XCrowd: Real-Time Dynamic Crowd Movement Simulation on Graph Networks

Jan Appel
Institute of Computer Science
Winterthur, Switzerland
appeljan@students.zhaw.ch

Andreas Weiler
Institute of Computer Science
Winterthur, Switzerland
andreas.weiler@zhaw.ch

ABSTRACT

We introduce *XCrowd*, an application that simulates human crowd movements in large areas using graph networks. Although other theoretical and simulation models exist, they fail to simultaneously account for dynamic crowd movements, realistic visitor behavior, and real-world disturbances. Our solution aims to bridge this gap between theoretical amusement park optimization and real-world operations. By representing park layouts as graph networks, XCrowd is able to efficiently and realistically provide tools to evaluate crowd behavior under specific conditions and, for example, help validate the efficiency of a given park layout or simulate a certain event. We discuss the methodology, implementation, evaluation and validation of our model, as well as potential implications for amusement park management and outline future directions, including scalability, limitations, user experience, and address real-world dynamic factors.

VLDB Workshop Reference Format:
Jan Appel and Andreas Weiler. XCrowd: Real-Time Dynamic Crowd Movement Simulation
on Graph Networks. VLDB 2024 Workshop: 3rd International Workshop on Large-Scale Graph Data Analytics (LSGDA 2024).

1 INTRODUCTION AND MOTIVATION

Amusement parks, hosting millions of visitors annually, have become key destinations for leisure and entertainment [8]. As with any large event, such high visitor numbers require great efforts and precision during planning in order to provide a smooth and safe experience for all guests. Changes in ride availability or weather conditions may lead to guests moving to different parts of the amusement park or seeking shelter from the rain. Around the opening and closing hours of an amusement park, some paths might also experience a drastically higher load than during daytime operation. Most importantly, however, in case of an emergency, such as a fire or a medical incident, the park management must be able to respond quickly and ensure that emergency responders reach the location in time as well as direct nearby guests away from the affected area.

Although existing theoretical and simulation visitor models do exist, they may fall short when simultaneously accounting for dynamic crowd movements, realistic visitor behavior and real-world disturbances within amusement parks. These factors are often overlooked in theoretical models, which may result in noticeable inaccuracies when predicting visitor behavior and park efficiency. [12]

Proceedings of the VLDB Endowment. ISSN 2150-8097.

2 BACKGROUND & RELATED WORK

Amusement parks are complex environments in which visitors move across various attractions, food stalls, and facilities, creating dynamic crowd patterns between different points of interest. Optimizing visitor flow and park operations is crucial for ensuring an enjoyable and safe experience for guests and hence maximizing the park's revenue.

Most theoretical models, such as the crowding analysis by Yuan and Zheng [16] or the macro- to micro-level model by Ivancevic *et al.* [3], as well as most simulative approaches such those as presented in publications by Sung *et al.* [14] with their scalable approach or Simonov *et al.* [11] in their multi-agent simulation, focus on rather specific aspects of crowd dynamics.

Yuan and Zheng's study on mitigating theme park crowding [16] focuses on strategies to alleviate congestion through predictive modeling and dynamic crowd management. By integrating factors such as visitor behavior patterns, attraction capacities, and temporal variations, they aim to enhance visitor distribution across the park in real-time, improving operational efficiency and reducing wait times. Similarly, Ivancevic *et al.*'s macro- to micro-level model [3] combines overall crowd density and movement patterns with micro-level interactions between individual agents, providing a framework for accurately simulating complex crowd scenarios, including emergency evacuations and large public gatherings.

Sung *et al.* [14] introduce a scalable approach for crowd simulation, utilizing a situation-based control structure that allows agents to adapt their behaviors based on specific contexts. This method enhances the realism of crowd simulations without significantly increasing computational demands. Simonov *et al.*'s multi-agent simulation [11] emphasizes scalability and flexibility, handling diverse scenarios from urban planning to disaster response by incorporating detailed agent behaviors and environmental factors.

Bridging this gap, XCrowd seeks to provide a solution by employing crowd simulation on graph networks to provide a realistic and adaptable, yet computationally feasible approach to amusement park management. We introduce XCrowd, detailing its methodology, implementation, and potential implications for amusement park management.

By offering a tool to evaluate crowd behavior under specific conditions, XCrowd aims to help validate the efficiency of park layouts and scheduling models, as well as the accuracy of theoretical scenarios, bringing an alternate perspective to the issue of

aligning theoretical amusement park optimization and real-world operations.

3 APPROACH

3.1 Technology & Model

XCrowd is designed as a client-side web application in order to offer real-time simulation capabilities and easy-to-access cross-platform support. It makes use of the *Pixi.js* graphic engine [2] for efficient rendering. The core of the simulation relies on a linked graph structure such as shown in Figure 1, which effectively models the layout and pathways within an amusement park. Here, nodes can represent either curves and crossings of a path or points of interest (POI). In case a node represents a POI, it is assigned one of the *node behaviors* shown in Figure 2. This will change the node's behavior – the *Entrance* node, for example, will allow visitors to enter the park – and allow guests to choose these nodes as their next stop in order to visit a ride on their list or satisfy a need.

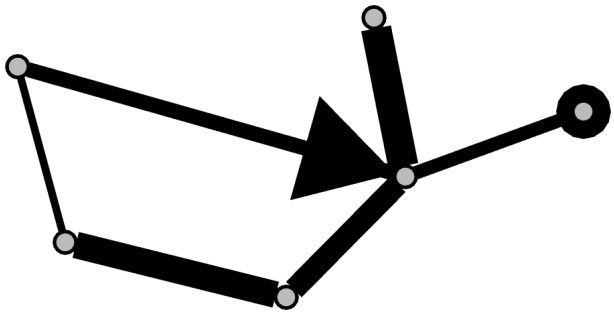


Figure 1: A simple linked graph as can be created in the application's editor. ***Dots represent nodes (cross junctions) and lines represent edges (paths). Thickness and length together represent the capacity of an edge. Arrowheads are displayed if a path is directed and passable one-way only.***

Figure 2: The *node behaviors* available in XCrowd. ***From left to right: Park entrance – roller coaster or similar attraction – food stand or restaurant – drink stand – bench or resting place – restroom – footpath.***

When entering the amusement park, in order to be created at an *entrance* node, the guest needs to choose a set of attractions they unconditionally want to visit. This is meant to mimic how most park guests have certain rides they definitely want to try, such as Europa-Park's *Silver Star* or *Blue Fire* roller coasters. Currently, a list of three to seven nodes of type *roller coaster* is chosen at random and a route to the first POI is computed. Once a guest reaches a POI, the corresponding needs are satisfied, the next POI is chosen based on the guest's current needs, and a new path is computed. For example, if a person reaches a *roller coaster* node, the value for *boredom* is reduced and the value for *nausea* might rise. Now, the highest value among the person's needs might be *hunger*, so a node of type *restaurant* is chosen next. Reaching it will reduce the guest's *hunger* and *thirst* values. This goes on until the guest has visited all POI on their list, upon which a route back to an *entrance* node is computed and the guest is removed.

The pathfinding algorithm, at its core, relies on Dijkstra's algorithm. Additionally, however, to make sure visitors avoid congested paths as well as to simulate slight acts of human randomness, each distance is multiplied by a *punishment* factor $punishment(e)$ and a slight random factor.

The cost of an edge e is then computed as

$$cost(e) = distance(e) \cdot punishment(e) \cdot (r \cdot 0.5 + 0.75), \quad (1)$$

with

$$punishment(e) = 1 + \frac{100}{(\delta + 1)^{100}}, \quad (2)$$

where

$$\delta = \frac{\max(0, currentCapacity - neededCapacity)}{maximumCapacity} \quad (3)$$

and r is a random value between 0 and 1.

This computation uses an edge's capacity value *maximumCapacity*, ranging from 0 to ∞. It denotes how many people may be on a certain section of the path at the same time. Further, *neededCapacity* denotes the space needed for a person to enter the edge, which is always 1 but could be changed to allow for groups of people. Lastly, the edge's current capacity *currentCapacity* is defined by the currently remaining space. It is also possible for an edge to be directed, as it is often the case in queues in front of attractions, in which case the pathfinding algorithm considers the edge in one direction only.

As performance tests have confirmed, a modern personal computer is capable of simulating several thousand visitors in real-time. To ensure realistic pathfinding, XCrowd employs a modified version of Dijkstra's algorithm that takes into account the current capacity of edges, allowing visitors to navigate the park and avoid congestion in a manner reflective of real-world crowd dynamics. The application utilizes a real amusement park map as its foundational base, providing a realistic reference for creating the graph network.

To aid the process, XCrowd offers an easy-to-use graphical editor shown in Figure 3 that lets the user create and modify a graph network based on a reference background image. In addition, an analytics tool helps record the simulation data every n frames and saves it in a CSV file for further processing.

A publicly accessible instance of XCrowd in its current form for readers to interact with and assess the project is available at https://xcrowd.cloudlab.zhaw.ch/.

3.2 Implementation

To facilitate the set-up and usage of XCrowd as much as possible, the software is implemented as a *Node.js* application [6] using the

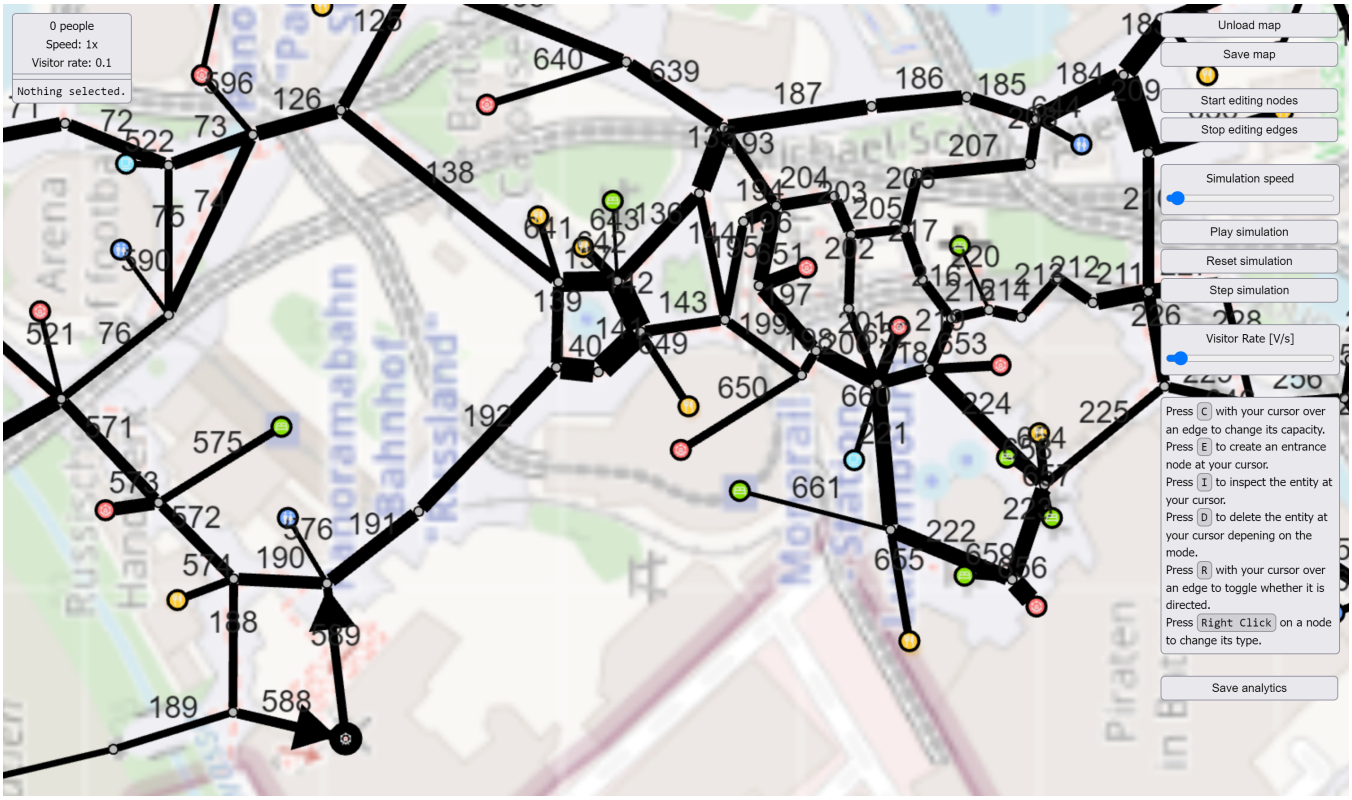


Figure 3: The graphical editor in XCrowd. ***Users can create and modify graph networks based on a reference background image, even with the simulation still running.***

Express.js framework [5], providing a robust base layer for serving resources and handling HTTP traffic. Since early performance tests showed that up to several thousand visitors can be simulated on a standard computer without major performance impacts, we opted for a client-side web application to allow for real-time simulation capabilities with minimal latency. The rendering is done using *Pixi.js* [2], a highly efficient WebGL-based rendering engine. Further, we make use of jQuery [4] for DOM manipulation and Toastify.js [15] for displaying status notifications.

A `GraphMap` object holds a set of `MapNodes` and `MapEdges`, which represent the park's layout. Each `MapNode` knows its properties such as its coordinates or the type of location it represents, as well as which `MapEdges` are connected to it. Each `MapEdge` knows its properties such as its length, the `MapNodes` it connects, its capacity and whether it is directed. The `MapEdge` also tracks the number of `Persons` currently on it as this is necessary for the congestion avoidance factor in the pathfinding algorithm.

Park visitors are represented as `Person` objects. When they are created at a `MapNode` whose `NodeBehaviour` is of type `MapEntrance`, which allows `Persons` to enter and exit the park, they are assigned a list of stops they need to visit. Upon initialization, the `Person`'s route to the first stop is computed using a modified version of Dijkstra's algorithm that takes into account the current congestion of the `MapEdges` as well as random factors.

The simulation is run in discrete steps, each representing a small unit of time. They are known as *ticks* or *frames*. Due to the nature of the simulation and varying computational load, the duration of a tick is not necessarily fixed. Instead, the simulation tries to run at a fixed frame rate, which in XCrowd is set to 60 frames per second, but may drop below this rate if the computational load is too high. In this case, the simulation will try to catch up in the next frame by simulating a slightly longer time span. This is done by multiplying the time passed since the last frame by a factor called *delta time*, which is the ratio of the current frame rate to the target frame rate. This way, the simulation will always run at the same speed, regardless of the computational load. Note, however, that this requires paying special attention when implementing the simulation, as the simulation code must be written in a way that ensures that the simulation's state changes are consistent regardless of the frame rate.

Each simulation step, each `Person` object moves along its current `MapEdge`. Once it reaches a `MapNode`, one of two possible events may have occurred: Either, the `Person` has reached its next stop, which is for example the last `MapNode` on the `Person`'s current route. If so, the stop is removed from the `Person`'s attraction list and a new route to the next stop is computed. The next stop is chosen based on the `Person`'s attraction list, current needs and position within the park. Alternatively, the `Person` has simply reached a `MapNode` along its current route, in which case the `Person` checks whether

the next MapEdge's current capacity is greater than zero *i.e.* whether there is space for the Person to enter. If so, the Person leaves its current MapEdge and enters the next one, updating any relevant counters. Otherwise, a new route to the next stop is computed. This way, the Person will try to avoid congested pathways and might find a way around the congestion much like a real-world visitor would.

Since in some circumstances of high congestion, the Persons may not be able to find a route to their next stop and end up calling the route computing algorithm over and over, we introduced a cooldown of 20 simulation steps for each Person. This way, the Person will simply wait for up to 20 simulation steps (1/3s) before trying to find a new route, which reduces the computational load dramatically.

Even though under specific circumstances, a situation may occur in which the Persons on a MapEdge are unevenly distributed, this poses no problem in the overall simulation as this effect is only noticeable on GraphMaps with very few or very long and slim MapEdges and vanishes as the GraphMap becomes more detailed.

4 RESULTS

4.1 Simulation Set-up

We chose to test our model on the *Europa-Park* in Rust, Germany, which in 2022 was Europe's second most visited amusement park after *Disney Land* in Paris, France. Each year, a total of around 5.4 million guests visit *Europa-Park*. With the park being open for 294 out of 365 days a year, and official opening hours going from 9:00 o'clock in the morning to 6:00 o'clock in the evening, this equates to ~2'040 guests per hour or about 0.57 visitors per second. [8] With each simulation tick, a slightly randomised number of people p is let into the park, with $\mathbb{E}[p] = {}^{0.57}/_{t}$ and t being the number of simulation ticks per second. Note that this approach does not take into account varying visitor numbers throughout the day or year, such as increased crowds at the park entrance in the morning or lower visitor numbers during the winter season.

For the layout of the graph network, we used freely available map tiles from *OpenStreetMap* [7] as the base layer on which we then traced paths and placed POI using *XCrowd's* editor according to the reference. It is important to highlight that, since we were not able to retrieve official data on the park's paths, the paths' capacities were estimated based on their length and width, among other features such as surrounding geometry. An overview of the final graph network is shown in Figure 4.

Since simulating human needs is quite a complex task, we decided to simplify the model by only considering the most basic needs, such as hunger, thirst, and tiredness among others. We took inspiration from existing simulations of park guests, such as the video game series *RollerCoaster Tycoon* [10], to narrow down the relevant needs and their recovery times. As laid out in Figure 5, each visitor's needs are set at random upon entering the park and changed over the course of the visit according to the corresponding *recovery time*. For example, a guest might not need to eat again within 2 hours of having a meal, but wants to get back on an attraction in less than half an hour after their last roller coaster ride. The guest's other properties are assigned at random as well: The walking speed is based on previous research done by Rastogi *et al.* [9] and varies between 60 and 70 $^{m}/_{min} \approx 1.08 \pm 0.08\ ^{m}/_{s}$. Additionally, as mentioned in Section 3.1, each guest chooses a list of obligatory nodes of type *roller coaster* to visit. Its three to seven items are assigned at random.

4.2 Evaluation

To evaluate the accuracy of our model, we chose to compare the resulting simulation data to real-world data. As we did not have access to exact or official visitor numbers, let alone the precise figures for individual sections of the path that would have been required, retrieving a ground truth dataset was rather difficult.

We opted to use GPS tracking data provided by *Strava*, a social network focused on tracking physical exercise, with over 100 million community members registered. Anonymised GPS trails of all users are publicly available in the form of a heat map. [13]

To be able to compare our simulation results from *XCrowd* to the real-world reference, we reconstructed the heat map using the tiles streamed to the *Strava* client. In addition, we determined the gradient map used for rendering the heat map, also by analyzing the *Strava* client. Although no documentation is given, the values seems to follow an exponential curve.

To facilitate the sampling process, we created a script with a graphical user interface (GUI), shown in Figure 6, that allows the user to select a coordinate on the heat map, which is then looked up in the gradient map and displayed as a final busyness value.

Obtaining equivalent values for each graph edge in the simulation was done by taking snapshots of the state of the graph during the simulation. Afterwards, the number of people on each edge was averaged out over all recorded simulation frames. The simulation was run for one hour in simulated time on 10× real-time speed. Only frames 2000 and later were considered, to leave enough time for the simulation to stabilise.

The sampling is done at the centre coordinate of each of 73 randomly chosen edges in the graph, resulting in pairs of a busyness value sampled from the heat map and a busyness value measured using *XCrowd's* analytics tool each.

4.3 Validation

Before moving to the validation, we remove one outlier from the dataset: Edge #377 has a measured busyness value of almost zero. We assume that, purely by chance, the analytics tool took most snapshots of the simulation when no guests were using this quite short path. As surrounding edges only reachable by edge #377 report reasonable values, we may safely assume edge #377 to be an outlier and will be doing all further computations without it.

4.3.1 Simulation Accuracy. To validate our simulation model against ground truth, we compare the simulated busyness value of an edge to its real-world counterpart on the heat map, as illustrated in Figure 11. The correlation plot is shown in Figure 7. Note that, since the heat map value scale is exponential, we use the natural logarithm $\log(b)$ for our simulated values when comparing them to values sampled from the heat map.

A Pearson correlation test confirms a significant correlation between the simulated and reference values with a correlation coefficient $\rho \approx 0.704$ and a p-value of around $5.529 \times 10^{-12} \ll 0.05$.

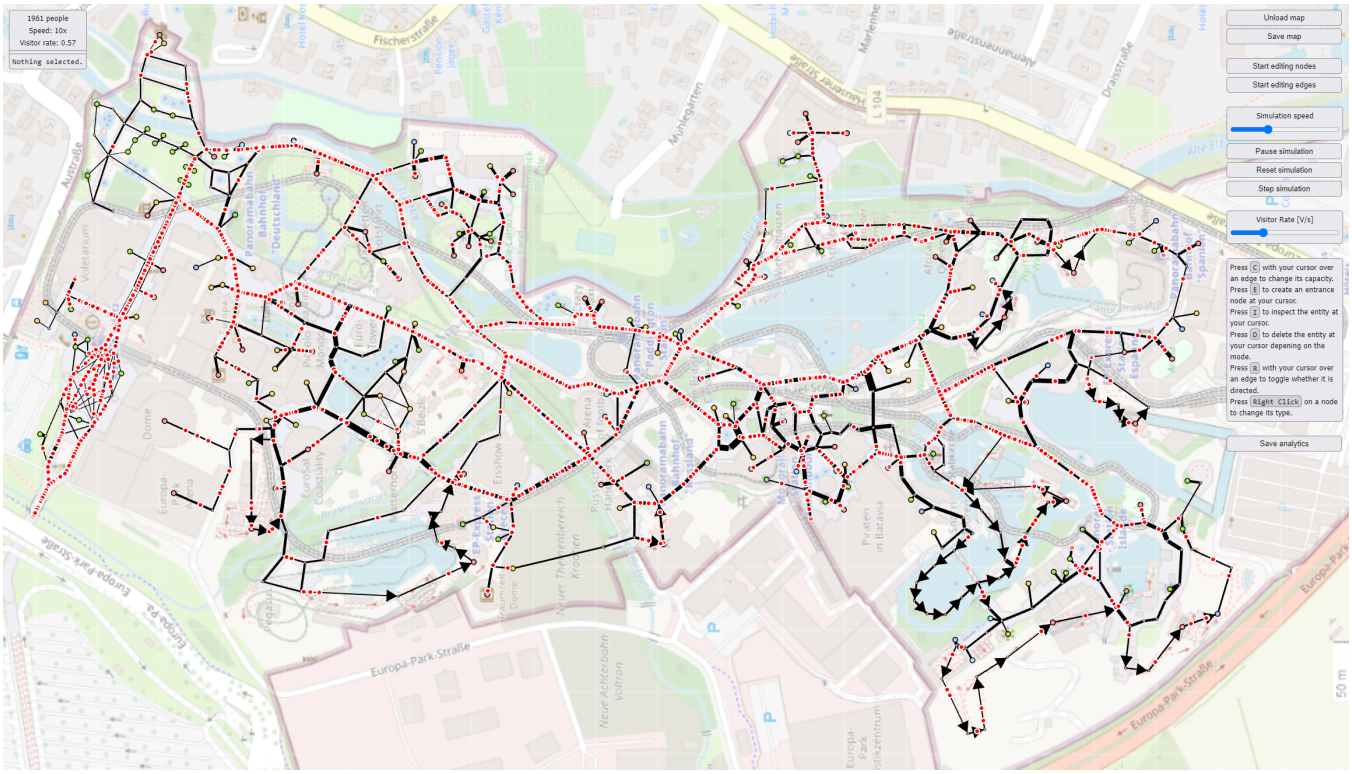


Figure 4: Overview of the *XCrowd* application interface. ***Dots represent nodes of types shown in Figure fig:node-behaviours, black lines of different thickness represent edges of different capacity, and red dots represent people. The park layout used for reference is Europa-Park in Rust, Germany and was rendered from publicly available OpenStreetMap data. [7]***

Need	Default	Recovery Time
Bladder	$0.8 \cdot r$	2 h
Boredom	$0.5 \cdot r$	30 min
Hunger	$0.5 \cdot r$	2 h
Nausea	$0.2 \cdot r$	15 min
Thirst	$0.75 \cdot r$	1 h
Tiredness	$0.05 \cdot r$	2 h

Figure 5: The guests' needs. ***The default values are set when the guest is created, with r being a random value between 0 and 1. Recovery times determine how quickly a need fully changes during a visit.***

4.3.2 Simulation Consistency. Although the spread in Figure 7 does seem rather consistent between edges of different capacity, we can confirm this by performing separate correlation tests on two different groups. We split the set of edges into groups of equal size, ordered by capacity. With a median capacity of 50 people, group A contains 37 edges of capacity $c \leq 50$ and group B contains 35 edges of capacity $c > 50$. In addition to the plots in Figure 8, we confirm their individual correlations through two more Pearson correlation tests with resulting correlation coefficients of $\rho_A \approx 0.683$, $\rho_B \approx 0.492$ and p-values of $p_A \approx 3.226 \times 10^{-6} \ll 0.05$, $p_B \approx 2.680 \times 10^{-3} < 0.05$.

4.3.3 Parameter Accuracy. Since the capacity values of the graph's edges are pure estimations, it is helpful to see how well our estimations hold up. By plotting the busyness and capacity for each edge in Figure 9, we notice that no edge's relative simulated busyness is higher than 34%, *i.e.* all edges operate well below their capacities.

As none of the average busyness values are anywhere near their maximum capacities, we may assume that either the paths in *Europa-Park* allow for a number of guests well above usual demand or our estimated capacities were consistently too high. Without any official data, however, it is not possible to confirm this hypothesis.

The fact that the edges' capacities were not a limiting factor for the simulated visitor flow allows us to perform another check confirming that the visitors in the simulation actually demonstrate realistic movement patterns. Although the edges' estimated capacities may have been consistently too high, they are likely to be accurate relative to each other. Thus, they should be able to serve as a rough guide for capacity comparisons among each other and we should be able to tell whether visitors show a similar movement distribution to the real world. To confirm this behavior, we plot the edges' simulated busyness against their estimated capacities in Figure 10. Another correlation test also confirms a significant

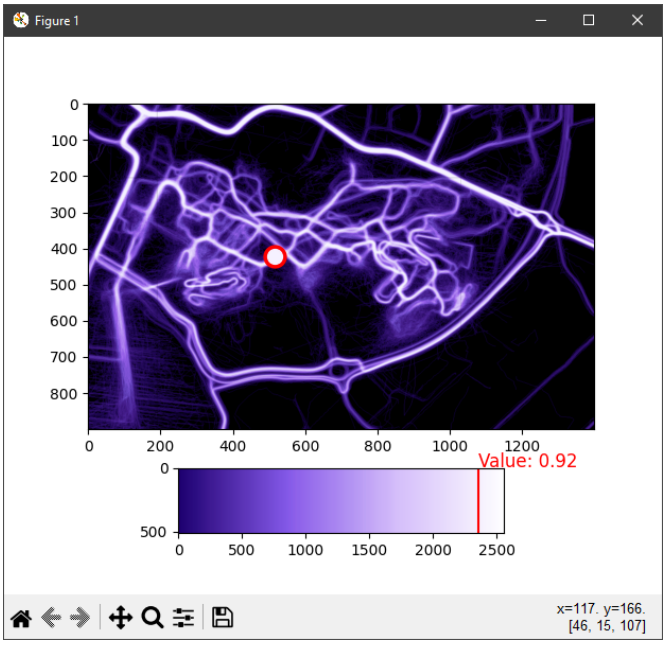


Figure 6: The heat map sampler GUI. ***When selecting a coordinate in the top heat map, the script calculates and displays the corresponding path's busyness value on the bottom gradient.***

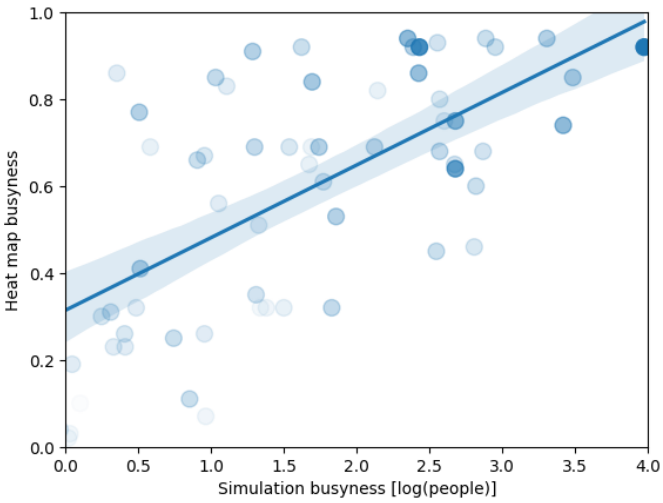
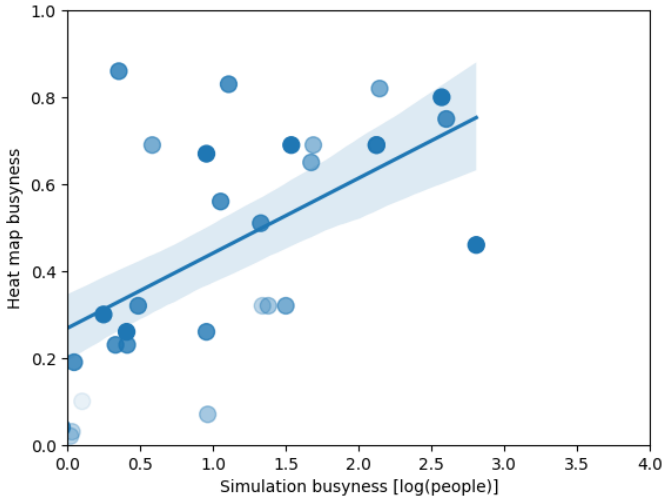


Figure 7: Simulated busyness *vs.* ground truth. ***Opacity represents the capacity of an edge.***

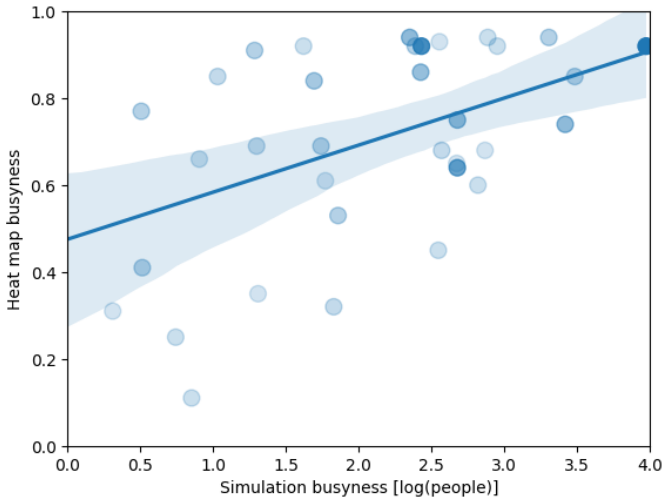
correlation with a correlation coefficient of $\rho \approx 0.643$ and a p-value of $1.103 \times 10^{-9} \ll 0.05$.

5 DISCUSSION & FUTURE WORK

As mentioned in Section 3.1, the POI on a visitor's must-visit attractions list are chosen at random, not taking into account a specific ride's popularity, for example, which may shift the busy areas in the park away from the real-world hotspots. Furthermore, as was



(a) Group A with $c \leq 50$



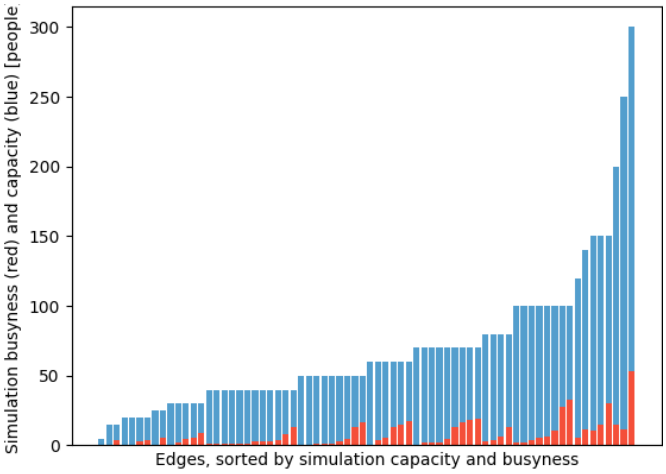
(b) Group B with $c > 50$

Figure 8: Simulated busyness *vs.* ground truth. ***Opacity represents the capacity of an edge.***

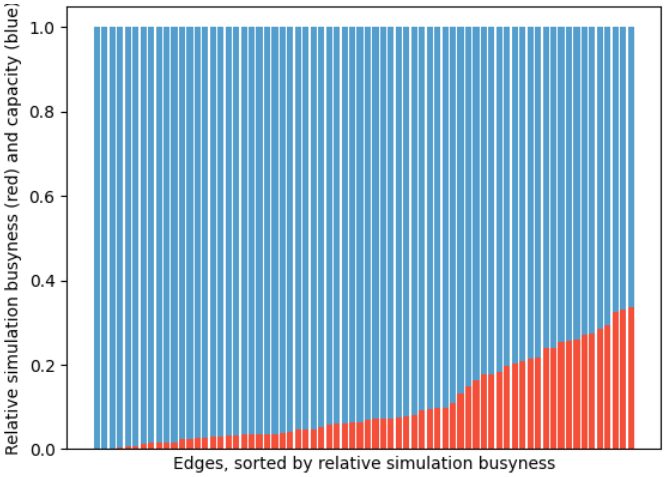
shown in Section 4.3.3, the guests never fully use up any edge's capacity and are thus rarely repelled by congestion. Whether this is a realistic assumption is unclear but it should be noted that this could be another contributing factor. Finally, it is noteworthy that the status of the two data sources (the *OpenStreetMap* tiles and the *Strava* heat map) is different, partly due to recent conversions and renovations at *Europa-Park*. [1] Given these circumstances, the results are reasonable.

As was shown during the validation process in Section 4.3, *XCrowd* in its current form is a useful and accurate simulation tool whose primary functionality is implemented. However, future improvements to the software and reference data could greatly increase the simulation's accuracy and consistency.

Better reference data such as a more precise park layout or official path capacities and more fine-grained visitor numbers throughout



(a) Absolute



(b) Relative

Figure 9: Edges *vs.* simulated busyness. ***Edges are sorted by estimated absolute capacity and busyness and by relative simulated busyness, respectively.***

the day would greatly improve the simulation parameters and thus increase the likeliness of an accurate simulation.

During the simulation, by far the most time-consuming operation is the pathfinding algorithm, run each time a new visitor enters the park or a current guest wants to find a route to a new destination. Reducing the computation time and improving the performance would allow for larger and more complex parks and the implementation of more possibilities and a more authentic simulation, such as realistic wait times for rides, restaurants and similar POI, alternative ways for guests to move around the park, *e.g. Europa-Park*'s *Monorail*, or dynamic events like emergencies or weather changes. In order to maximize the potential for efficiency, the software should shift to a compiled language and make use of server-side computation, only streaming the results to the client. Multi-core computation would be ideal for a multi-agent simulation like this one.

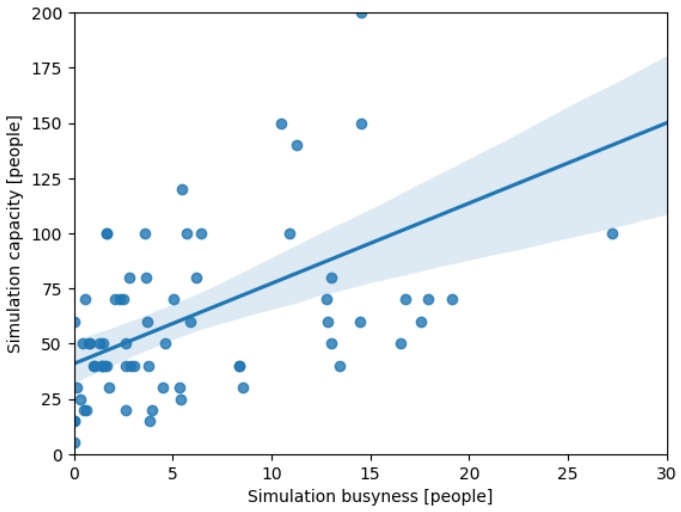


Figure 10: Simulated busyness *vs.* estimated capacity. ***Even without being limited in their decisions, simulated park guests follow realistically plausible routes.***

Although the usage of *XCrowd* in this paper focuses solely on amusement parks, the software is not limited to this domain. The simulation can be applied to various scenarios, such as public transportation, shopping malls, or event management, where crowd movements play a crucial role.

Future iterations should also focus on improving the user interface and experience. This includes working on the simulation customization options, the map editing tools, and the inspector capabilities, making the software more intuitive and user-friendly for theme park managers and planners.

To enhance compatibility and streamline the workflow, *XCrowd* could support common standards and allow direct import of data such as the park's expected visitor numbers, opening hours, or geographical layout without any additional manual work. This would facilitate the integration of real-world park semantics and enable users to work with a wider range of data sources seamlessly.

XCrowd offers a vision for the future of amusement park management evaluation by combining theoretical models with a more realistic crowd simulation. The software helps gain valuable insights into visitor behaviours and park dynamics, paving the way for more informed decision-making and enhanced visitor experiences.

REFERENCES

[1] 2024. Europa-Park investiert in mehr als 40 Bauprojekte. *Europa-Park-Presse* (2024). https://presse.europapark.com/de/presse/nachricht/datum/2024/01/23/europa-park-investiert-in-mehr-als-40-bauprojekte
[2] Mat Groves, Matt Karl, Sean Burns, Andrew Start, Shukant Kumar Pal, Milton Candelero, Tianlan Zhou, Viktor Persson, and Dmytro Soldatov. [n.d.]. *PixiJS*. https://pixijs.com/
[3] Vladimir G Ivancevic, Darryn J Reid, and Eugene V Aidman. 2010. Crowd behavior dynamics: entropic path-integral model. *Nonlinear dynamics* 59 (2010), 351–373.
[4] John Resig and jQuery contributors. 2006. jQuery. https://jquery.com/.
[5] OpenJS Foundation and Express.js contributors. 2010. Express.js. https://expressjs.com/.
[6] OpenJS Foundation and Node.js contributors. 2009. Node.js. https://nodejs.org/.

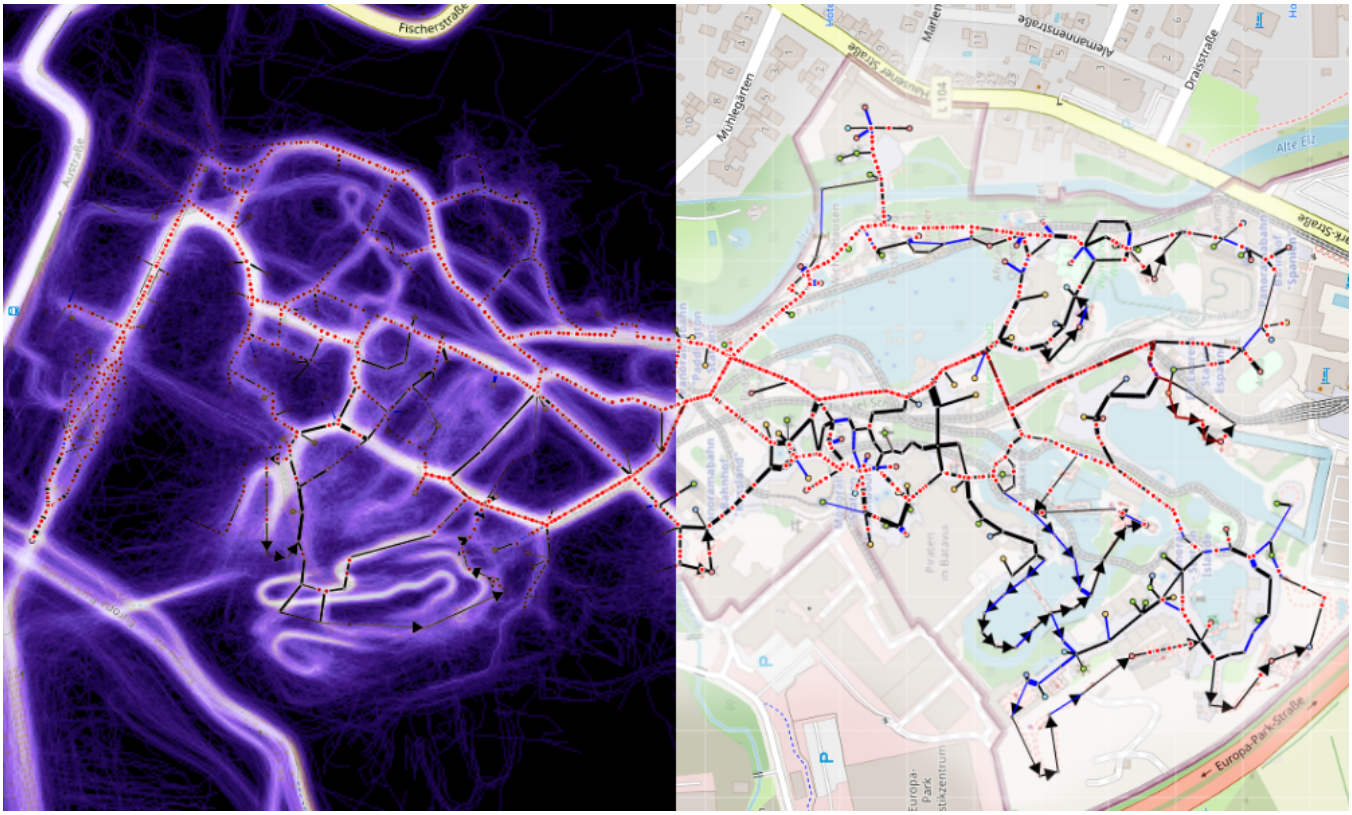


Figure 11: The heat map overlaid on top of *XCrowd's* map overview. *The brighter a pixel is on the heat map, the busier it is. Black means no recorded activity while bright areas resemble a large number of GPS tracks.*

[7] OpenStreetMap contributors. 2017. Planet dump retrieved from OpenStreetMap. https://www.openstreetmap.org

[8] Martin Palicki, Beth Chang, Linda Cheu, John Dietz, Doris Li, Jodie Lock, Andrew Logan, Daisy Long, Jason Marshall, Michael Posso, Francisco Refuerzo, Catherine Ritter, John Robinett, Chris Yoshii, and Skyler Young. 2022. Museum Index: Theme Index 2022. *TEA/AECOM Global Attractions Attendance Report* (2022).

[9] Rajat Rastogi, Ilango Thaniarasu, and Satish Chandra. 2011. Design implications of walking speed for pedestrian facilities. *Journal of transportation engineering* 137, 10 (2011), 687–696.

[10] RollerCoaster Tycoon Wiki. [n.d.]. Guests and Staff. https://rct.fandom.com/wiki/Guests_and_Staff

[11] Andrey Simonov, Aleksandr Lebin, Bogdan Shcherbak, Aleksandr Zagarskikh, and Andrey Karsakov. 2018. Multi-agent crowd simulation on large areas with utility-based behavior models: Sochi Olympic Park Station use case. *Procedia Computer Science* 136 (2018), 453–462.

[12] G Keith Still. 2000. *Crowd Dynamics.* Ph.D. Dissertation. University of Warwick UK.

[13] Strava. 2024. Strava Global Heatmap. https://www.strava.com/maps/global-heatmap

[14] Mankyu Sung, Michael Gleicher, and Stephen Chenney. 2004. Scalable behaviors for crowd simulation. *Computer Graphics Forum* 23, 3 (2004), 519–528. https://doi.org/10.1111/j.1467-8659.2004.00783.x

[15] Varun Adiyeri Parambath and toastify.js contributors. 2017. toastify.js. https://apvarun.github.io/toastify-js/.

[16] Yuguo Yuan, Weimin Zheng, et al. 2018. How to mitigate theme park crowding? A prospective coordination approach. *Mathematical Problems in Engineering* 2018 (2018).