
Article:
Zakhary, S., Rosser, J., Siebers, P.-O. et al. (2 more authors) (2021) Using unsupervised learning to partition 3D city scenes for distributed building energy microsimulation. Environment and Planning B: Urban Analytics and City Science, 48 (5). pp. 1198-1212. ISSN 2399-8083

https://doi.org/10.1177/2399808320914313

Using Unsupervised Learning to Partition 3D City Scenes for Distributed Building Energy Microsimulation

EPB: Urban Analytics and City Science
XX(X):2–24

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Prepared using sagej.cls

Abstract

Microsimulation is a class of *Urban Building Energy Modeling* techniques in which energetic interactions between buildings are explicitly resolved. Examples include *SUNtool* and *CitySim+*, both of which employ a sophisticated radiosity-based algorithm to solve for radiation exchange. The computational cost of this algorithm increases in proportion to the square of the number of surfaces of which an urban scene is comprised. To simulate large scenes, of the order of 10,000 to 1,000,000 surfaces, it is desirable to divide the scene to distribute the simulation task. However, this partitioning is not trivial as the energy-related interactions create uneven inter-dependencies between computing nodes. To this end, we describe in this paper two approaches (*K-means* and *Greedy Community Detection* algorithms) for partitioning urban scenes, and subsequently performing building energy microsimulation using *CitySim+* on a distributed memory High Performance Computing Cluster. To compare the performance of these partitioning techniques, we propose two measures evaluating the extent to which the obtained clusters exploit data locality. We show that our approach using Greedy Community Detection performs well in terms of exploiting data locality and reducing inter-dependencies among sub-scenes, but at the expense of a higher data preparation cost and algorithm run-time.

Keywords

hierarchical clustering, greedy community detection, urban scene, partitioning, scalability, building energy, microsimulation

Introduction

Sustainability is an important aspiration when planning the future of our urban settlements. The built environment contributes to a major proportion of greenhouse gas emissions ($\approx 55\,\%$) as a result of the considerable final energy used in this sector ($\approx 62\,\%$) (Anderson et al. 2015; Robinson 2012). To mitigate climate change and improve sustainability, there is a need to assess the environmental impact of the built environment and to test potential interventions as accurately as possible (*e.g.* retrofit strategies, heating/cooling systems and the use of renewable energy).

Urban Building Energy Modeling (UBEM) has emerged as a standard term (Reinhart and Davila 2016) for modeling the operational energy use of buildings. UBEM is particularly useful for testing strategies to reduce buildings' energy use and emissions in the urban context. A variety of strategies have been developed to model these energy flows, depending on the scale and objectives. These range from policy-oriented statistical models of city- (NOUVEL et al. 2015) regional- (Braulio-Gonzalo et al. 2016; Cheng and Steemers 2011) or national- (Sousa et al. 2018) scale aggregated stocks of buildings, through detailed dynamic modeling of building stocks represented by archetypes, to explicit microsimulation of buildings in their urban context for more detailed energy system (demand, storage, supply) planning and design (Robinson 2018). For further details, we refer interested readers to (Reinhart and Davila 2016; Robinson 2012).

In this paper, we are concerned with the microsimulation approach, using *CitySim+*. Despite the significant increase in computing power of single machines, microsimulation tasks at a neighborhood or a district scale tend to be beyond the capabilities of a single computing node. This is because the computational requirements for microsimulation grow significantly as the scene grows (*e.g.* to thousands of buildings, or hundreds of thousands of surfaces). We propose and evaluate a

methodology for splitting geospatial urban areas (we call them scenes hereafter) to facilitate urban scale building energy microsimulation for upwards of thousands of buildings. This methodology will enable modelers to partition the simulation problem in a way that can harness the capabilities of modern distributed computing platforms (*e.g.* cloud infrastructure (Kalms and Göhringer 2018; Assunção et al. 2015)) while minimizing the dependencies among the simulated parts to reduce overall simulation time.

CitySim+ (Zakhary et al. 2016) is a successor of CitySim (Robinson et al. 2009). In addition to extensions in its functional scope, it offers a newly designed data layer architecture that works directly with a standardized data model for 3D urban scenes (CityGML) (Kolbe et al. 2005) and the Energy Application Domain Extension (ADE) (Bruse et al. 2015; Rosser et al. 2019). This process is computationally intensive and the intensity depends on a number of factors. Some of these factors are: 1) the size of the scene (*i.e.* in number of buildings or number of surfaces), 2) the duration of the simulation (*i.e.* simulating a day or simulating a whole year), 3) the phenomena being simulated (*e.g.* full energy simulation, or solar irradiation simulation to study the potential for solar collectors). The algorithm of *CitySim+* has O(n^2) time complexity, where n is the number of surfaces in the scene.

Microsimulations of large geospatial scenes comprised of many thousands of buildings, while preserving simulation accuracy is an open research challenge (Frayssinet et al. 2018). We propose two novel approaches for scene partitioning using established unsupervised machine learning, namely *K-means* clustering and Greedy Community Detection (GCD). By partitioning scenes into smaller interacting (e.g. due to radiant energy couplings between surfaces) component parts (we call these sub-scenes), we can deploy a High Level Architecture (HLA) to distribute our

simulation tasks over a High Performance Computing (HPC) cluster. In this way, our partitioned sub-scenes may be simulated whilst accounting for inter-sub-scene energetic couplings between multiple instances of the *CitySim+* solver.

Our approach to partitioning scenes into inter-connected sub-scenes should respect the following requirements: 1) limit the size of the largest sub-scene to match the resource available on a single computing node, 2) balance the total number of clusters with the available number of computing nodes, 3) place a lower limit on any sub-scene to avoid wasting computing resources, 4) balance the distribution of workload, by minimizing the variance in sub-scene sizes. We apply these requirements to a scene of residential buildings in the *Sneinton* neighborhood of Nottingham, England. Details about the study area including some demographics, characteristics and a 3D map are presented in the supplementary material.

Through achieving these requirements, we exploit *data locality* in most of the phenomena simulated using *CitySim+*, leading to a reduction of the total computational cost (Baum and Winget 1990). The resulting smaller sub-scene can be run *concurrently* over a distributed memory HPC facility with minimal communication (*e.g.* simulation using HLA). We show the performance in-terms of cluster separation, simulation errors/reliability and overheads.

Related work

Many urban simulation problems face similar challenges as UBEM when tackled at scale. For example, performing microsimulation for the purpose of flood management using physically-based hydraulic model (*e.g.* shallow water equations) at the urban-scale at fine cell details is

generally not feasible to compute (Hunter et al. 2007) (*i.e.* requires both significant computing resources and long simulation time). Instead, different approaches have been proposed that simplify the model to only consider a 2D representation (Guidolin et al. 2016) in an attempt to provide rapid simulations result.

Unless the phenomenon of interest lends itself to multi-scale simulation approaches, in which fine resolution domains are nested within progressively coarser and larger simulation domains, as in the case of climate modeling (Robinson 2012), then the problem either needs to be simplified, or broken down and distributed. Examples of the former in the urban context include the use of image processing techniques to simulate shadow projections and sky view factors (Richens 1997) and combining these techniques with energy modeling for urban energy analysis (Ratti et al. 2000).

But in the present case, we wish to preserve the physical integrity of our urban simulations, so a distributed approach is of more interest. One such strategy is to partition large scenes into smaller sub-scenes while minimizing the interactions between them. For example, Sanjurjo et al. (2013) proposed a convex partitioning approach to calculate the *global illumination* over a large scene using a parallel Monte Carlo ray tracing algorithm. The approach reduced the computing overhead by minimizing the number of rays crossing from one processor to another over a distributed memory architecture. This approach is statistically driven and does not consider reduction of communication overhead at the level of the integration platform (*e.g.* HLA). Other approaches for managing large scenes are based on limiting processing overheads in the case of radiosity calculations, for example, 1. via visibility analysis (Cohen-Or et al. 2003) 2. using an importance-based approach to speed-up radiosity calculation (Aguerre et al. 2019) 3. by superimposing a regular

grid (Muñoz et al. 2018; Rodríguez et al. 2017). With the regular grid approach, each grid cell is simulated independently with some boundary overlap with surrounding cells. A regular grid-based splitting of the urban scene would not be suitable in our case as cells will contain different number of buildings (hence unbalanced computing load and memory requirements).

Many researchers have used *image segmentation* for partitioning view-space, for example (Lowe 1999; Kumar et al. 2015; Dhanachandra et al. 2015). Image segmentation is a low-level image processing problem that deals with the challenge of automatically classifying an image into different regions. It has mostly been employed for high-level computer vision operations, ranging from object recognition (Lowe 1999) to scene understanding (Kumar et al. 2015). Clustering has been used to perform *image segmentation* by dividing an image into discrete regions. K-means is one of the most popular algorithms for clustering due to its simplicity and performance (Dhanachandra et al. 2015). It divides a given scene into a disjoint set of smaller sub-scenes where there is high similarity within and high differences between the sub-scenes according to a defined distance function.

More recently, a greedy optimization approach for clustering based on *modularity* was proposed (Clauset et al. 2004). This approach belongs to a family of algorithms known as *agglomerative hierarchical clustering*. This is an unsupervised machine learning approach that performs a bottom-up clustering (Fahad et al. 2014). The process starts with each object belonging to its own cluster, with adjacent clusters being progressively merged, as defined by a given hierarchy. Clauset et al. (2004) proposed a fast hierarchical agglomeration algorithm for finding various communities in a large graph (we call it GCD hereafter).

Method: Machine Learning for Scene Splitting

In this section we present our approach for splitting urban scenes. We start by introducing our proposed means for evaluating scene partitioning efficiency. We then introduce a base-line using K-means for comparative purposes. Finally, we detail our proposed approach using GCD and present its resulting sub-scenes.

Measures of Efficiency

To measure the efficiency of the splitting algorithm, we analyze the relationship between the sub-scenes to show the strength of their interdependence. We developed two measures to better understand how any two sub-scene are linked. Firstly, we measure the relative number of edges (*Links(%)*) that exist between each pair of sub-scenes (*i.e.* inter-sub-scene links on the graph) to the *mean* number of links intra-sub-scene. Secondly, we measure the total relative Inter-Reflection (IR) of these edges (*IR(%)*) representing the total weight given to these edges relative to the *mean* of the total IR on all links inter-sub-scenes.

The first measure provides direct information about how many buildings will need to be reprocessed at each synchronization point at the various computing nodes. The second measure provides information about the impact of the contributions from one sub-scene to another. As it is the case with IR, this impact is not symmetrical (*i.e.* the sub-scene that is receiving the inter-reflected radiation is being impacted and not the one reflecting it). Hence, we analyze the measures of inter-sub-scene edges as directed graph that show radiation from the reflecting to the receiving sub-scene (called *Emitting* to the *Receiving* sub-scene).

Base-line Case Using K-means Clustering

K-means is one of the widely used unsupervised machine learning algorithms (Jain 2010). We show in this section the application of K-means in clustering the centroids of buildings' footprints. In comparison to GCD, K-means has the following important advantages: 1) It performs fast clustering, 2) the way it was set-up in our case requires little pre-processing to obtain the buildings' centroids (*i.e.* it doesn't require pre-processing of the whole urban scene to calculate the IR relationship graph between buildings), 3) and it assumes spherically shaped clusters. This is particularly useful when partitioning a large scene of tens of thousands of building, or indeed a whole city.

K-means Algorithm for splitting large scenes

The K-means algorithm aims to find a set of clusters consisting of groups of buildings based on the *Euclidean distance* between their footprints. The hypothesis is that closer buildings will have higher IRs, hence stronger interactions compared to buildings that are further away. This leads to clusters that have more edges within each cluster than between clusters.

A required *Hyperparameter* for the K-means algorithm is the number of clusters to represent the data. Research has been conducted to find a suitable value (Jain 2010). The prefix "*hyper*" indicates that this is a required input set by the user for the algorithm, as opposed to a parameter which can be deduced through the training process. For operational purposes, we choose the maximum number of clusters empirically based on the available resources to each of the computing nodes as this will dictate how large a sub-scene can be. It is important to note that is the minimum number of sub-scenes (and hence the lowest number of HPC computing nodes) needed to simulate the whole scene distributively.

We run K-means using Lloyd's algorithm (Lloyd 1982) (also known as Voronoi iteration) to search for evenly spaced and disjoint sets of clusters. The partitioning of points is based on the Euclidean distance. Our criteria for the analysis is to minimize the *inertia* which is the sum-of-squares within each of the clusters.We set the number of clusters to the required number of computing nodes (as discussed above), and the algorithm divides the N building centroid samples BC_i into K disjoint clusters C. For each cluster C_i, the algorithm calculates the center CC_i and each point is assigned to the cluster closest to it. The mean for all the samples within each cluster μ_i is recalculated to minimize inertia. This is repeated until the results stabilize (*i.e.* both C_i and CC_i do not change significantly). Although CC_i falls within the same space as BC, it does not have to be one of these points. The formal definition for the minimization of inertia can be described as follows:

$$\frac{1}{n}\sum_{j=1}^{K}\sum_{x_i \in C_j}\left(|x_i - \mu_j|^2\right) \tag{1}$$

where x_i denotes a point $i \in [1..N]$ which belongs to cluster C_j, and μ_j is the current mean for cluster j.

K-means scales well with large datasets compared to other algorithms (Arthur and Vassilvitskii 2006). One particular problem with K-means is that it can get locked in local minima. In order to avoid this, we performed multiple initial sampling of different cluster centers. We subsequently choose the set of cluster centers with the minimum inertia as the initial set of cluster centers for the K-means algorithm.

K-means Algorithm Results in a Realistic 3D Urban Scene

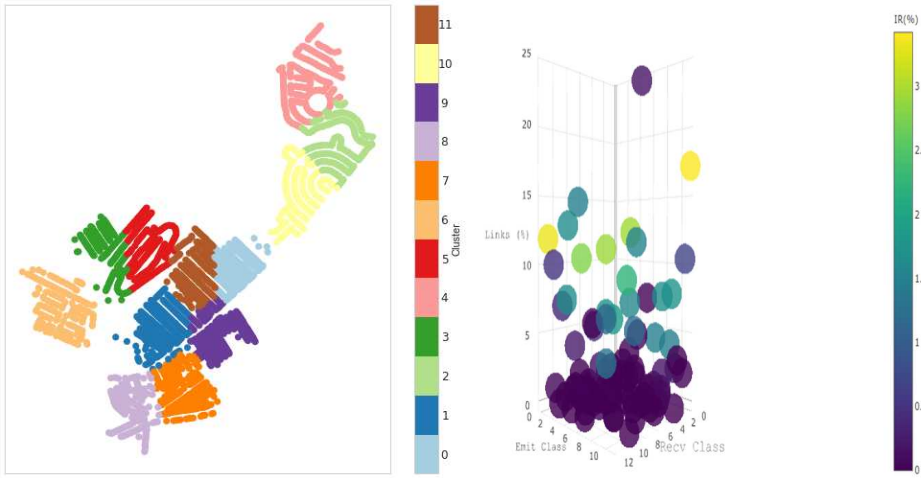
In this section, we present our base-line approach. We show the results of splitting a large urban scene (shown in figure 1 in the supplementary

material) into a number of smaller sub-scenes. We perform the clustering using K-means as discussed above to label each of the buildings with one of the cluster labels. We performed each run using 10 randomly assigned centers for each of the clusters shown in figure 1a.

One way to fulfill the dependency between the sub-scenes is by performing regular synchronization and exchange of intermediate results throughout the parallel execution over K computing nodes (*e.g.* using the HLA standard). This exchange preserves the energy-relations between sub-scenes and ensures the accuracy of simulation results. Another approach is to replicate buildings affecting each sub-scene during the simulation in order to account for their effect without performing further synchronization during execution. Since our HLA-enabled simulation platform development (Amponsah et al. 2019) is not sufficiently complete for the purposes of demonstration at this time, we adopt the latter approach for the evaluation in this paper.

Figure 1a show the labels assigned using the K-means algorithm to each of the buildings in our example scene (shown in figure 1 in the supplementary material). The figure shows 12 different clusters to partition the scene which could be easily changed. Some of these clusters are clearly separated by distinctly clear *Euclidean* distance such as clusters 3 and 6 on the *west* part of the scene. On the other hand, the algorithm did not make a clear distinction between some other clusters (*e.g.* clusters $2, 4$ and 10 at the *north* and towards the *center* as well). We can see that the K-means algorithm do not produce clear separate classes in difficult cases where buildings are densely positioned and with little gap existing between them.

The distribution of load across clusters is also important as it will affect the performance of the whole simulation (*i.e.* nodes reaching synchronization points will have to wait for the slowest node before



(a) Produced clusters.

(b) Clustering performance (class refers to a sub-scene)

Figure 1. K-means results for the Sneinton scene (shown in figure 1 in the supplementary material).

they all exchange data -if needed- and proceed). The simulation speed is directly linked to the number of buildings (more specifically the total number of surfaces) that are simulated by *CitySim+*, and equally sized clusters will ensure that all nodes proceed at a comparable execution speed. We have found that the number of buildings in each of the clusters does not vary significantly (*i.e.* clusters are of relatively similar size), hence processing time is expected to be homogeneous (apart from local node-specific variations).

Figure 1b shows the measurements of performance for the K-means algorithm. The figure shows the *Links(%)* and *IR(%)* between each pair of clusters. *Emit* refers to the links where the irradiation is emitted from the sub-scene, and *Recv* refers to the links where the irradiation is received by the sub-scene. We can see that the maximum *Links(%)* reaches 25%. This indicates that the two clusters have about quarter of the links between them compared to the mean number of links inside any of the clusters. At

the same time, this maximum value for *Links(%)* is associated with a low *IR(%)* (close to $0.5\,\%$), which means that, although there are many links between these two clusters, the amount of data to be exchanged expressed by *IR(%)* is quite low. It is interesting to observe that for the values where *IR(%)* is higher than $1.37\,\%$ (or mean + 1 Std. Dev.), the *Links(%)* ranges from $4\,\%$ to $18\,\%$.

To further analyze the results obtained using K-means and GCD, we have performed statistical analysis of various variables (including *Links(%)* and *IR(%)*) and we provide details in the supplementary material.

Greedy Community Detection Algorithm

We adopt the Greedy Community Detection algorithm by Clauset et al. (2004) for the discovery of community structure in large network. In our case, the buildings form a network where they are linked with each other by the IR. This section presents the results obtained using the GCD algorithm. This machine learning algorithm, similar to K-means, is unsupervised. Moreover, it is a hierarchical agglomeration algorithm, hence has a significant overhead compared to K-means. We present the results similar to those shown for K-means in previous section for ease of comparison.

GCD Algorithm for splitting large scenes

We define a community as a collection of buildings that have stronger and a higher number of links between them compared to those with other communities. Hence, we require a graph that depicts such IR interconnections between the buildings (which we have illustrated in figure 4) in order to preform the community detection. The algorithm starts with each building belonging to its own community. During the

agglomerative process, each of these communities is iteratively merged with another community whose amalgamation produces the highest increase in the graph modularity *Q* which is defined as follows:

$$Q = \sum_i e_{ii} - a_i^2 \tag{2}$$

where generally e_{ij} is the fraction of edges that link vertices from community i to vertices in community j, and e_{ii} is the fractions of edges that starts and ends with vertices in community i. a_i represents the fraction of the ends of edges that are linked to vertices in community i.

Unlike K-means, GCD does not require an input parameter setting the number of clusters. On the other hand, the algorithm uses a set of weights associated with the graph edges (*i.e.* community detection using a weighted graph). A GCD algorithm starts by setting each building to its own cluster to form *N* clusters (where *N* refers to the number of buildings). The algorithm does not require a target number of clusters as the agglomeration process is done iteratively to test for a possible merge between clusters (*e.g.* test which two closely linked clusters could be merged together into as a single cluster). This merging process is repeated for *N-1* iterations. Once the process is completed, the resulting number of communities could be anything from 1 (where all the buildings are tightly linked) to *N* (no two buildings are interconnected). More often than not, we have observed that GCD clustering produces more communities than needed to perform distributed UBEM for the whole scene as the number of computing nodes $\ll N$.

In order to reduce the number of cluster (*i.e.* to a number equal to the processing nodes available for the distributed UBEM) while observing the boundary between the detected communities using GCD, we have introduced an edge contraction stage after running the initial GCD. We

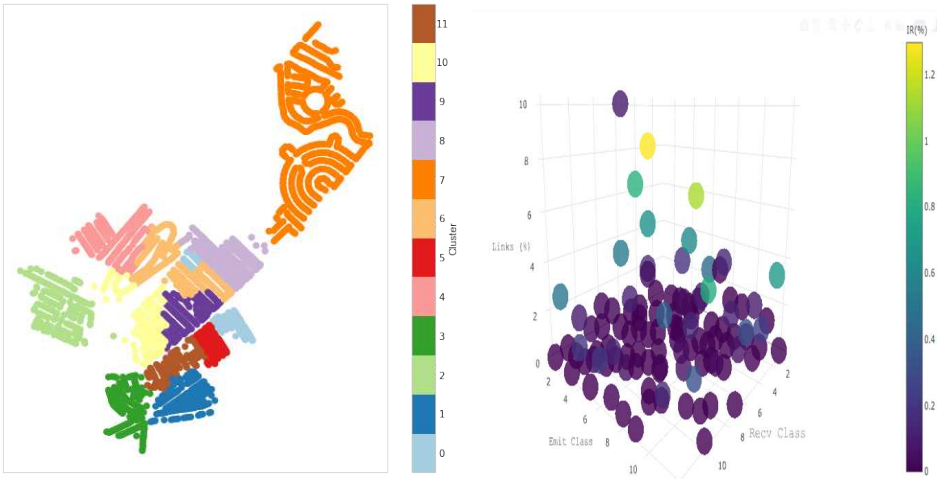
note that merging two communities produced by GCD would -as a worst case- have the effect of adding their interdependencies when running the UBEM (*i.e.* the new formed community will have a dependency that is the union of the interdependency of its constitutions). For this step, we require a community-level weighted graph that describes community-to-community links with the weight representing the total original IR relation between the two communities. The community-level graph is similar to the original weighted graph we have used to run GCD but with all the edges that fully exist within the same community being contracted (*i.e.* edges where the two buildings belong to the same community). *Edge contraction* is a graph theory operation by which two vertices can be merged into one while preserving the original graph connectivity. After performing this post-processing step, we have a graph with each vertex representing one of the detected communities by GCD and where the number of vertices is still higher than the available processing nodes. To reduce the number of communities further, we iteratively perform edge contraction on the community-level graph merging strongly linked communities first until we reach the desired number of communities.

GCD Algorithm Results in a Realistic 3D Urban Scene

Figure 2a shows the sub-scenes obtained using the GCD algorithm. The results show the assigned clusters over the scene (shown in figure 1 in the supplementary material). The resulting sub-scenes are clearly different from those obtained using K-means and shown in figure 1a. This is especially clear at the central part of the scene (*e.g.* around clusters 6, 8, 9 and 10).

These clusters at the middle of the scene are difficult to separate using machine learning algorithms for two reasons. Firstly, the buildings are compacted and there are no clear separation lines between these clusters. Secondly, the terrain (*i.e.* affected by the ground height) features a valley

at the center (*i.e.* around clusters 6, 9 and 10) (as it can be seen in figure 1 in the supplementary material).



(a) Produced clusters.

(b) Clustering performance (class refers to a sub-scene).

Figure 2. GCD results for the Sneinton scene (shown in figure 1 in the supplementary material).

Figure 2b shows the measurements of performance for the *GCD* algorithm. Similar to figure 1b, it shows the *Links(%)* and *IR(%)* between each pair of clusters. The maximum *Links(%)* is much lower and reaching only $10\,\%$. This indicates that these two clusters have about only $10\,\%$ of the links between them compared to the mean number of links inside any of the clusters. This maximum value for *Links(%)* is associated with a low *IR(%)* (close to $0.5\,\%$), but the next highest value for *Links(%)* has a much higher *IR(%)* than the mean of about $1.3\,\%$. Even for this instance with higher *IR(%)*, it is much lower than when using K-means which goes up to $3.5\,\%$. It is interesting that compared to K-means, for the values where *IR(%)* is higher than $0.36\,\%$ (or mean + 1 Std. Dev.), we find the *Links(%)* ranges from around $2\,\%$-$8\,\%$ (compared to the range of $4\,\%$-$18\,\%$ for K-means).

UBEM Simulation Results

In this section we demonstrate the performance of the two scene splitting approaches using *CitySim+*. We performed two sets of experiments to compare the impact of considering buildings outside each sub-scene that contribute to the simulation errors. Firstly, we executed *CitySim+* simulations only considering buildings that exist in each sub-scene (*i.e.* without sharing any data concerning buildings located beyond them). Secondly, we execute simulations in which each sub-scene includes all external buildings (to this sub-scene) that affects the *CitySim+* simulation due to the existence of *links*. This produces a modified set of sub-scenes that are larger than the original clustering results. As noted earlier, this is an intermediate step for the purposes of evaluating our scene partitioning strategies, which will be implemented in the future as part of a progressively comprehensive urban simulation platform, employing HLA.

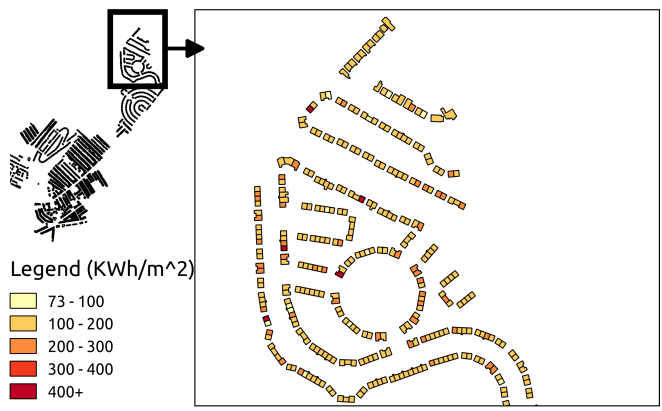


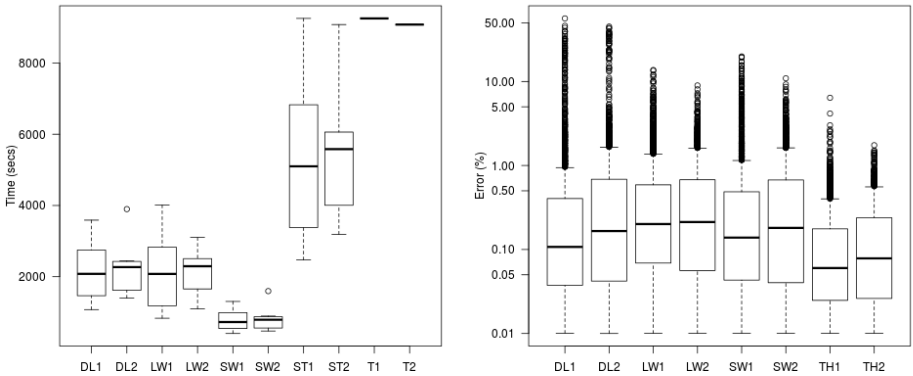
Figure 3. Exploded view (right) of annual heating energy demand simulation results obtained following simultaneous microsimulation of the whole scene (left): *i.e.* no splitting.

To calculate the errors introduced by the two approaches, we have used a "*truth*" case in which we have performed a full microsimulation of the

whole scene, without splitting, on a single computing node (so that the scene size was determined based on the capacity of this node; a restriction that is avoided in the distributed case that is the motivation underlying this work). Figure 3 presents the yearly energy demand per floor area for a small part of the whole scene (using average story height of $2.7\,\mathrm{m}$ and the whole building footprint area).

Figure 4a compares the execution times of the sub-scenes simulations produced by K-means and GCD. In this and subsequent figures, we denote K-means by 1 and GCD by 2. Each sub-scene is simulated without considering any interactions existing outside the boundary of each sub-scene (*i.e.* not considering inter-sub-scenes links). We can see that the median execution time for the three components (Daylight, Longwave and Shortwave: the calculation of which each use the same sub-scene view information) and the total are slightly higher for GCD sub-scenes compared to the K-means case. On the other hand, we find that the K-means sub-scenes show greater variability (*i.e.* the degree of imbalance is much higher). In a distributed simulation scenario, this will lead to slower overall execution, as the slowest *federate* will in turn delay other federates, as they get held at each synchronization point. We note particularly that the upper quartile is systematically lower for GCD. The total simulation time -which is the time when the slowest sub-scenes finishes- shows that GCD-based simulation is slightly faster than K-means.

Figure 4b shows the errors in simulation results when not considering external buildings. The K-means and GCD sub-scenes results are compared to the base case of running the scene as a whole (*i.e.* without partitioning). This base case was constructed for comparison purposes but it would not be possible for larger scenes (as it would not be possible to run it as a single simulation). In this figure, we omit buildings that have very small simulation errors (*i.e.* error value below $0.01\,\%$) for clarity of



(a) Execution times **(b)** Simulation errors (y-axis in log scale)

Figure 4. Performance of *CitySim+* components (no external buildings) for K-means(1) & GCD(2): DL, LW, SW, sub-scene total(ST)/Total(T) respectively.

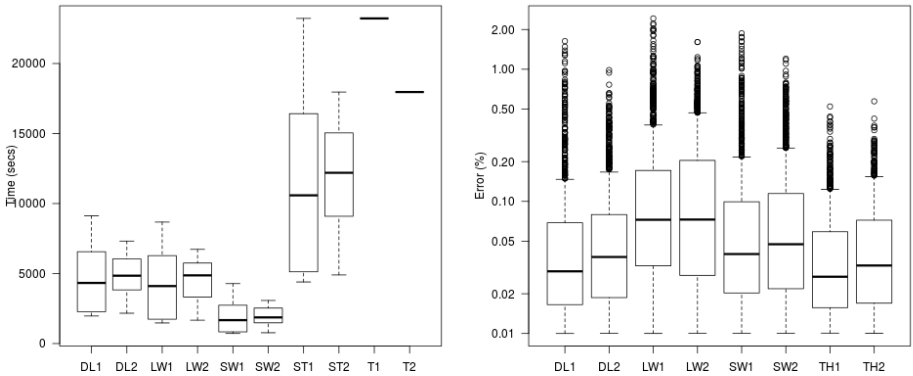
presentation. Firstly, we note that for all the components the maximum errors (without considering the outliers as defined by the box-plot graph) were around $2\,\%$. These results are attributed to the fact that most buildings are not on the boundary of the sub-scenes, hence most of their IR occurred with other buildings that already existed within the same sub-scene.

Secondly, we note that for most of the simulation components shown in Fig. 4b, there are many observations where the errors, marked as outliers or circles, reached as high as $50\,\%$. These significant errors could be attributed to buildings that are affected by the missing IR interactions due to the lack of the influence of external buildings in each sub-scene. The results show median errors, for all the components, that are close to $0.10\,\%$, with the 3rd quartile being below $0.50\,\%$, which means that for most buildings, their interactions were contained within their respective sub-scenes. We also note that the range of error is similar for K-means and GCD sub-scenes. For each of the sub-scenes, only a small percentage of buildings would be affected around the boundary where most of the missing IR interactions from external buildings were present.

We now demonstrate the partitioning algorithms by considering the influence of external buildings. For this, we augment each of the sub-scenes by including external buildings from surrounding sub-scenes before performing our microsimulations. We select which buildings needs to be considered in each of the sub-scenes based on the IR *links*. We show the impact of the two different approaches on execution time and errors (noting that these errors should be minimized or eliminated when using a distributed simulation platform employing HLA).

Figure 5a shows the execution times of the various simulations of all sub-scenes produced by both K-means and GCD. Although the median execution time for K-means sub-scenes is slightly lower, they still show a much larger variance and range. This is not only the case for each sub-scene total, but also at the individual components simulation time. This means that when performing synchronization at the component level, we should expect even higher impact on performance (*i.e.* slower simulation). Moreover, it is important to highlight that the increase in median cost is about a factor of two (compared to those shown in fig. 4a), underlining the importance of strategies to minimize interconnections. We could also see that the total simulation time in this case is significantly lower for GCD.

Figure 5b shows the errors in simulation results when considering the effect of external buildings affecting each of the sub-scenes. Overall, the errors are much lower compared to those shown in Fig. 4b as the median is about $0.05\,\%$ and the third quartiles are below $0.12\,\%$. Meanwhile, we can still observer slightly higher errors of below $2\,\%$ which could be attributed to how external buildings were considered in each sub-scene (*i.e.* cascade impact effect due to considering the external buildings outside of their spatial context). From these results, we can see considerable increase in accuracy but with an additional increase in execution time. The simulation errors vary slightly between K-means and GCD sub-scenes. This is due



(a) Execution times **(b)** Simulation errors (y-axis in log scale)

Figure 5. Performance of *CitySim+* components (with external buildings) for K-means(1) & GCD(2): DL, LW, SW, sub-scene total(ST)/Total(T) respectively.

to the fact that external buildings—which contribute to these errors if missed—were included to supplement each of the sub-scenes with either approaches, hence both were rendered similar in terms of their simulation accuracy.

Conclusion

In this paper, we presented the problem of splitting large-scale UBEM simulations. We proposed an approach based on GCD, a graph partitioning machine learning algorithm, to automate the splitting process. For this, we developed a graph-based representation of the scene to minimize energy flows which subsequently reduces computation dependency when using distributed computing. We compared the performance to a baseline approach based on using the K-means algorithm to cluster buildings' centroids over the 2D map utilizing the Euclidean distance. We have proposed two measures to evaluate the effectiveness of our splitting algorithm (Links(%) and IR(%)).

From this we conclude the following: 1) A spatial-only partitioning using *K-means* -although having minimal overhead in-terms of data preparation compared to GCD- produces uneven sub-scenes, 2) Partitioning the IR graph of a large scene using GCD reduces the energy-related interactions which subsequently evenly distribute the inter-dependencies among computing nodes, 3) The total runtime required for simulating the sub-scenes independently when using GCD approach is lower compared to *K-means*. We expect to report on the utility of this distributed simulation architecture for urban building energy microsimulation (and other complementary) purposes in the near future.

References

Aguerre JP, Fernández E and Beckers B (2019) Importance-driven approach for reducing urban radiative exchange computations. *Building Simulation* 12(2): 231–246.

Amponsah K, Zakhary S, Robinson D, Logan B, Nathanail P and Siebers PO (2019) Distributed building energy simulation with the HLA.

Anderson JE, Wulfhorst G and Lang W (2015) Energy analysis of the built environment—a review and outlook. *Renewable and Sustainable Energy Reviews* 44(0): 149 – 158.

Arthur D and Vassilvitskii S (2006) How slow is the k-means method? In: *Proceedings of the twenty-second annual symposium on computational geometry*. ACM,

Assunção MD, Calheiros RN, Bianchi S, Netto MA and Buyya R (2015) Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*

Baum DR and Winget JM (1990) Real time radiosity through parallel processing and hardware acceleration. In: *ACM SIGGRAPH Computer Graphics*, volume 24. ACM,

Braulio-Gonzalo M, Juan P, Bovea MD and Ruá MJ (2016) Modelling energy efficiency performance of residential building stocks based on bayesian statistical inference. *Environmental Modelling & Software* 83: 198 – 211.

Bruse M, Nouvel R, Wate P, Kraut V and Coors V (2015) An energy-related CityGML ADE and its application for heating demand calculation. *International Journal of 3-D Information Modeling* 4(3): 59–77.

Cheng V and Steemers K (2011) Modelling domestic energy consumption at district scale: A tool to support national and local energy policies. *Environmental Modelling & Software* 26(10):

1186 – 1198.

Clauset A, Newman ME and Moore C (2004) Finding community structure in very large networks. *Physical review E*

Cohen-Or D, Chrysanthou YL, Silva CT and Durand F (2003) A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*

Dhanachandra N, Manglem K and Chanu YJ (2015) Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*

Fahad A, Alshatri N, Tari Z, Alamri A, Khalil I, Zomaya AY, Foufou S and Bouras A (2014) A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on emerging topics in computing*

Frayssinet L, Merlier L, Kuznik F, Hubert JL, Milliez M and Roux JJ (2018) Modeling the heating and cooling energy demand of urban buildings at city scale. *Renewable and Sustainable Energy Reviews*

Guidolin M, Chen AS, Ghimire B, Keedwell EC, Djordjević S and Savić DA (2016) A weighted cellular automata 2d inundation model for rapid flood analysis. *Environmental Modelling & Software* 84: 378 – 394.

Hunter NM, Bates PD, Horritt MS and Wilson MD (2007) Simple spatially-distributed models for predicting flood inundation: A review. *Geomorphology* 90(3): 208 – 225.

Jain AK (2010) Data clustering: 50 years beyond k-means. *Pattern recognition letters*

Kalms L and Göhringer D (2018) Scalable clustering and mapping algorithm for application distribution on heterogeneous and irregular FPGA clusters. *Journal of Parallel and Distributed Computing*

Kolbe TH, Gröger G and Plümer L (2005) Citygml: Interoperable access to 3d city models. In: *Geo-information for disaster management*. Springer,

Kumar MP, Turki H, Preston D and Koller D (2015) Parameter estimation and energy minimization for region-based semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence*

Lloyd S (1982) Least squares quantization in PCM. *IEEE transactions on information theory*

Lowe DG (1999) Object recognition from local scale-invariant features. In: *The proceedings of the seventh IEEE international conference on Computer vision*, volume 2.

Muñoz D, Beckers B, Besuievsky G and Patow G (2018) A technique for massive sky view factor calculations in large cities. *International Journal of Remote Sensing* 39(12): 4040–4058.

NOUVEL R, BRASSEL KH, BRUSE M, DUMINIL E, COORS V, EICKER U and ROBINSON D (2015) SimStadt, a new workflow-driven urban energy simulation platform for CityGML

city models. In: *Proceedings of International Conference CISBAT*. LESO-PB, EPFL,

Ratti C, Robinson D, Baker N and Steemers K (2000) LT urban-the energy modeling of urban form.

Reinhart CF and Davila CC (2016) Urban building energy modeling – a review of a nascent field. *Building and Environment* 97: 196 – 202.

Richens P (1997) Image processing for urban scale environmental modelling. In: *Proceedings of the Int. Conf. on Building Simulation'97*.

Robinson D (ed.) (2012) *Computer Modelling for Sustainable Urban Design: Physical Principles, Methods and Applications*. Taylor & Francis.

Robinson D (2018) Integrated resource flow modelling of the urban built environment. In: Hensen J and Lamberts R (eds.) *Building Performance Simulation for Design and Operation*, 2st edition. Taylor & Francis,

Robinson D, Haldi F, Kämpf J, Leroux P, Perez D, Rasheed A and Wilke U (2009) Citysim: Comprehensive micro-simulation of resource flows for sustainable urban planning. In: *Proc. Building Simulation*. Eleventh International IBPSA Conference,

Rodríguez LR, Nouvel R, Duminil E and Eicker U (2017) Setting intelligent city tiling strategies for urban shading simulations. *Solar Energy*

Rosser JF, Long G, Zakhary S, Boyd DS, Mao Y and Robinson D (2019) Modelling urban housing stocks for building energy simulation using citygml energyade. *ISPRS International Journal of Geo-Information* 8(4).

Sanjurjo JR, Amor M, Bóo M and Doallo R (2013) Parallel Monte Carlo radiosity using scene partitioning. *The International Journal of High Performance Computing Applications* 27(3): 318–334.

Sousa G, Jones BM, Mirzaei PA and Robinson D (2018) An open-source simulation platform to support the formulation of housing stock decarbonisation strategies. *Energy and Buildings*

Zakhary S, Andrew A, Siebers P and Robinson D (2016) A computational workflow for urban micro-simulation of buildings' energy performance.

Aguerre JP, Fernandez E, Besuievsky G and Beckers B (2017) Computing urban radiation: A sparse matrix approach. *Graphical Models*

Cohen MF, Greenberg DP, Immel DS and Brock PJ (1986) An efficient radiosity approach for realistic image synthesis. *IEEE Computer graphics and Applications*

Robinson D and Stone A (2004) Solar radiation modelling in the urban context. *Solar Energy* 77(3): 295 – 309.

Wilson RJ (1979) *Introduction to graph theory*.