Appendices

Appendix A: Executable

A sample executable implementing our framework can be found in the supplemental material.

Appendix B: Metrics

Below we describe various metrics that can be easily used in our framework. While a metric can take any form, for ease of notation we describe the following ones as their value at a given timestep. The complete metric (M) is then the absolute value of the sum of its results over all timesteps (M_k):

$$M = |\sum_{k=1}^{m} M_k|. \tag{7}$$

Note that all metrics are defined to have a value of zero whenever the simulated motion exactly matches the reference data.

Microscopic Data Metrics

In general, microscopic data can be used with a wide variety of similarity metrics, which capture different aspects of the data. Here we summarize several microscopic metrics tested in this work. In all cases of microscopic data, we assume the reference data $\mathbf{z}_k$ consists of a vector of positions for all the agents.

The **absolute difference metric (D)** computes the total distance in position over all agents over all timesteps:

$$D_k = \|\mathbf{z}_k - \mathbf{x}_k\|. \tag{8}$$

The **path length metric (L)** compares the difference in total length traveled between agents in the reference data and the simulated agents:

$$L_k = (\mathbf{z}_{k+1} - \mathbf{z}_k) - (\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{9}$$

The **inter-pedestrian distance metric (I)** compares the difference in average distance (as a 2-norm) between every pair of agents. If P is the ensemble of all agent pairs $P = \bigcup\{i, j\}$, then:

$$I_k = \sum_P |\mathbf{x}_k^i - \mathbf{x}_k^j| - \sum_P |\mathbf{z}_k^i - \mathbf{z}_k^j|. \tag{10}$$

The **progressive difference metric (P)** measures the absolute difference between the simulated agents and the reference data *when the simulation is reinitialized at each timestep*.

$$P_k = \|\mathbf{z}_{k+1} - f(\mathbf{z}_k, speed(\mathbf{z}_k), \mathbf{z}_m, \mathbf{p})\|. \tag{11}$$

Macroscopic Data Metrics

Unlike microscopic metrics, which are computed per agent, macroscopic metrics are computed over all the agents. In these cases, the form of the reference data ($\mathbf{z}_k$) generally varies for each metric.

The **vorticity metric (V)** measures the vorticity of the crowd flow. First a velocity field $\overrightarrow{v}$ is generated from the agents' motion, then:

$$V_k = \mathbf{z}_k - (\nabla \times \overrightarrow{v}), \tag{12}$$

where $\mathbf{z}_k$ is the target vorticity for the current timestep.

Finally, the **fundamental diagram metric (F)** compares the speed of an agent to the density of agents in its location. This metric is inspired by the field of pedestrian dynamics, where it is commonly used to measure pedestrian flow rates (e.g., [CM12]). Our implementation of the metric defines a "gate" area on the agent's path (as in [CSC09], which allows us to compute the density of population at an agent's location ($\frac{numberAgentsInsideGate}{areaOfGate}$) when the agent is inside this gate.

$$F_k = |\mathbf{z}_k(d_k) - \|\mathbf{v}_k\||, \tag{13}$$

where $\mathbf{d}_k$ is the density at the location of each agent while inside the gate, and the reference data $\mathbf{z}_k$ is a function that maps density to speed based on results known from human motion studies.

Appendix C: Optimization Techniques

Greedy algorithm

The greedy approach works by first selecting random parameters for every agent. The chosen data similarity metric is then evaluated to establish a baseline measure of how well the simulation matches the data. The algorithm then performs several iterations, where in each iteration starts with the best set of simulation parameter seen so far and one simulation parameter is randomly replaced by a sample from the user defined distribution. This new set of parameters is evaluated, whichever set of parameters has the lowest error metric over all the iterations is chosen as the optimal parameters for the agents.

Simulated annealing

The main limitation with a greedy approach is that it will get stuck in local minimum in search space. Simulated Annealing (SA) address this problem by occasionally accepting a slightly worse evaluation as the current best parameter set. This allows the procedure to "jump out" of a local minimum with some non-zero probability. By convention, the probability of choosing a parameter set with worse evaluation decreases over time, and decreases if the new evaluation is much worse than the old one.

Algorithm 1 gives the pseudocode for the process where:

Algorithm 1: Simulated annealing.

```
k ← 0                                   // initialize loop counter
while k < K do
    T ← temperature(k, K)               // compute temperature
    s_new ← neighborState(s)            // try new neighbor
    e_new ← cost(s)                     // compute cost
    if move(e, e_new, T) then           // is new state better?
        s ← s_new; e ← e_new            // yes, change state
    end
    if e < e_best then                  // did we find a new minimum?
        s_best ← s; e_best ← e          // save new optimum
        k ← 0                           // reset loop counter
    end
    k ← k + 1                           // increase loop counter
end
```

neighborState(): pick a new random value for a random parameter according to the parameter's base distribution

move(): is $True$ iff $e_{new} < e_{old}$, $exp(\frac{e_{old}-e_{new}}{T})$.

temperature(): is $\frac{K-k}{K}$, k being the number of iterations with no improvement and K the number of such iterations allowed.

cost(): the cost as returned by the currently used metric.

Genetic algorithm

Like simulated annealing, genetic algorithms seek to overcome the problem of local minima in optimization. This is accomplished by keeping a pool of parameter sets which have performed well so far and during each iteration creating a new pool of potential states based on combining and modifying the previously successful candidates.

Algorithm 2: Genetic algorithm.

```
pop ← initialize()                  // initialize population
while true do
    selection(pop)                  // evaluate and select fittest
    if termination() then           // should we terminate?
        stop                        // yes, stop loop
    end
    pop ← reproduction(pop)         // new generation
end
```

Algorithm 2 provides pseudocode for the method given the following functions:

initialize(): parameters randomly initialized in accordance with the base distribution for each parameter.

selection(): individuals are sorted according to their score and divided into 3 groups: Best (of size m), Middle (of size n) and Worst (the remaining individuals).

termination(): the algorithm is terminated after finding K successive loop iterations without any new optimum.

reproduction(): based on which group it belongs to, a parameter set is attributed three probabilities α, β and γ. For each parameter of this individual, α decides if the value is changed or not, β decides if the value is changed by crossover or mutation and, finally, γ decides which type of mutation is done.

- crossover: a crossover is done by copying a value from an individual belonging to the Best group.
- mutation: a mutation is done by picking a new value at random based on either the base distribution or the current real distribution of an individual from the Best group (according to γ).

Covariance Matrix Adaptation

Lastly, we also tested the CMA optimization algorithm [HHOO96], implemented in the Shark Machine Learning Library [sha]. Being a solution-pool based technique it shares the same general algorithm 2 as the genetic algorithm except it generates new solutions by picking values from distributions defined by a covariance matrix that is continuously modified over the iterations.

Appendix D: Optimization Comparison

Optimizing crowd parameters is a unique and challenging problem. Because most simulation methods have several parameters to tune *for each agent*, even moderately-sized scenarios with a few dozen agents can become hundred-dimensional optimization problems. We tested several different combinatorial optimization strategies on different scenarios to measure how they perform on two measures: computational speed (how long it takes to converge to an answer) and quality (how close the answer is to the true optimum).

In total we tested 8 optimization algorithms: the four algorithms described above (Greedy algorithm (G), Simulated Annealing (SA), a Genetic Algorithm (GA), and CMA), as well as four hybrid approaches. The first hybrid approach was to take the solution from the Genetic Algorithm approach (which searches broadly in parameter space) and refine it using Greedy optimization, denoted as (GA+G). In a similar manner, we have also tried refining the output of the Genetic Algorithm with Simulated Annealing (GA+SA), then CMA+G and CMA+SA.

We evaluated the optimization techniques on four different crowd simulation methods: the RVO2 algorithm, a

Boids-like steering model, the Helbing Social Force model, and the Vision-based steering model. The initial parameter sets for each of these methods are given in Appendix E.

The optimization methods were tested across different scenarios. Because of the stochastic nature of the optimization techniques, each scenario was run multiple times with each simulation method to ensure a statistically meaningful comparison. First, we found that the scores for various metrics were improved by all optimization methods, and by a statistically significant margin (Friedman test [Fri37] at the p=0.05 level). Second, we performed a statistical ranking test between optimization methods (post-hoc analysis with the Wilcoxon signed-rank test [Wil45] at the p=0.0018 level). For each pair of optimization methods, this second test measures whether the improvement in simulation scores differs between the two methods.

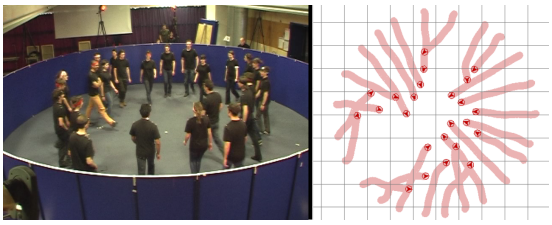
We batched several tests into three sets of scenarios. The first set of scenarios (Fig 12a) involved a small number of agents (2-5 agents) crossing paths to reach their goals; for this evaluation, we used the Difference metric (D). Here GA+G and GA+SA algorithms give the best score improvements and the CMA algorithm is the fastest. The second set of scenarios comes from the data in [ZKSS12] (Fig 12c). In this set of scenarios, many agents (between 30 and 100 agents) walked down a hallway; we evaluated these using the Fundamental Diagram metric. Here, SA, GA+SA and CMA+SA algorithms performed best at optimizing the metric, and the GA and CMA algorithms are the fastest. In the final scenario (Fig 12b), 24 agents walked to antipodal positions in a circle and were tested using the Progressive Difference (P) and Inter-pedestrian Distance (I) metrics. Here, the CMA+G algorithm (resp. CMA+G, GA+SA and CMA+SA) gave the best score improvements, and the CMA algorithm (resp. CMA) is the fastest based on the Progressive Difference metric (resp. Inter-pedestrian Distance).

The complete ranking of the algorithms by their ability to optimize the simulation metrics is given Table 1. A ranking by their computational speed is given in Table 2. As can be seen in these tables, GA+G provided the best balance between runtime and performance. For the rest of the paper, we will use GA+G as the global optimization method.

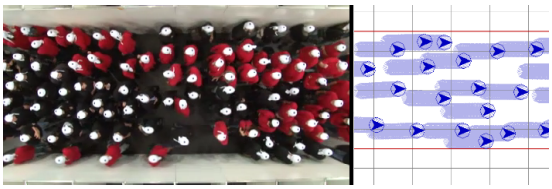
Figure 14 shows the raw score and runtime results. Notations for optimization methods are: (G) Greedy, (SA) Simulated Annealing, (GA) Genetic Algorithm, (CMA) Covariance Matrix Adaptation. Figure 15 shows the Friedman and Wilcoxon tests' results for the score. Figure 16 shows the Friedman and Wilcoxon tests' results for runtime.

Appendix E: Initial Parameters for Optimization

(a) Crossing Scenario



(b) Circle-24 Scenario



(c) Hallway Scenario

Figure 12: *Illustration of reference data used for batch testing. (a) A few people standing on a circle are asked to reach the antipodal positions, they were motion-captured to record their global trajectories. (b) The same experiment as the previous one with a larger number of subjects (up to 24). (c) Several subjects walk through a hallway while being recording with optical tracking equipment ([Zhang et al. 2012]).*

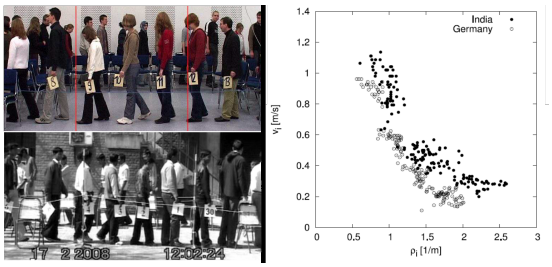


Figure 13: *Figures extracted from [Chattaraj et al. 2009]. Subjects from India (top) and Germany (bottom) were asked to walk in a line. Video analysis was performed to extract fundamental diagrams (speed vs. density).*

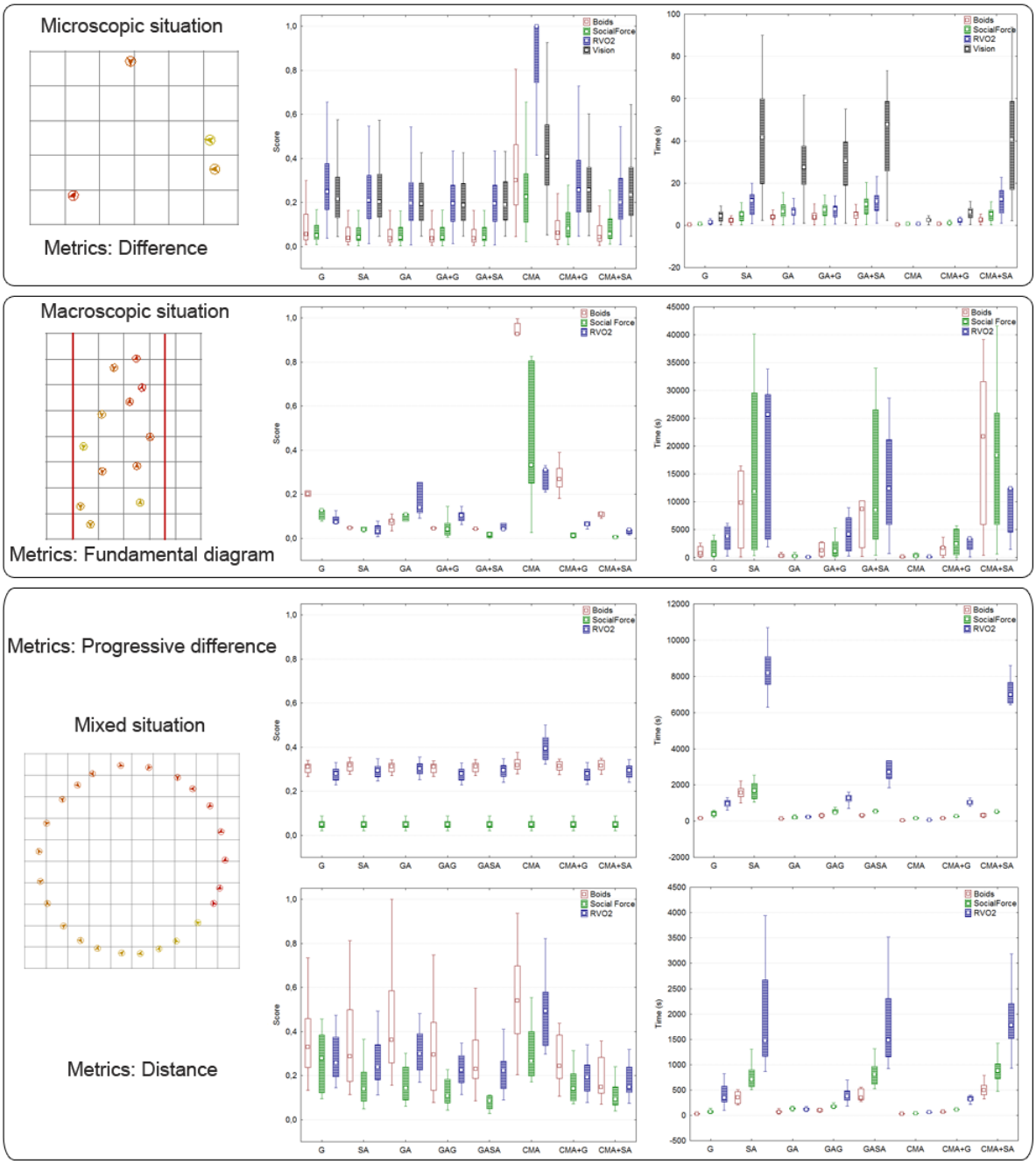


Figure 14: *Summary of the experiment testing optimization algorithms. Each row shows results for a different type of data. In each row, the left side describes the data and metric. The middle shows* $\frac{scoreAfterClibration}{scoreBeforeClibration}$ *with a lower value indicating better optimization. The right shows time in seconds. Each graph shows its results for each model for each optimization method.*

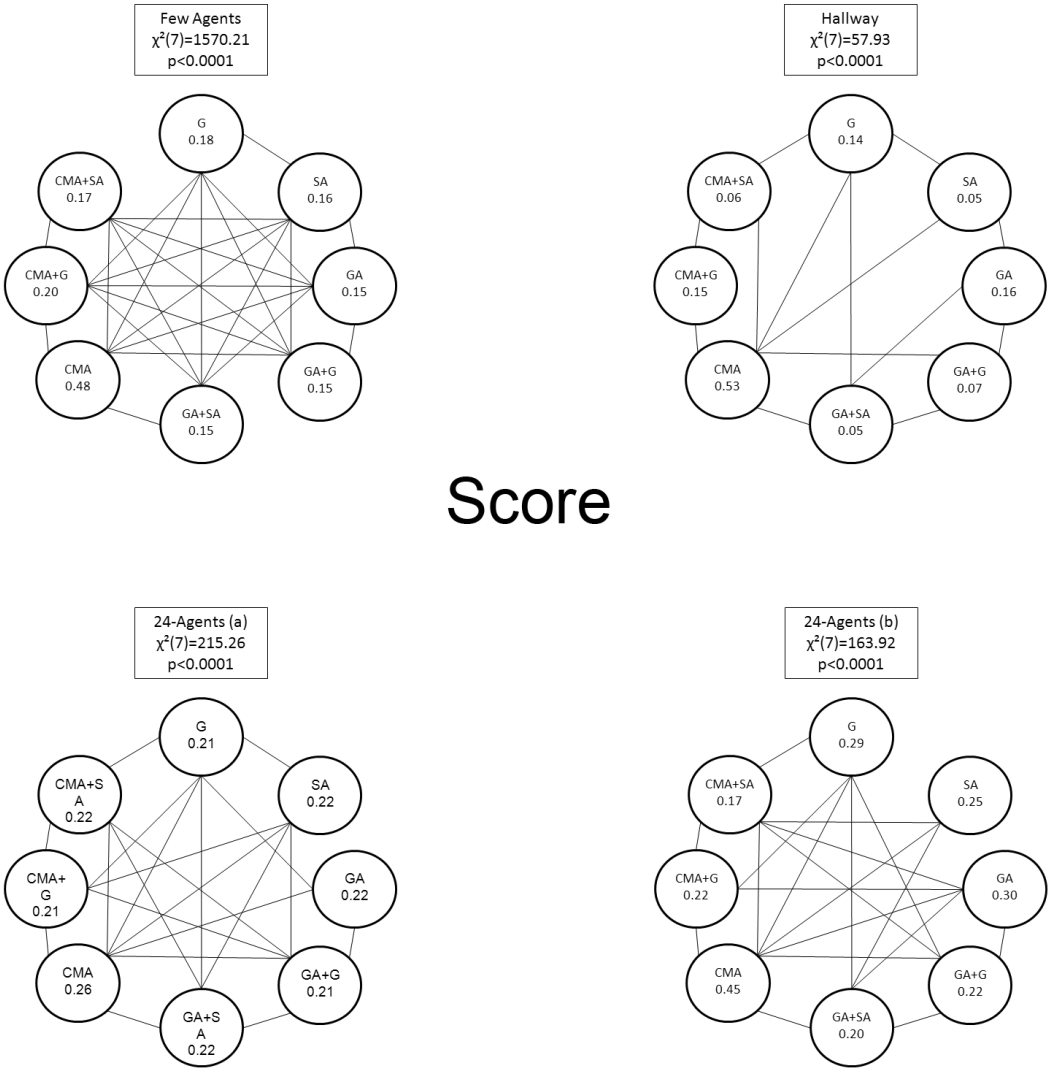


Figure 15: *Statistical results of the Friedman's Anova and Post-Hoc Wilcoxon signed rank tests on the score depending on the optimization algorithm. Mean values are reported for each algorithm. Friedman's Anova showed an influence of the optimization algorithm on the score for each type of data (Few Agents, Hallway, 24-Agents (a), 24-Agents (b)). Significant differences between algorithms are represented through a line ($p < 0.0018$).*

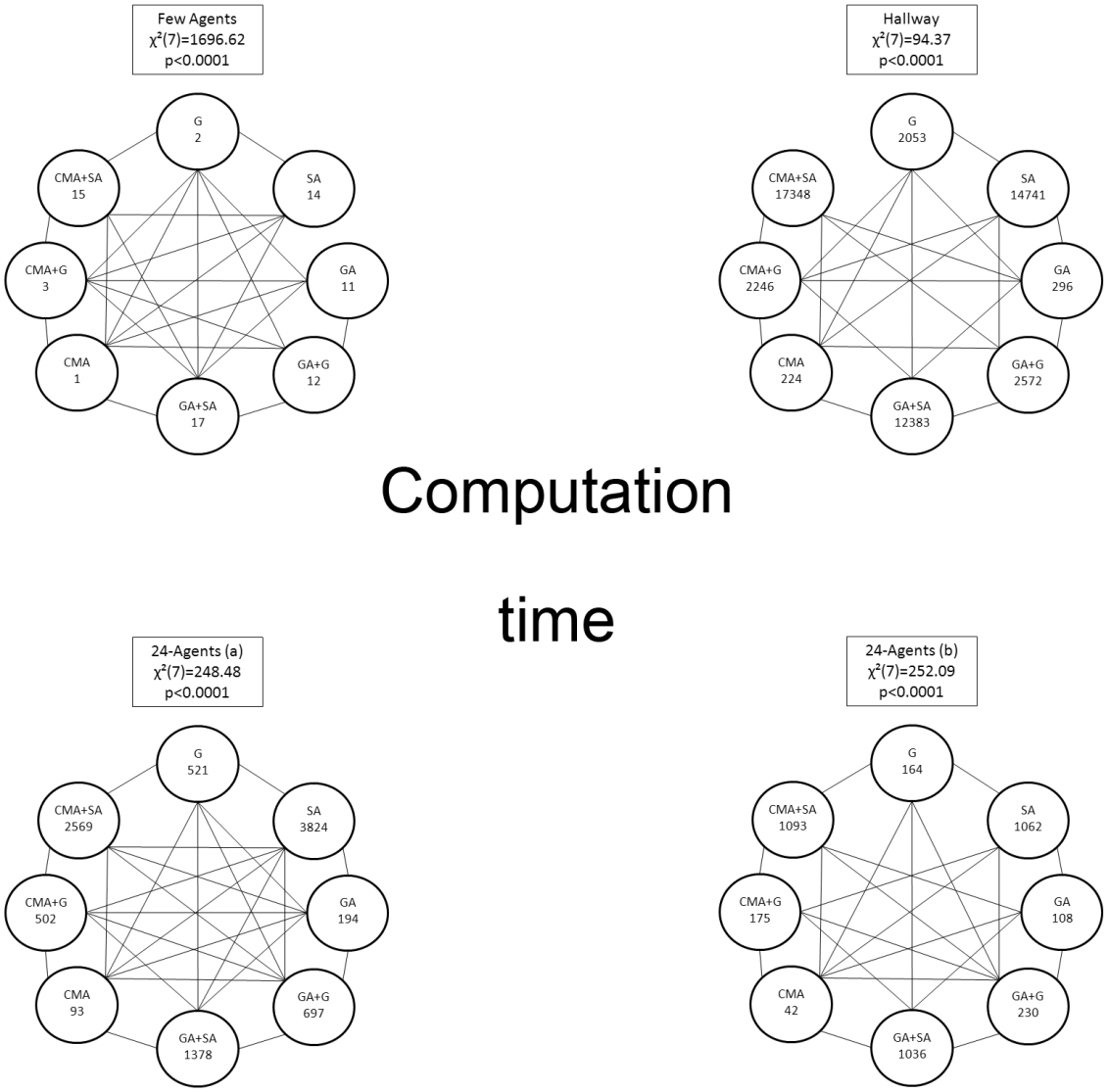


Figure 16: *Statistical results of the Friedman's Anova and Post-Hoc Wilcoxon signed rank tests on the computation time depending on the optimization algorithm. Mean values (s) are reported for each algorithm. Friedman's Anova showed an influence of the optimization algorithm on the computation time foreach type of data (Few Agents, Hallway, 24-Agents (a), 24-Agents (b)). Significant differences between algorithms are represented through a line ($p < 0.0018$).*

Crossing	{GA+G, GA+SA} > GA > SA > {G, CMA+SA} > CMA+G > CMA
Hallway	{SA, GA+SA, CMA+SA} > GA+G > {G, GA, CMA+G} > CMA
Circle-24 (P)	CMA+G > {G, GA+G} > GA+SA > {GA, SA, CMA+SA} > CMA
Circle-24 (I)	{CMA+G, GA+SA, CMA+SA} > GA+G > {G, GA, SA} > CMA

Table 1: *This shows which optimization algorithms most successfully optimize the metrics, the formulation $A > B$ means A optimizes better than B. In the Circle-24(a) example, we have used the Progressive Difference metric, while in the Circle-24(b) example, we have used the Inter-pedestrian distance metric, with the same data in both cases.*

Crossing	CMA < G < CMA+G < {SA, GA, GA+G} < CMA+SA < GA+SA
Hallway	{GA, CMA} < G < CMA+G < GA+G < GA+SA < SA, {CMA+SA}
Circle-24 (P)	CMA < GA < {G, CMA+G} < GA+G < GA+SA < CMA+SA < SA
Circle-24 (I)	CMA < {G, GA, CMA+G} < GA+SA < {SA, GA+SA, CMA+SA}

Table 2: *This shows which optimization algorithms most quickly optimize the metrics, the formulation $A < B$ means A is faster than B. In the Circle-24(a) example, we have used the Progressive Difference metric, while in the Circle-24(b) example, we have used the Inter-pedestrian Distance metric, with the same data in both cases.*

Parameter	**min**	**max**	**mean**	**std. dev.**
Boids model				
radius (m)	0.1	1	0.3	0.2
comfort speed ($m.s^{-1}$)	1	2	1.5	0.5
Helbing model				
radius (m)	0.1	1	0.3	0.2
comfort speed ($m.s^{-1}$)	1	2	1.5	0.5
RVO2 model				
comfort speed ($m.s^{-1}$)	1	2	1.5	0.5
neighbor distance (m)	10	20	15	5
radius (m)	0.2	0.8	0.5	0.25
agent time horizon (s)	0.1	5	2	2
obstacle time horizon (s)	0.1	5	2	2
Vision model				
a	0	1	0.5	0.2
b	0.5	1.5	1	0.2
c	1	2	1.5	0.2
comfort speed ($m.s^{-1}$)	1	2	1.5	0.5
Tangent model				
comfort speed ($m.s^{-1}$)	1	2	1.5	0.5
radius (m)	0.2	0.8	0.5	0.25
a	0	1	0.5	0.4
b	0	0.6	0.3	0.2

Table 3: *Default values for simulation parameters for the 5 models integrated to the framework*