by combining the two libraries `Dask.distributed` and `joblib`. `Dask.distributed` is a centrally managed, distributed, dynamic task scheduler. The central `dask scheduler` process coordinates the actions of several `dask worker` processes, which are spread across multiple machines, and the concurrent requests of several clients. `joblib` provides a simple helper class to write parallel for loops using multiprocessing. The core idea is to write the code to be executed as a generator expression and convert it to parallel computing. `Dask` can scale Joblib-backed algorithms out to a cluster of machines by providing an alternative to the default Joblib backend which is capable of running parallel tasks only in a single computing environment. The choice of the Joblib library, which natively offers thread- and process-based parallelisms, has been natural since it is used to write several Scikit-Learn algorithms for parallel execution [44]. The source code implementing the Parallel CHD algorithm shown in this paper is made available to the research community.[2]

In order to evaluate the proposed approach, we developed an ad-hoc data generator to produce synthetic data (named Chess data). More in detail, to deal with different data sizes, we built several datasets, composed by 1M, 2M, 4M, and 8M instances respectively. Each dataset has been generated by first building a set of pre-defined spatial squared cells, and then populating each cell by points whose density is randomly selected within a set of 64 possible different densities. The final result is a multi-density dataset, composed of squared cells having different densities. The points within each cell are randomly positioned according to a uniform probability distribution, on both the vertical and horizontal axes. As an instance, Fig. 5 shows the 1M instances Chess dataset, where cell densities vary from 64 points per cell to 1764 points per cell. The data generator code is also available to the community.[3]

Finally, let us introduce the performance metrics adopted during our tests. As a matter of fact, the goal of the evaluation is to assess the execution time and scalability of the algorithm, by analyzing the time taken by each step and comparing the performances achieved by both sequential and parallel executions. In particular, we evaluated the results by exploiting the following performance metrics:

- *Execution time*: the total execution time of the distributed algorithm varying the number of running nodes, that is, the elapsed time from task submission until the final result is returned to it; formally, the time taken by the algorithm to execute on a single node is called the *sequential execution time* and it is denoted by ET_1, while the execution time of the corresponding parallel implementation on n identical nodes is called the *parallel execution time* and it is denoted by ET_n.
- *Speed-up*: the ratio of the turnaround time elapsed by exploiting one node to the turnaround time on n nodes, which measures how much performance gain is achieved by parallelizing a given application over a sequential implementation; formally, the speedup on n nodes is defined as $S_n = \frac{ET_1}{ET_n}$.

[2] The Parallel CHD implementation exploited for evaluating the approach is available here: https://gitlab.com/chd3/parallel-chd.

[3] https://gitlab.com/chd3/datasets

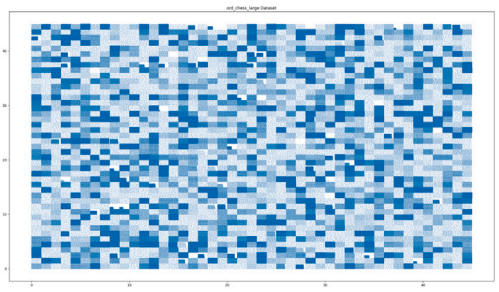


Fig. 5. Chess Synthetic Dataset, 1 million points.

Table 1
Experimental setting values.

Symbol	Meaning	Values
D_x	x instance dataset	1M–8M
N	n. of nodes	1–32
ω	density variation threshold	−0,27
k	k-nearest neighbor distance parameter	64
s	smoothing window size	5000

- *Efficiency*: the ratio between speedup and the number of processing nodes, which measures the percentage of time for which processing nodes are usefully exploited for computation (and not for communication tasks or even idling); formally, the efficiency on n nodes is defined as $E_n = \frac{S_n}{n}$.

5.2. Scalability analysis

The performance evaluation of the approach, to assess the execution time, scalability and efficiency of the whole task, has been carried out by running our tests varying the data size and the number of computing nodes. Specifically, from the whole dataset we created four data partitions composed of 1, 2, 4 and 8 million instances, referred in the following as D_{1M}, D_{2M}, D_{4M}, and D_{8M}, respectively. Those four datasets have been used in the experimental evaluation. Table 1 reports a summary of the experimental setting, with dataset sizes, number of nodes, and the input parameters values fixed during the tests.

As an initial result, Fig. 6 shows the turnaround times of the approach as it runs on a range of computing nodes, from 1 to 32, across various data sizes. Therefore, the chart can also be seen as a comparison between sequential and parallel executions. In particular, Fig. 6(a) shows how the turnaround time decreases as the number of nodes increases, for different dataset sizes. For instance, for the D_{1M} dataset the turnaround time decreases from 699 s obtained on a single node, to 49 s on 32 nodes. For the D_{2M} dataset the turnaround time diminishes from 1663 s to 84 s. For the D_{4M} dataset the turnaround time decreases from 3577 s to 176 s. Finally, on the D_{8M} dataset, the turnaround time ranges from 7634 to 365 s using 32 nodes. Fig. 6(b) shows the relationship between turnaround time and dataset size, for a different number of servers. The graph shows that the time required to execute the entire approach increases proportionally with the increase of the input size. On the other side, the time required to execute the entire algorithm diminishes in proportion to the augmentation of computational resources.

Fig. 7 shows the execution speed-up and the efficiency of the approach as the number of computing nodes increases, versus different data sizes. In Fig. 7(a) it becomes evident that the speed-up is nearly linear with all datasets, up to the case of 8 nodes. For instance, on the D_{8M} dataset it achieves a value of 6.63. Then, for 16 and 32 nodes, it still maintains a notable trend, assessing on 11.97 and 20.88, respectively. Overall, these results show a reasonable scalability of the parallel approach till 32 nodes. Fig. 7(b) shows the application efficiency, vs the number of servers and for different number of instances. As shown in the figure, efficiency maintains a good trend and notable values till the case of 16 parallel nodes, in particular for large data volumes. For example, for the largest dataset the efficiency on 8 servers is equal to 0.83, whereas on 16 servers it is slightly below 0.8 (0.75). So, it means that the 83% and 75% of the computing power of each used server is exploited, respectively. For 32 nodes the efficiency is 0.65. Furthermore, it is worth noting that fixed the number of servers, the efficiency increases with the data size. This suggests that as the problem size increases, the distributed architecture becomes more convenient, which is an indicator of good scalability characteristics.

Fig. 8 shows the parallel execution time, partitioned for the amount required by each step of the approach, varying the data size and the number of servers. In particular, Fig. 8(a) shows the parallel execution time varying the number of data instances, achieved by exploiting 32 nodes. It is evident that the k-distance detection and DB-Scan execution steps take the majority of the total execution time. For example, for D_{8M} and 32 nodes, the k-distance detection and DB-Scan execution steps take around the 97% of the total execution time (25% and 72% respectively), and the other steps take only the remaining 3% of the total time. In particular, while the first one has a linear increasing trend with respect to data size, the second one shows a quadratic order increasing with respect to the number of instances (according to the temporal complexity of the DBScan algorithm). On the other side, Fig. 8(b) shows the parallel execution time for each step varying the number of nodes, for the eight-million instances case. The chart confirms that the turnaround time decreases with higher number of nodes, in particular due to the reduction of the parallel DB-Scan executions.

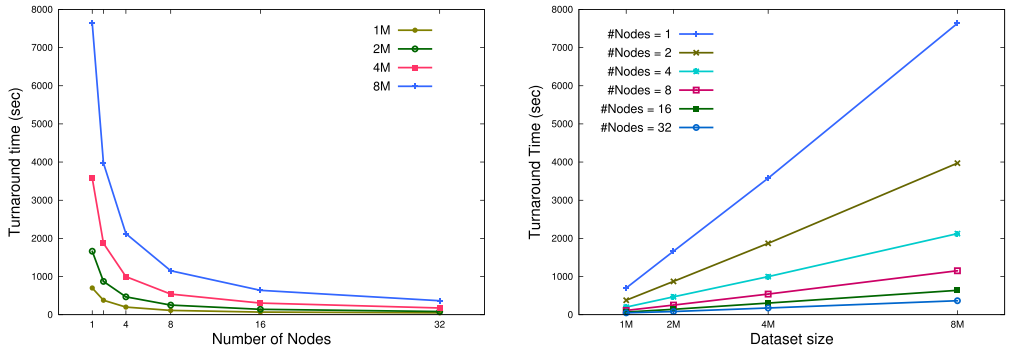
Fig. 9 shows overhead and turnaround times, for 16 and 32 nodes, versus increasing data sizes. We considered as overhead the time required by the approach to perform all the additional operations with respect to the sequential execution. This includes algorithmic operations (horizontal data splitting, clusters collection) as well as communication operations of each step of the workflow to execute. The overhead increases with the dataset size and the number of running nodes. However, we can observe that the overhead takes only a very small amount (almost negligible) of the total turnaround time, showing that the majority of the execution time is due to computation.

Fig. 10 compares the speedup retrieved by our experimentation with the speedup of two parallel density-based algorithms, namely, HPDBSCAN and PDSDBSCAN, as reported in the paper [36], on datasets of about 4M tuples. The Parallel CHD algorithm outperforms PDSDBSCAN by considering nodes 1 to 32, while, even if performances of HPDBSCAN seem to be better w.r.t. nodes 1 to 16, the results on 32 nodes are more favorable to the proposed parallel implementation of CHD. As a final remark, it is worth noting that, even if the three compared algorithms are all density-based, only the CHD can detect clusters each characterized by a different internal density.

6. Conclusion

This paper has presented a parallel multi-density clustering algorithm, to discover spatial hotspots from urban data. The experimental evaluation of the proposed approach has been performed on several synthetic datasets, showing good results in terms of execution time, speed-up and efficiency of the approach. Also, a case study based on the analysis of a real-world dataset (a collection of geo-referenced crime data of Chicago) is presented, with the aim to show a concrete scenario on which such an algorithm can be applied on.

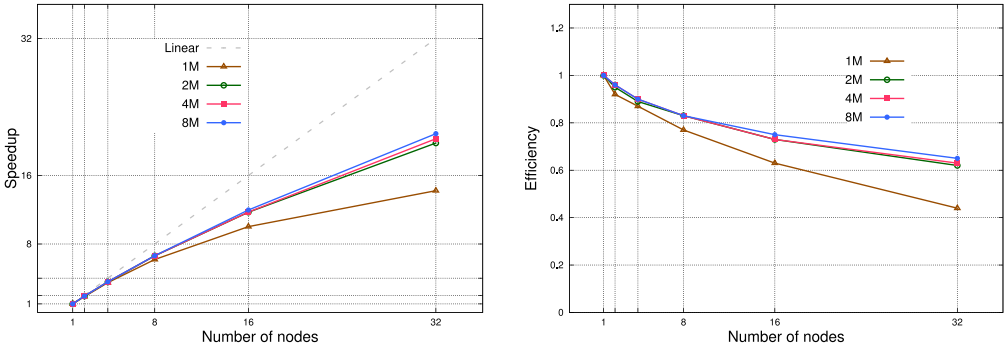
In future work, other research issues may be investigated. First, we plan to extend the approach to perform an automatic setting of the input parameter values, by executing a parameter sweeping procedure exploiting the running nodes in parallel. Second, we may further explore the algorithm application on other urban domains, i.e., mobility, pollution, micro-climate, etc., to discover multi-density hotspots for other kinds of events. Finally, we are interested in studying the algorithm scalability and efficiency on a large-scale Edge–Cloud architecture, in order to assess the performance on a real-time applicative domain.



(a) Execution time vs the number of available nodes, for different data sizes.

(b) Execution times vs data sizes, for different number of available nodes.

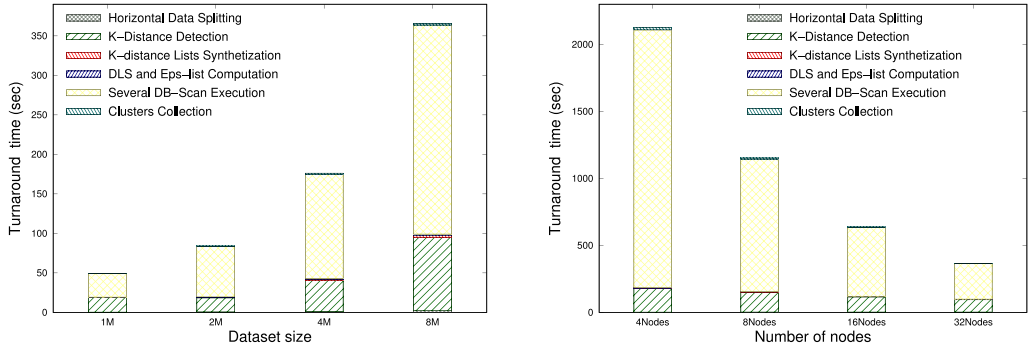
Fig. 6. Execution times in different scenarios.



(a) Speed-up

(b) Efficiency

Fig. 7. Speed-up and efficiency vs the number of available nodes, for different data sizes.



(a) Turnaround time for different dataset sizes, on 32 nodes.

(b) Turnaround time for different number of nodes, on the 8 million data size.

Fig. 8. Turnaround time (with the partial times required by each step) vs the number of available nodes (a) and different data sizes (b).

CRediT authorship contribution statement

Eugenio Cesario: Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology. **Paolo Lindia:** Writing – review & editing, Writing – original draft, Validation, Methodology, Conceptualization. **Andrea Vinci:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This research has been supported by the “PNRR MUR project PE0000013-FAIR” and the “ICSC National Centre for HPC, Big Data and Quantum Computing” (CN00000013) within the NextGenerationEU program. This work has also been partially supported by European Union - NextGenerationEU - National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) - Project: “SoBigData.it - Strengthening the Italian RI for Social Mining and Big Data Analytics” - Prot. IR0000013 - Avviso n. 3264 del 28/12/2021, and by the Italian

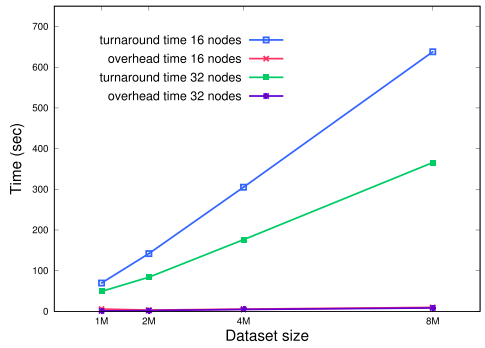


Fig. 9. T-Drive: Overhead time vs data sizes.

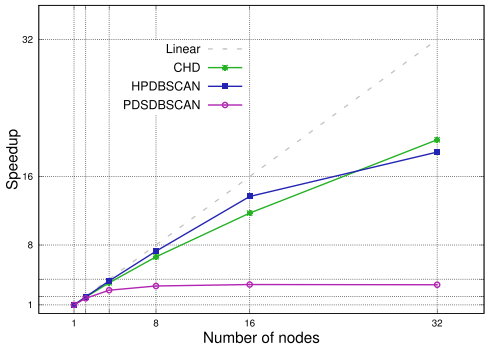


Fig. 10. Speedup Comparison between CHD, HPDBSCAN and PDSDBSCAN.

Ministry of University and Research, PRIN 2022 "INSIDER: INtelligent ServIce Deployment for advanced cloud-EdgeintegRation", grant n. 2022WWSCRR, CUP H53D23003670006.

References

[1] Guanxiong Liu, Hang Shi, Abbas Kiani, Abdallah Khreishah, Joyoung Lee, Nirwan Ansari, Chengjun Liu, Mustafa Mohammad Yousef, Smart traffic monitoring system using computer vision and edge computing, IEEE Trans. Intell. Transp. Syst. 23 (8) (2021) 12027–12038.

[2] Giuseppina Garofalo, Andrea Giordano, Patrizia Piro, Giandomenico Spezzano, Andrea Vinci, A distributed real-time approach for mitigating CSO and flooding in urban drainage systems, J. Netw. Comput. Appl. 78 (2017) 30–42.

[3] Grazia Belli, Andrea Giordano, Carlo Mastroianni, Daniele Menniti, Anna Pinnarelli, Luigi Scarcello, Nicola Sorrentino, Maria Stillo, A unified model for the optimal management of electrical and thermal equipment of a prosumer in a DR environment, IEEE Trans. Smart Grid 10 (2) (2019) 1791–1800.

[4] Juan Luis Pérez, Alberto Gutierrez-Torre, Josep Ll Berral, David Carrera, A resilient and distributed near real-time traffic forecasting application for Fog computing environments, Future Gener. Comput. Syst. 87 (2018) 198–212.

[5] Charith Perera, Yongrui Qin, Julio C. Estrella, Stephan Reiff-Marganiec, Athanasios V. Vasilakos, Fog computing for sustainable smart cities: A survey, ACM Comput. Surv. 50 (3) (2017) 1–43.

[6] Dixon Vimalajeewa, Chamil Kulatunga, Donagh P. Berry, Learning in the compressed data domain: Application to milk quality prediction, Inform. Sci. 459 (2018) 149–167.

[7] Albino Altomare, Eugenio Cesario, Andrea Vinci, Data analytics for energy-efficient clouds: design, implementation and evaluation, Int. J. Parallel Emergent Distrib. Syst. 34 (6) (2019) 690–705.

[8] Franco Cicirelli, Antonio Guerrieri, Giandomenico Spezzano, Andrea Vinci, Orazio Briante, Antonio Iera, Giuseppe Ruggeri, Edge computing and social internet of things for large-scale smart environments development, IEEE Internet Things J. 5 (4) (2017) 2557–2571.

[9] Marica Amadeo, Franco Cicirelli, Antonio Guerrieri, Giuseppe Ruggeri, Giandomenico Spezzano, Andrea Vinci, When edge intelligence meets cognitive buildings: The COGITO platform, Internet Things (2023) 100908.

[10] Peng Liu, Dong Zhou, Naijun Wu, VDBSCAN: varied density based spatial clustering of applications with noise, in: 2007 International Conference on Service Systems and Service Management, IEEE, 2007, pp. 1–4.

[11] Eugenio Cesario, Paolo Lindia, Andrea Vinci, Detecting multi-density urban hotspots in a smart city: Approaches, challenges and applications, Big Data Cogn. Comput. 7 (1) (2023) 29.

[12] Cities: The century of the city, Nature 467 (2010) 900–901.

[13] Franco Cicirelli, Antonio Guerrieri, Carlo Mastroianni, Giandomenico Spezzano, Andrea Vinci, The Internet of Things for Smart Urban Ecosystems, Springer, 2019.

[14] Mehdi Hosseinzadeh, Atefeh Hemmati, Amir Masoud Rahmani, Clustering for smart cities in the internet of things: a review, Cluster Comput. 25 (6) (2022) 4097–4127.

[15] Clara Pizzuti, Annalisa Socievole, Bastian Prasse, Piet Van Mieghem, Network-based prediction of COVID-19 epidemic spreading in Italy, Appl. Netw. Sci. 5 (2020) 1–22.

[16] Maria Pia Canino, Eugenio Cesario, Andrea Vinci, Shabnam Zarin, Epidemic forecasting based on mobility patterns: an approach and experimental evaluation on COVID-19 data, Soc. Netw. Anal. Min. 12 (1) (2022) 116.

[17] Eugenio Cesario, Paschal I. Uchubilo, Andrea Vinci, Xiaotian Zhu, Discovering multi-density urban hotspots in a smart city, in: IEEE International Conference on Smart Computing, SMARTCOMP 2020, Bologna, Italy, September 14-17, 2020, IEEE, 2020, pp. 332–337.

[18] Rosamunde Van Brakel, Paul De Hert, Policing, surveillance and law in a pre-crime society: Understanding the consequences of technology based strategies, Technol. Led Polic. 20 (2011) 165–192.

[19] Wim Hardyns, Anneleen Rummens, Predictive policing as a new tool for law enforcement? Recent developments and challenges, Eur. J. Crim. Policy Res. 24 (2018) 201–218.

[20] Eugenio Cesario, Paschal I. Uchubilo, Andrea Vinci, Xiaotian Zhu, Multi-density urban hotspots detection in smart cities: A data-driven approach and experiments, Pervasive Mob. Comput. 86 (2022).

[21] Eugenio Cesario, Big data analytics and smart cities: applications, challenges, and opportunities, Front. Big Data 6:1149402 (2023) 1–13.

[22] Safanaz Heidari, Mahmood Alborzi, Reza Radfar, Mohammad Ali Afsharkazemi, Ali Rajabzadeh Ghatari, Big data clustering with varied density based on MapReduce, J. Big Data 6 (1) (2019) 77.

[23] Madhuri Debnath, Praveen Kumar Tripathi, Ramez Elmasri, K-DBSCAN: Identifying spatial clusters with differing density levels, in: 2015 International Workshop on Data Mining with Industrial Applications, DMIA, IEEE, 2015, pp. 51–60.

[24] Charlie Catlett, Eugenio Cesario, Domenico Talia, Andrea Vinci, Spatio-temporal crime predictions in smart cities: A data-driven approach and experiments, Pervasive Mob. Comput. 53 (2019) 62–74.

[25] Eugenio Cesario, Domenico Talia, Distributed data mining models as services on the grid, in: Workshops Proceedings of the 8th IEEE International Conference on Data Mining, ICDM 2008, December 15-19, 2008, Pisa, Italy, IEEE Computer Society, 2008, pp. 486–495.

[26] Sushmita Mitra, Jay Nandy, KDDclus: A simple method for multi-density clustering, in: Proceedings of International Workshop on Soft Computing Applications and Knowledge Discovery, SCAKD 2011, Moscow, Russia, Citeseer, 2011, pp. 72–76.

[27] M.G. Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, Faraz Hussain, Machine learning at the network edge: A survey, ACM Comput. Surv. 54 (8) (2021) 1–37.

[28] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: Kdd, vol. 96, 1996, pp. 226–231.

[29] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, Jörg Sander, OPTICS: ordering points to identify the clustering structure, in: ACM Sigmod Record, vol. 28, ACM, 1999, pp. 49–60.

[30] Bi-Ru Dai, I-Chang Lin, Efficient map/reduce-based dbscan algorithm with optimized data partition, in: 2012 IEEE Fifth International Conference on Cloud Computing, IEEE, 2012, pp. 59–66.

[31] Hwanjun Song, Jae-Gil Lee, RP-DBSCAN: A superfast parallel DBSCAN algorithm based on random partitioning, in: Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 1173–1187.

[32] Yiqiu Wang, Yan Gu, Julian Shun, Theoretically-efficient and practical parallel DBSCAN, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 2555–2571.

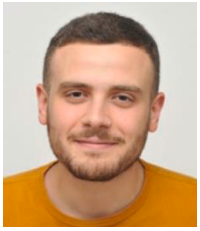
[33] Guoqing Wu, Liqiang Cao, Hongyun Tian, Wei Wang, HY-DBSCAN: A hybrid parallel DBSCAN clustering algorithm scalable on distributed-memory computers, J. Parallel Distrib. Comput. 168 (2022) 57–69.

[34] Md. Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, Alok N. Choudhary, A new scalable parallel DBSCAN algorithm using the disjoint-set data structure, in: Jeffrey K. Hollingsworth (Ed.), SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, IEEE/ACM, 2012, pp. 1–11.

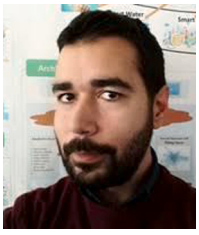
[35] Min Chen, Xuedong Gao, Huifei Li, Parallel DBSCAN with priority R-tree, in: 2010 2nd IEEE International Conference on Information Management and Engineering, 2010, pp. 508–511.

[36] Markus Götz, Christian Bodenstein, Morris Riedel, HPDBSCAN: highly parallel DBSCAN, in: Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, MLHPC 2015, ACM, 2015, pp. 1–10.
[37] Guangchun Luo, Xiaoyu Luo, Thomas Fairley Gooch, Ling Tian, Ke Qin, A parallel DBSCAN algorithm based on spark, in: Zhipeng Cai, Rafal A. Angryk, Wen-Zhan Song, Yingshu Li, Xiaojun Cao, Anu G. Bourgeois, Guangchun Luo, Liang Cheng, Bhaskar Krishnamachari (Eds.), 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom), BDCloud-SocialCom-SustainCom 2016, IEEE Computer Society, 2016, pp. 548–553.
[38] Yaobin He, Haoyu Tan, Wuman Luo, Shengzhong Feng, Jianping Fan, MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data, Front. Comput. Sci. 8 (2014) 83–99.
[39] Ziqing Wang, Zhirong Ye, Yuyang Du, Yi Mao, Yanying Liu, Ziling Wu, Jun Wang, AMD-DBSCAN: An adaptive multi-density DBSCAN for datasets of extremely variable density, in: 2022 IEEE 9th International Conference on Data Science and Advanced Analytics, DSAA, 2022, pp. 1–10.
[40] Eugenio Cesario, Andrea Vinci, Shabnam Zarin, Towards parallel multi-density clustering for urban hotspots detection, in: 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2021, IEEE, 2021, pp. 245–248.
[41] Yufang Zhang Zhongyang Xiong, Xuan Zhang, Multi-density DBSCAN algorithm based on density levels partitioning, J. Inf. Comput. Sci. 9 (10) (2012) 2739–2749.
[42] Antonio Congiusta, Domenico Talia, Paolo Trunfio, Parallel and grid-based data mining, in: Data Mining and Knowledge Discovery Handbook, Springer US, 2005, pp. 1017–1041.
[43] Mohammed J. Zaki, Parallel and distributed data mining: An introduction, in: Large-Scale Parallel Data Mining, Springer, 2002, pp. 1–23.
[44] Scikit-learn & joblib, 2023, https://ml.dask.org/joblib.html. (Accessed 12 December 2023).

Eugenio Cesario is an Associate Professor of Computer Engineering at University of Calabria (Italy). His research interests fall in the broad areas of Data Analytics and Parallel/Distributed Data Mining, and include Urban Computing, Smart Cities, Crime Data Mining, Energy-aware Cloud Computing, Cloud\Grid services architectures, Knowledge Discovery applications. He co-authored over seventy scientific papers in international journals, conference proceedings, and edited volumes. He is currently serving as member of the Editorial Board of two journals. He received two best paper awards and a best paper nomination in three international conferences. He has been serving as a chair, organizer, panelist and program committee member of several international conferences.



Paolo Lindia is a Ph.D. student in Information and Communication Technologies (ICT) at the Department of Computer Science, Modeling, Electronics and Systems Engineering (DIMES) of the University of Calabria. He received the M.Sc. degree in Data Science from Bicocca University, Italy. He is also a member of the Scalable Computing and Cloud Laboratory (SCALab) at DIMES, where he contributes to research and development projects.



Andrea Vinci received a Ph.D. in system engineering and computer science from the University of Calabria, Rende, Italy. He is a Researcher with ICAR-CNR, Rende, Italy, since 2012. His research mainly focuses on the Internet of Things and cyber–physical systems. He has authored or co-authored researches on the definitions of platforms and methodologies for the design and implementation of cyber–physical systems, and on distributed algorithms for the efficient control of urban and building infrastructures based on artificial and swarm intelligence.