Crowd simulation: realism assessment of interaction models

David Wolinski

ECOLE SUPERIEURE INGENIEUR RENNES
Master's Degree - Research in Computer Science
INRIA Rennes - MimeTIC
UNC at Chapel Hill, NC, USA - Gamma group

Research Internship Report

05/06/2012

Crowd simulation: realism assessment of interaction models

David WOLINSKI
Supervisor: Julien PETTRÉ

Contents

Introduction

We are used to interacting with other humans on a daily basis. These can vary from individual people to groups to large crowds and display behaviors on a wide range of scales. Crowd simulation, by modeling all these different interactions (though usually inter-pedestrian relations), aims to offer a better understanding of the mechanics behind these daily behaviors as well as reproduce the resulting, fascinating large-scale patterns.

There is, therefore, a number of domains where crowd simulation plays an important role. In movies, it is often easier and simpler to generate a crowd using a computer than ask hundreds of people to participate; which is even more true for animated films. The same applies to video games and, by extension, to virtual reality simulations in general, be it for entertainment or serious case studies. It is also of great help to be able to understand the behavior of crowds when designing buildings - e.g. to facilitate the flow of visitors at a convention or setting up evacuation scenarios. The common aspect to these applications, and all the ones that weren't mentioned, is the compromise that needs to be made between autonomy and control, large-scale realism versus local realism or performance versus quality (figure 1).

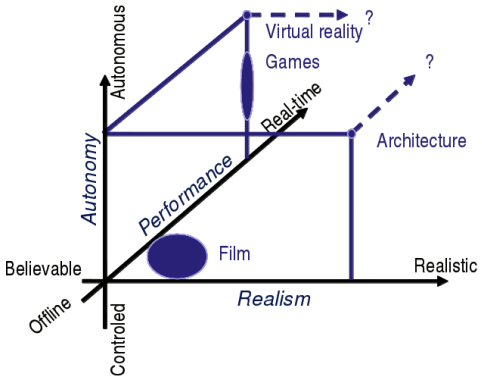


Figure 1: Possible applications can vary along these three main criteria [2].

In particular, among the various constraints crowd simulation is subjected to, the most important - and the most difficult to satisfy - is behavior realism. This constraint encompasses every aspect of crowds from the interactions between two pedestrians to the influence of groups of pedestrians to large scale behaviors such as spectators entering/leaving a stadium before/after a game. The challenge with realism at the small scale lies in the fact that we are used to seeing other people so much, thereby being able to immediately detect any defects. At the larger scale, on the other hand, there is an increasing number of influencing factors leading to extremely complex behaviors culminating in what can be observed during a panic outbreak.

As such, a vast number of varied approaches have emerged, based on different ideas; i.e. the construction of crowds from pair-interactions between two pedestrians or global crowd models as an analogy to fluid mechanics. The laws that govern these different models are dictated by conclusions we take from what we can observe in real life. However, these conclusions and principles can be erroneous. As mentioned earlier, we are used to seeing crowds very often and therefore trained to detect the most insignificant of flaws. Thus, to further improve the existing models for crowd simulation, it is necessary to be able to grade the realism of these models and therefore to define metrics and procedures to help us in this task.

Although simulating crowds has been much developed over the years, very little work has been conducted on their evaluation. The most advanced of which was done in [17] with SteerBench, part of SteerSuite. SteerBench is a framework which contains pre-set simulation scenarios as well as metrics to grade the models. The evaluation doesn't, however, rely on any data. Instead, the metrics represent different opinions the authors had on the properties of crowds such as pedstrians choosing the shortest path. These metrics, along with the lack of reliance on experimental data, don't allow objective comparisons of the models. Furthermore, they ignore the importance of the models' parameters which can greatly affect their performance. These drawbacks are then the motivation behind the work described in this report which proposes a new framework allowing to make objective, data-driven comparisons of crowd simulation models. To this end, my contributions were:

- setting up ways of using the data:
  - fully simulated data
  - partially simulated data to reduce the influence of unknown experimental conditions
- designing metrics:
  - Difference metric
  - Progressive Difference metric
  - Crossings metric
  - Vorticity metric
- integrating pre-existing metrics:
  - Path Difference metric
  - Entropy metric
- adapting optimization algorithms to the calibration problem:
  - Greedy algorithm
  - Simulated Annealing
  - Genetic Algorithm
  - combining them thanks to their properties
- implementing these ideas into a flexible, modular framework which can:
  - easily load experimental data
  - easily interface with crowd simulation models
  - calibrate and evaluate these models
  - visualize the data and simulation

- experimenting with the framework, thus acquiring preliminary results on:
  - the framework's validity as a tool
  - comparison of crowd simulation models

This report first gives a state-of-the-art of existing crowd simulation models and the little work that has been done to evaluate them. It then presents the mechanics behind the framework: ways to use the data, which models have been integrated, the metrics that have been designed and the algorithms that have been defined to calibrate the models. Finally, before concluding, the last chapter aims to describe the resulting framework as well as the preliminary results that have been obtained by using it.

State-of-the-art

1.1 Macroscopic point of view

1.1.1 Introduction

Macroscopic-based approaches aim to realistically simulate global crowd phenomena such as the formation of lanes when two groups cross ways, giving less emphasis to local phenomena such as collision-avoidance between two pedestrians. As such, the underlying models are often continuous with the discretization (crowd, space and time discretization) being introduced only while doing calculations. The following paragraphs will describe two representative approaches coming from the field of computer animation (more models have been developed in the fields of applied mathematics and statistical physics). The first of these two models is purely continuous and the other incorporates a basic notion of agents.

1.1.2 Continuum crowds

The described approach is found in the work of Treuille *et al.* [18] which is itself an extension of previous work by Hugues [7]. This method is inspired by fluid dynamics. Here, only large (medium-density) crowds with a common goal are concerned as opposed to crowds composed of individuals with differing objectives. The "individuals" that are displayed are incapable of decision-making and thus the method is a purely per-particle energy minimization without agent-based dynamics. It uses a combination of static fields which represent the influence of the environment (in the same way river currents carry boats) and dynamic fields which model the particles' effect (comparable to two boats trying to stay away from each other).

Unit cost Given a pedestrian at a certain position and with a given goal, that pedestrian will choose the path that minimizes the following function:

$$\alpha \int_P 1ds + \beta \int_P 1dt + \gamma \int_P gdt, \tag{1.1}$$

where the terms represent the pedestrians' criteria, that is the length, the duration and the discomfort of the path. After simplifying ($ds = fdt$, f being the speed), we get:

$$\int_P Cds, \quad where \quad C \equiv \frac{\alpha f + \beta + \gamma g}{f} \tag{1.2}$$

is the *unit cost field*.

Speed field The speed field is calculated based on the density of the crowd at a given place. The higher the density, the lower the speed and vice-versa.

Potential field The potential field ϕ is what governs the velocities of the particles. Since this potential is decreasing when approaching the goal, the particles will move according to $-\nabla\phi$. This potential field is calculated with dynamic programming akin to the Viterbi algorithm starting with the unit cost field.

The algorithm The algorithm used by the overall method is shown in table 1.

Algorithm 1: Overall algorithm.

```
foreach timestep do
    discretize crowd into groups and convert it to a density field
    foreach group do
        construct the unit cost field C
        construct the potential φ and its gradient ∇φ
        update particle positions
    end
    enforce minimum distance between particles
end
```

1.1.3 Continuous and agent-based mix

The following approach is taken from the work done by Narain *et al.* [11]. This approach is inspired by aggregate fluid dynamics composed of objects of finite size. While bearing similarities to Treuille's model, this one was developed to accomodate large crowds with varying degrees of density, effectively treating the crowd as a fluid which can be both compressible or incompressible. It also considers the individual goals of pedestrians thus introducing agent-based mechanics into the continuous model. Nevertheless its structure is similar to Treuille's, as seen in figure 1.1.

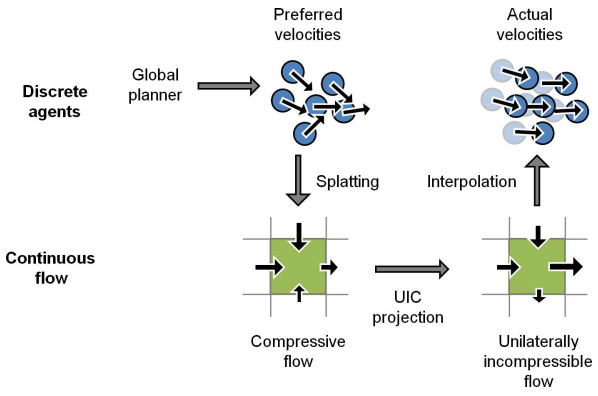


Figure 1.1: Overall algorithm [11].

There are two important components to this algorithm. The first (discrete) computes the preferred velocities based on a global planner. The second (continuous) enforces compressibility constraints onto the crowd.

Agent-based component The major difference with Treuille's method is that here the speeds correspond to the preferred speeds of the agents obtained with a separate global-planner. It is also possible to add more complexity at this step, enabling the system to interact with RVO [20, 19, 21] based crowds for example.

Continuous component The main goal of this step is to ensure that the crowd isn't overly compressed by the motion that would result from the preferred speeds of the agents. Therefore it adapts these speeds according to compressibility laws. The correct velocity v is thus the closest velocity to the preferred velocity v_{pref} which satisfies the equation of conservation of the number of agents:

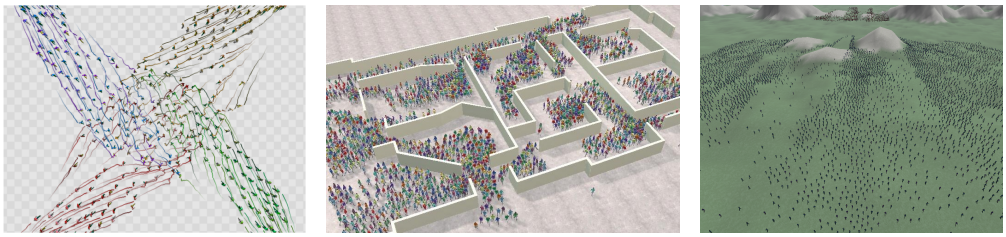
$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = 0 \tag{1.3}$$

where the density ρ is constrained by the minimum inter-agent distance d_{min}:

$$\rho \leq \rho_{max} = 2\alpha / \left(\sqrt{3} d_{min}^2\right) \tag{1.4}$$

1.1.4 Discussion

The main property of these methods is that the crowds are considered as a whole and the hypothesis of a common goal allows for good performance. Since the velocities are calculated mainly according to the terrain (first approach) or basically smoothed over a certain area (second approach), individual pedestrians' goals have, at best, a very limited impact on the final trajectories. This means that if we want more complex interactions such as two groups crossing ways, we need to include additional static fields, additional rules on dynamic fields and generally more complex algorithms. Moreover, the realism at the local scale is not entirely convincing as the models lack elaborate collision avoidance schemes, for example. Furthermore, the particles move equally forward/backward and side-ways as opposed to real pedestrians who prefer walking forward and turning when necessary. However, these methods give a good level of realism at the global scale, allowing to identify potential bottle-necks (office exits for example on figure 1.2b) and also allow the emergence of patterns (such as lanes when two or more crowds cross ways, figure 1.2a). Finally, these models scale very well and up to *1 million* pedestrians can be simulated on desktop PCs at 3 seconds per frame (or in the tens of thousands of pedestrians at interactive frame rates, figure 1.2c).



(a) (b) (c)

Figure 1.2: Different scenarios. (a) A vortex pattern emerges as four groups cross ways [18]. (b) The evacuation of an office [11]. (c) A large army chasing a smaller one [18].

1.2 Microscopic point of view

1.2.1 Introduction

Microscopic-based approaches define rules at the local scale and, through the progressive assembly of agents into crowds, expect them to result in global-scale behaviors. The following paragraphs will describe methods according to the level of agent sophistication.

1.2.2 Discrete representations

Discrete approaches are not the discrete approximations of continuous models. Where continuous models still need to be discretized during the computations, thus being approximated to the limit defined by the timestep, discrete models are as precise as possible "in the instant". Discrete models also have two other main advantages:

- Thanks to their probabilistic nature, it becomes possible to evaluate the plausibility of an outcome.
- The rules that govern the pedestrians are also quite simple thus allowing for scalable algorithms.

Cellular automata [9, 16] are modeled on grids where each cell contains a probability that the considered pedestrian will move there. The actions carried out by one pedestrian (one action can correspond to several transitions [9]) are executed sequentially during *rounds*. At a given round, all pedestrians execute their actions in parallel.

Local interactions Locally, a pedestrian can transition to one ore more cells within a certain neighborhood during the same round, which simulates the notion of speed [9]. Each pedestrian also contributes a presence factor to his neighborhood which repels other pedestrians.

Thus, at a given round, each pedestrian chooses his preferred *exit* (among cells in the neighborhood, as on figure 1.3), which is the cell he would attain during this round if only his preferred velocity and direction applied.

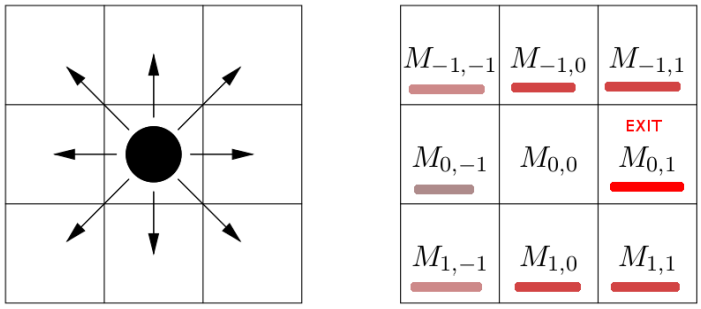


Figure 1.3: The matrix of preferred transitions for an agent [16].

He then transitions taking into account the presence factors of his neighbors. Two pedestrians cannot occupy the same cell, in which case the pedestrian with the highest relative probability is chosen to occupy the cell (figure 1.4). This solves extreme congestion cases where presence factors alone cannot prevent collisions.

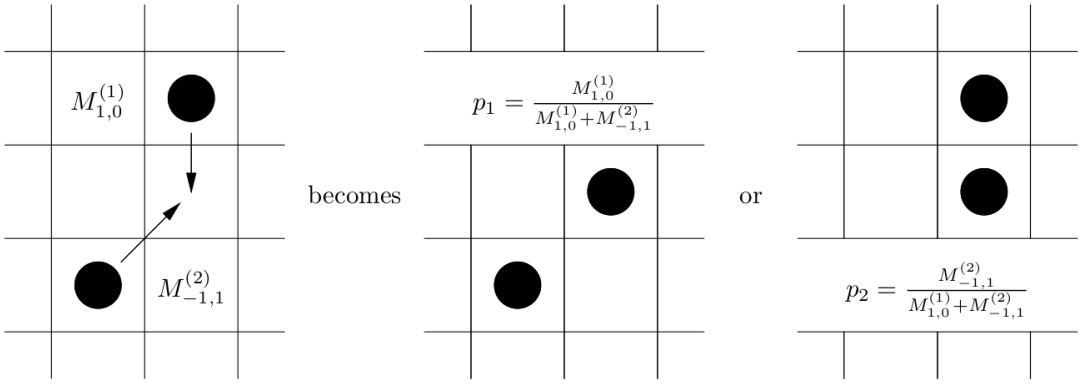


Figure 1.4: Conflict between two agents, solved by comparing relative probabilities [16].

Larger scale behaviors The simple rules that govern local interactions can also be expanded to model behaviors at a larger scale. It has been observed that when two groups cross ways, lanes tend to form and people follow the pedestrians that are in front of them. Also, people tend to follow trodden paths rather than take unexplored routes. These two phenomena can easily be incorporated into cellular automata by adding *floor fields*. The cells of these fields contain favorable transition probabilities if a pedestrian recently visited them [16]. These probabilites also "decay" over time to avoid locking scenarios where the crowds are reduced to a few lanes.

1.2.3 Position-based models

Position-based methods introduced the first non-discrete, microscopic models. These methods operate on crowds composed of basic pedestrians obeying physics-inspired laws. Helbing and Molnár's work [6] is often cited at this stage and works as follows.

Principle In their work, a pedestrian α is subjected to different forces $\overrightarrow{F}_\alpha$ that, when combined, define his velocity $\overrightarrow{v}_\alpha$ according to the equation:

$$\frac{d\overrightarrow{v}_\alpha}{dt} = \overrightarrow{F}_\alpha(t) + fluctuations \tag{1.5}$$

which is analogous to Newton's second law of motion.

The social forces The forces applied to a pedestrian can be both attractive, such as his destination, or repulsive, such as obstacles or strangers. However, it is also possible to introduce other social interactions such as the presence of friends or store fronts, which are modeled through attractive forces that fade over time (so the pedestrian can continue his journey). It is then possible to give the pedestrians some more decision-making abilities. All these forces, in Helbing's work, are functions of the pedestrians' positions and are directly/invertly proportional to the distances (depending on being attractive/repulsive), thus justifying the denomination of position-based approach.

1.2.4 Predictive/Velocity-based models

Predictive models are similar to what has been introduced with position-based methods. However, they process more information as the agents react to each other's trajectories. By predicting these trajectories, it then becomes possible to estimate which velocities are admissible and which would, otherwise, lead to collisions. These are the models that lead to the most realistic local behaviors while also being able to produce good global behaviors.

Prediction models The various prediction models that can be found in the literature establish admissible velocity domains. These are computed in the relative position space where one agent's motion is determined relative to the reference agent.

One of the first proposed predictive models was the one introduced by Paris *et al.* in [13]. In this model, the near future time windows were discretized and each of the obtained intervals lead to an angular section in which the possible positions of the other pedestrians were reported, thus forming, by elimination, admissible velocity domains (figure 1.5).

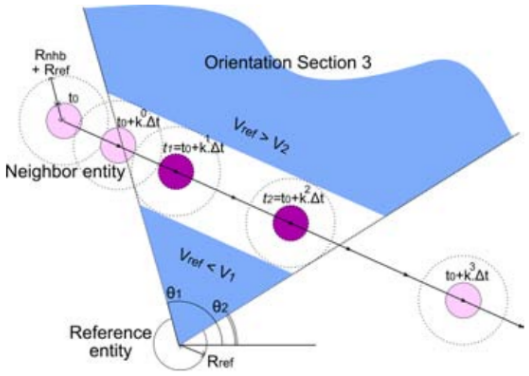


Figure 1.5: An angular section with possible neighbor pedestrians' positions where admissible velocities should not be able to move the reference robot [13].

One example can be found in Van den Berg's works [20, 19] which expanded the notions of *Velocity Obstacles (VO) and Reciprocal Velocity Obstacles (RVO)* and introduced the notion of *Optimal Reciprocal Collision Avoidance (ORCA)*.

Let τ be the following time period, A and B two robots with radii r_A and r_B, positions $\mathbf{p}_A$ and $\mathbf{p}_B$ and possible speed sets V_A and V_B. The velocity obstacle $VO^{\tau}_{A|B}$ induced by B over A during τ is defined as:

$$VO^{\tau}_{A|B} = \{\mathbf{v} | \exists t \in [0, \tau] :: t\mathbf{v} \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)\} \qquad figure\ 1.6b \tag{1.6}$$

The set of admissible velocities for A while B chooses a velocity from V_B is:

$$CA^{\tau}_{A|B}(V_B) = \{\mathbf{v} | \mathbf{v} \notin VO^{\tau}_{A|B} \oplus V_B\} \qquad figure\ 1.6c, \tag{1.7}$$

Here, $\oplus$ is the Minkowski sum which actually is a limitation as it works well only for pedestrians that are centrally symmetrical or without any rotation movement. The ORCA sets are then defined such that $CA^{\tau}_{A|B}(ORCA^{\tau}_{B|A}) = ORCA^{\tau}_{A|B}$ and vice-versa. These sets also contain the most velocities that are close to A and B's preferred velocities $\mathbf{v}^{opt}_A$ and $\mathbf{v}^{opt}_B$. If the robots A and B choose velocities that are inside $ORCA^{\tau}_{A|B}$ and $ORCA^{\tau}_{B|A}$, they are assured to be collision-free for at least τ time.

This model can be further expanded with the notion of *Acceleration Velocity Obstacle (AVO)* [21] where the walkers' acceleration constraints are taken into account. This leads to changes on the admissible velocity domains as seen on figure 1.6d.

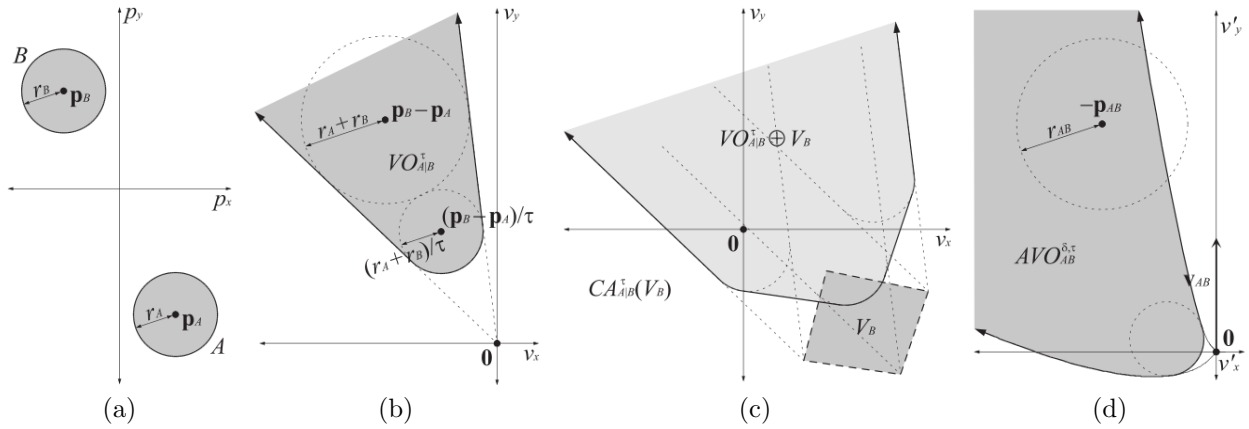
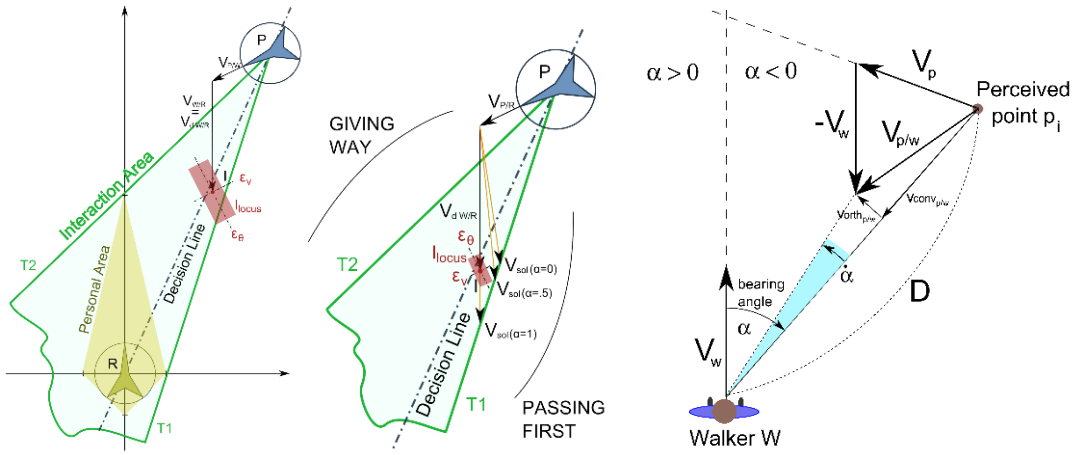


Figure 1.6: (a) A possible configuration [19]. (b) the resulting velocity obstacle $VO^{\tau}_{A|B}$ for $\tau = 2$ [19]. (c) The resulting collision-avoiding set $CA^{\tau}_{A|B}(V_B)$ [19]. (d) Acceleration velocity obstacle as opposed to the velocity obstacle from (b) [21].

Additional features Other examples of predictive models can also be found which include additional features such as the asymmetry of the personal space around a pedestrian and the resulting effort division [14]. In such a division, the pedestrian that gives way makes more effort to change his trajectory than the pedestrian passing first does, just as in real life. This is determined by the *Interaction Point* (I_{locus} on figure 1.7a). According to its position relative to the decision line, the walker will give way or pass first; the admissible velocities (outside the interaction area) also introduce greater changes if the walker gives way.

Finally, another approach aims to recreate the perception of the scene as witnessed by the pedestrian [12]. Here, the reactions are functions of three parameters that can be directly extracted from the walker's visual flow: the bearing angle α (angle under which incoming obstacles are seen), its derivative $\dot{\alpha}$ and the time to interaction tti (estimated from the rate of growth of the obstacles); which can be seen on figure 1.7b. The focus is on the perception-reaction loop rather than the kinematic aspect.



(a) (b)

Figure 1.7: (a) Asymmetrical model, passing first or giving way and solution velocity according to the position of I_{locus} relative to the decision line [14]. (b) The inputs of the $\{\alpha, \dot{\alpha}, tti\}$ model [12].

1.2.5 Discussion

The first remarks concern the features that are modeled in the different methods as there isn't a single one that combines all aspects. For example, except for Helbing's work, various social factors are not implemented (e.g. small groups of pedestrians, places of interest such as store fronts etc...). These social factors are only beginning to appear in the literature with very recent publications [10, 8] (small groups of people) and [5] (personality traits). Also, there aren't always explicit calibration possibilities, with the notable exception of Paris' model (where unified calibration parameters could not, however, be found). Furthermore, Pettré's model is the only one which is explicitly asymmetrical, that is with a clear distinction (and determination method) between the pedestrians giving way or passing first and the associated respective efforts. Finally, Ondrej's vision-based model best simulates the waker's perception instead of using oracle-like knowledge of the neighboring pedestrians' motions.

Behavior-wise, the various models more or less successfully lead to emergent self-organized patterns, that is lanes (figures 1.8c, 1.8b) and vortices (figure 1.8a) in various cross-ways scenarios or movement oscillations between the congested sides of a gate (figure 1.8d). There are also various artifacts that can be observed such as *overshoot* where some agents can arrive with overly aggressive velocities or agents that are entirely carried by the flow of a dense crowd [20]. Additionally, there are cases where some odd interactions can be observed such as permanent or temporary locking scenarios where two agents that arrive strictly head-on simply stop [6] or perform an overly long *reciprocal dance* [20] (where the two pedestrians trying to avoid each other both choose the same side several times in a row). Finally,

considering raw performance (number of people at interactive frame-rates), it can be noted that the newer methods scale reasonably well - close to linearly - with the discrete approaches having a definite edge since there map-based evaluation allows for $\mathcal{O}(n)$ complexity.

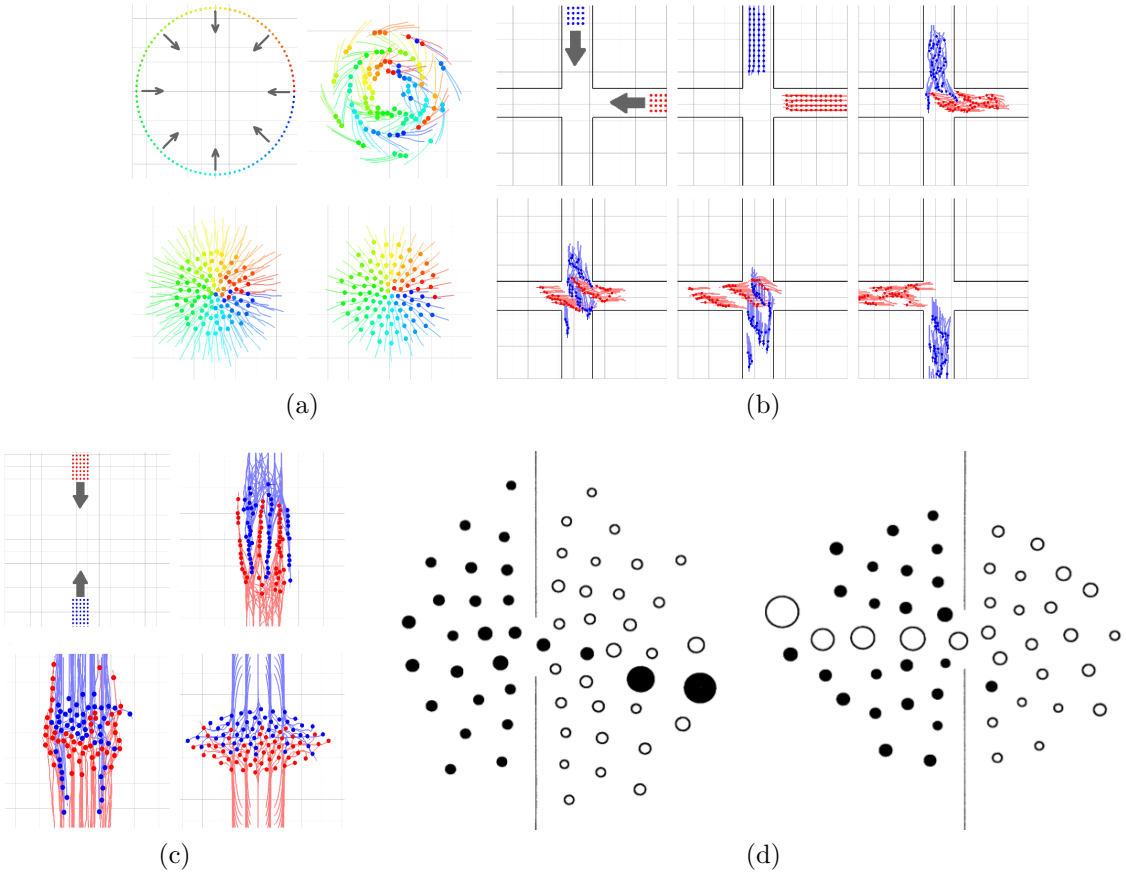


Figure 1.8: (a) & (c) Vortex and lane formation, top left: initial setup, top right: Ondrej, bottom left: RVO, bottom right: Helbing [12]. (b) 45° lines at a road junction (Ondrej)[12]. (d) Two phases of an oscillating crowd movement direction at a gate (Helbing) [6].

1.3 Realism assessment

Compared to the affluence of crowd and behavior models that have emerged, relatively little work has been conducted on the assessment of the realism achieved by these methods.

Some papers that presented simulation models, however, additionally introduced interesting real-life experiments to model and calibrate behaviors that can also be used to grade the realism. Two good examples of this can be found in Pettré's [14] and Lemercier's [10] work. Here, different scenarios were set up where pedestrians interacted in various ways. All of which were carried out in controlled environments and under strict protocol. Several

aspects of the situations that followed were measured:

- motion capture allowed to plot trajectories,
- key moments' (moment when the pedestrians notice each other, moment when they started adapting their trajectories...) times were recorded [14],
- possible estimations were recorded (*minimum predicted distance* to interaction) [14]
- speeds were also recorded, modeling behavior in *following* situations [10].

Several conclusions could be made from the results:

- both pedestrians (two converging pedestrians) adapt their trajectories [14],
- the pedestrian passing first makes less effort than the one giving way [14],
- when the pedestrians notice each other at a late moment (minimum predicted distance close to zero), the observation period becomes longer and the reaction is delayed leading to close-to-collision cases [14],
- stop-and-go motions can be observed in following scenarios [10].

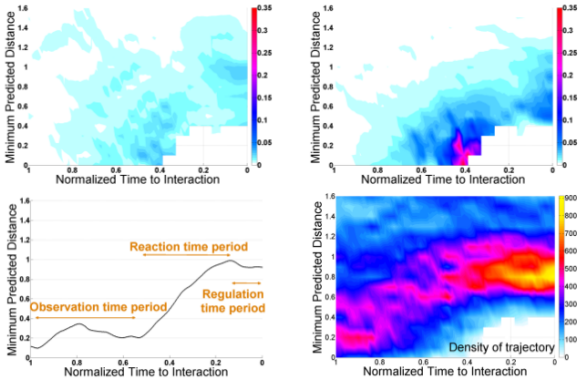


Figure 1.9: Graphs showing the measures made during the experiments [14]. Top: efforts made by the pedestrian passing first (left) and the one giving way (right); bottom: minimum predicted distance with respect to the normalized time to interaction showing the different phases for one experiment (left) and the density over all experiments (right).

Some work, however, has been done specifically in the field of realism assessment. In their 2008 paper, Singh *et al.* [17] propose a framework to grade steering models. In this approach, three criteria have been devised to describe the effectiveness of a grading framework:

- test cases used for comparison should be representative of a range of real-life scenarios,

- the specific aspects of the tested steering approach should not interfere with the evaluation,
- there should be some flexibility to the evaluation to enable the user to focus on certain scenarios.

The actual framework, uses several metrics:

- number of collisions per agent, and the average number of frames spent in collision per agent,
- time and effort efficiency; respectively the time for an agent to reach his destination and the accelerations he had to apply,
- various other secondary metrics such as speed, angular velocity, changes in speed...

The procedure then becomes: selecting scenarios (a set of scenarios has been included in the framework), computing metrics on these situations and finally giving a global benchmark grade.

Finally, there is a project named *Pedigree* [2] which has started and which includes a campain of data collecting similar to what has been done in [14, 10] except on a bigger scale.

Figure 1.10: An experiment for the Pedigree project [2].

Evaluation of crowd simulation models in theory

As stated earlier, there are four core aspects to the problem of evaluating crowd simulation models. These include the data that is being used for comparison, the model that is being evaluated, the metric which grades one or more aspects of the simulation and then the evaluation/calibration process which gives the final verdict. This section aims to describe in detail how each of these components is dealt with.

2.1 The data

The framework's goal is to offer a data-driven evaluation of the realism of crowd simulation methods. As such, the data that is used is of primary concern. It must allow the results to be statistically relevant, thus requiring various reruns of the same scenario. The working conditions under which the experiments are run should also be known and stable enough to be satisfiable independently of the tested model. Finally, the data should be "representative", which is still very difficult to define. And of course, the measured values should be as precise as possible.

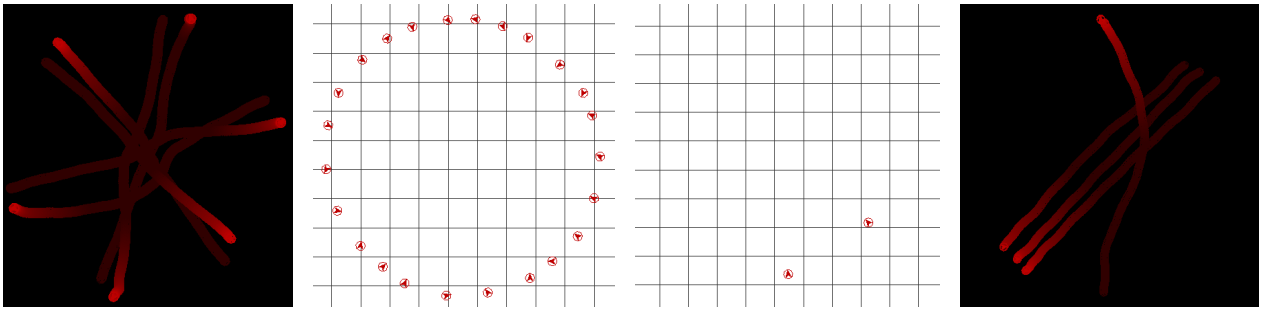
The data that is considered here comes from two sources: the ANR-Pedigree project [2] and "Seyfried data" [3].

2.1.1 ANR-Pedigree data

In the scope of the ANR-Pedigree project, people were gathered and fit with body markers which were then tracked by multiple cameras during various scenarios.

The data that was available came in two sorts:

- The first series of experiments consists of 6 to 24 pedestrians positioned around a circle and who were asked to reach the opposite side of the same circle (Fig. 2.1a and 2.1b).
- The second series consists of 2 to 4 pedestrians crossing each other (Fig. 2.1c and 2.1d).



(a) (b) (c) (d)

Figure 2.1: (a) paths of 6 pedestrians initially in a circle, brighter colors indicate more recent positions. (b) initial positions of 24 pedestrians in a circle. (c) 2 pedestrians crossing ways. (d) paths of 4 pedestrians crossing ways.

It is interesting to note that the resulting data offers precise position information about each pedestrian and that the conditions stay the same throughout the experiments; most importantly, all involved pedestrians are tracked at all times.

2.1.2 Seyfried data

The data which was used consists of pedestrians moving in a corridor and filmed by two cameras (Fig. 2.2a and 2.2b). The acquired data then represents the trajectories of the pedestrians passing in the cameras' fields of vision (Fig. 2.2c)).

(a) (b) (c)

Figure 2.2: (a) left camera view. (b) right camera view. (c) associated paths.

The main difference with the data from ANR-Pedigree lies in the varying number of tracked pedestrians. In this case, pedestrians keep appearing and disappearing at the two ends of the corridor (the edges of the cameras' fields of vision). The fact that all pedestrians are not tracked at all times introduces uncertainty about the state of the crowd. In this case, the experiments' conditions are not stable.

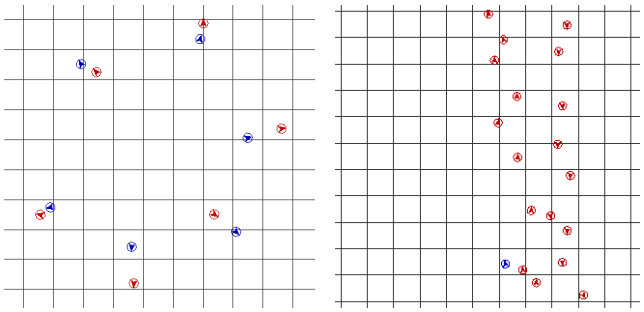
2.1.3 Using the data

The data is used by associating a simulated agent to each of the pedestrians tracked during the experiments. In the case of data where all parameters are known, it is easy to replicate the initial and final conditions - i.e. pedstrians' starting positions and velocities and their goals. All that remains to be done is then to run the simulation (Fig. 2.3a).

For data with varying conditions on the other hand (Seyfried data), it is not so easy. With the uncertainty that is introduced about the state of the crowd, it becomes difficult to justify creating agents with the same position and velocity as the corresponding pedestrians. Over the course of a scenario, the observed and simulated crowds' states will differ. Since part of it is unknown, specifically the state of the part of the crowd encountered by a pedestrian who is about to enter the cameras' fields of vision, it is unjustified to create the corresponding agent with the same conditions. It would be equivalent to arbitrarily "resetting" some agents while dealing with data from the ANR-Pedigree project.

A solution would be to find chunks of data where no pedestrians appear nor disappear. This would then result in a "self-contained" system where all conditions are known and stable. However, if the number and density of agents in the scenario increase, the size of these chunks decreases, and eventually the probability to find non single-frame chunks becomes 0.

The final solution which was adopted is an adaptation of this previous partial solution. To increase the size of these chunks of "self-contained" data, not all pedestrians are simulated. By decreasing the number of pedestrians that must be tracked from beginning to end, it becomes possible to find "long" chunks. In this case, "long" means that the chunks' duration approaches the time it takes for a single pedestrian to reach his goal as is the case for ANR-Pedigree data. The remaining pedestrians that are present in this chunk are simply treated as moving obstacles. As a result, none of the simulated pedestrians are subject to unknown data (Fig. 2.3b).



(a) (b)

Figure 2.3: (a) fully simulated crowd. (b) partially simulated crowd. Simulated agents in blue.

Furthermore, simulating only part of a crowd can help when the scenario includes a large number of people, thus alleviating scaling issues, especially during the evaluation/calibration process (see section 2.4).

2.2 The models

The second core aspect is the models that we want to evaluate. A state-of-the-art of the prominent models has been given in chapter 1. Five models have been used thus far in the limits of their implementation:

- the Boids model [15], unofficial
- the Social Forces model [6], unofficial
- the Tangent model [14], INRIA internal development version
- the Vision model [12], INRIA internal development version, CPU-based
- the RVO2 model [19], official library, version 2.0.1

2.3 The metrics

Metrics are the third core aspect of the framework. They allow to effectively grade the models. However, contrary to what is done in the SteerBench framework, they grade the models with respect to the experimental data. Being comparative metrics, the score obtained by using them should be lower when a model's output is similar to the data, approaching zero in the case of an ideal model.

As the availbale data only contains path information, these metrics are mostly low-level, basing their output on the pedestrians' positions and velocities. It is still possible, however, to design metrics which focus on different aspects of these trajectories, with the idea of being able to combine them.

A few metrics are currently in the framework which analyze different properties of the paths; they are listed below.

2.3.1 Path length metric

The path length is a very simple metric which outputs the difference between a real pedestrian's total path and the associated simulated agent. This metric can be considered as an "efficiency" metric since a real pedestrian is able to pick a near-optimal path to reach a goal. As such, this metric is part of the "efficiency" category of metrics proposed in SteerBench (although in a non-comparative version).

2.3.2 Crossings metric

One very local aspect of pedestrian interaction which has been well studied is how pairs of pedestrians cross ways. For example, in [14] it has been shown that the pedestrian passing first performs a relatively small adaptation to his movement, which is also mainly a speed increase. On the other hand, the pedestrian giving way makes more important changes, mainly adjusting his orientation. The crossings metric is a simple adaptation of this notion, which reports in what order pedestrians pass each other. In a future version of this metric, it should be interesting to also record what adjustments are made to the velocities of both agents.

2.3.3 Vorticity metric

It has been observed (for instance in [12]) that several patterns emerge in crowd interactions. These can include the well-known formation of lanes when two groups cross ways, oscillating passage of pedestrians at gates or vortices. This last phenomenon can be observed when a large number of people converge in a certain area; it allows an efficient circulation as speeds are minimally reduced. This inspired the vorticity metric which computes the vorticity as defined in fluid dynamics:

$$\overrightarrow{\omega} = \overrightarrow{rot}(\overrightarrow{v}), \text{ where } \overrightarrow{v} \text{ is the velocity field.} \tag{2.1}$$

2.3.4 Difference metric

While the previous metrics all focus on particular aspects of the pedestrians' trajectories, the difference metric tries to evaluate the paths as a whole. It basically computes the difference between the paths of the real and simulated pedestrians. It is the sum of the differences between their positions at each timestep. It is both sensitive to the difference of the trajectories themselves and the progress made by the agents along these trajectories.

2.3.5 Progressive difference metric

While advancing in the data/simulation, a model continuously makes errors compared to the data. These errors can compensate for each other to a degree but are also very likely to stack, especially when the scenario is long. In such a case, the simulation's and data's states just keep on diverging. It thus becomes less and less justifiable to use the difference metric to compare these results, which motivated the progressive difference metric. This metric, instead of comparing the paths as a whole, considers these paths as a series of situations to which pedestrians must react. At each timestep, the simulated crowd is set to the same state as the real crowd, the decisions made by each pedestrian are then defined as their new velocity. The score given by this metric is then the sum of all these velocity differences for each timestep.

2.3.6 Entropy metric

The entropy metric was developed by Stephen Guy [4]. This metric works in much the same way as the progressive difference metric, setting the simulated crowd to the same state as the real crowd at each timestep. However, where the progressive difference metric sums the errors made by pedestrians, the entropy metric aims to estimate the size of the distribution of these errors while taking into account the inaccuracy of the experimental data.

The only current challenge with this metric is that it is susceptible to the timestep and scale (which can be chosen by a user) of the experimental data - its output can thus be variable even if the data itself does not change. Furthermore, the error estimation in this case is not based on the pedestrians' chosen velocities but instead on their next positions. This makes it further susceptible to the timestep and scale of the experiments. In the case of the data currently available, this susceptibility translates into NaN (Not a Number) scores. One solution to this is currently to change from using positions to velocities, which is equivalent to normalizing the timestep to 1s (the choice to use the positions was due to the available data which doesn't necessarily provide accurate velocity information). Another solution would be to change the way the final size is computed from the distributions. This would be done by using the sum instead of the product of the eigenvalues of covariance matrices to reduce the impact of flat distributions (where one eigenvalue can be close to zero).

2.4 Evaluation - Calibration

Finally, the last core aspect is how the evaluation itself is done. Ideally, this evaluation should be fair, and to this end, several dependences need to be taken care of. In section 2.1.3, we have dealt with challenges linked to the kind of data that can be used. In this section, we will deal with the two remaining dependences: metric-dependence and model-dependence.

2.4.1 Metric-dependence

The result of the evaluation of crowd simulation models is case-dependent and metric-dependent (it depends on the experimental scenario, conditions... and a model might have better results when considering a certain aspect while another model might outdo it in another area). Moreover, the behavior of the metrics themselves might vary. Some metrics, such as the crossings metric, revolve around only part of the simulation (in this case, the order of passage between pedestrians). It might then be relevant to combine them with other metrics that "take care" of the rest - for example the difference metric. However, all these metrics, including the ones that aren't in the framework yet, or even user-proposed ones, have potentially very different value ranges and progressions. Some metrics might give "absolute" scores stating how good the simulation is and which will stay relatively constant accross test-cases whereas others might give scores which are only valid when compared. For

instance, the entropy metric's output depends not only on the experiment but also on the timestep of the simulation as well as its scale.

It then becomes necessary to "normalize" these results. This is done by using a reference "dummy" model where the sole purpose of the agents is to reach their goals in a direct path and with constant speed, without taking into account any possible collisions. As a result, when a model is evaluated, the score it obtains is not a direct grade but an indication of how much better or worse it is when compared with the dummy model. An interesting observation is that in SteerBench this sort of dummy model does extremely well in the "efficiency" category of metrics.

2.4.2 Model-dependence

The evaluation is not only data-dependent and metric-dependent but also model-dependent in the sense that any given model that has parameters will perform in different ways based on the values of said parameters. A generic set of parameters will not enable a model to perform well in every case. To be able to compare models in a fair way, they need to be calibrated. Through calibration, we should be able to find the parameters that allow a model to perform best with respect to a given test-case and a given metric.

The calibration process becomes an optimization problem where we aim to minimize the cost function - the data/model difference metric with respect to the data - over the search space composed of the pedestrians' parameters.

$$\min_{p \in P} m(D, M(p)) \tag{2.2}$$

where m is the metric, D the data and M the model with parameters p. The search space P is of dimension:

$$\dim(P) = nb_{ped}.nb_{param} \tag{2.3}$$

It is also assumed that the pedestrians' parameters follow certain - mostly Gaussian - distributions specified by the user (referred to as base distributions). For example, if a model includes a preferred speed parameter, the user can specify that this parameter follows a Gaussian distribution of mean 1.4 $m.s^{-1}$ and of standard deviation 0.3 $m.s^{-1}$. These distributions may also need to be clamped, for instance, if a model represents agents as circles with a certain radius, it is justified to consider it as a parameter as it may also represent personal space; however, its value may be required to not be lower than a certain threshold to prevent collisions.

$$X \in P,\ X_{i,j} \sim \mathcal{N}(\mu, \sigma) \cap [\alpha_j, \beta_j] \tag{2.4}$$

where i and j respectively denote a certain pedestrian and parameter.

This minimization can currently be achieved through three algorithms: a greedy algorithm, simulated annealing and a genetic algorithm.

Simulated annealing and greedy algorithm

The simulated annealing algorithm is a very popular optimization algorithm which generally performs very well.

To use this algorithm within a particular problem, three aspects need to be defined: the procedure to find a neighbor state, the probability to move to this neighbor state and the temperature (pseudo-code is shown in 2).

neighbor states: pick a new parameter value at random according to the parameter's base distribution - as previously mentioned, generally Gaussian $\mathcal{N}(\mu_{param}, \sigma_{param})$.

move probability: classic, 1 if $e_{new} < e_{old}$, $exp(\frac{e_{old}-e_{new}}{T})$ otherwise.

temperature: $\frac{K-k}{K}$, where k is the number of iterations without any improvement and K the allowed number of such iterations.

Algorithm 2: Simulated annealing.

```
k ← 0                                              // initialize loop counter
while k < K do
    T ← temperature(k, K)                          // compute temperature
    s_new ← neighborState(s)                       // try new, neighbor state
    e_new ← cost(s)                                // compute energy
    if move(e, e_new, T) then                      // should we move to the new state?
        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
```

The greedy algorithm is then obtained by setting K to 1 and move() to 0.

Genetic algorithm

It is worth noticing that these Simulated Annealing and Greedy algorithms "blindly" use values based on the parameters' base distributions. But it should also be noted that the search space presents some interesting properties by having quasi-repeating dimensions: the same parameter accross all pedestrians, although having different values, follows the same base distribution. However, the real distribution - the distribution of the optimal parameters - is case-dependent. By maintaining a pool of solutions where the subset of best solutions contains similar solutions, it may be possible to approach these real distributions.

There are a few optimization algorithms which maintain a pool of solutions. The most popular ones include Ant Colony Optimization, Particle Swarm Optimization and the Genetic Algorithm. This last one is an optimization algorithm which uses a population of multiple individuals (potential solutions to the problem) and creates new populations (a new generation) by changing the previous individuals through recombination (crossovers) and arbitrary changes (mutations). These two methods of generating new individuals for new generations are particularly adapted for the calibration problem since they can be made to use parameter distributions computed on the best individuals - i.e. the distributions which will hopefully converge towards the real distributions.

The Genetic Algorithm here functions in the following way:

Algorithm 3: Genetic algorithm.

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

To adapt it to a particular problem, four operations need to be defined: intialization, selection, reproduction and termination.

initialization: each parameter's value is picked at random from the base distribution.

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 K successive loop iterations did not lead to a new optimum.

reproduction: based on which group he belongs to, an individual 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).

The pseudo-code for the reproduction process can be found in 4 and an example can be found in appendix 1.

Algorithm 4: Reproduction

```
foreach individual i do
    b_i := rand(B)                                  // assign best indiv to current indiv
    foreach parameter p ∈ i do
        if prob a then                                                  // change parameter
            if prob b then                                                      // crossover
                p := p_rand(B)
            else                                                                // mutate
                if prob c then                  // based on current real distribution
                    p := rand(distrib(b_i))
                else                                        // based on base distribution
                    p := rand(N(μ_param, σ_param))
                end
            end
        end
    end
end
```

Combining algorithms - properties

It is interesting to note that, contrary to the Simulated Annealing and Greedy algorithms which simply follow base distributions, this implementation of the Genetic Algorithm builds a coherent set of parameters for the agents. These opposing properties can benefit the overall calibration process by allowing combinations of algorithms, the results of which are presented and discussed in section 3.2.2.

Evaluation of crowd simulation models in practice

To evaluate crowd simulation models, a framework was developed. It is composed of several modules which reflect the components described in section 2. This section will first globally describe the framework itself and then present and discuss some preliminary results.

3.1 Presentation of the framework

The framework covers the four main aspects listed in the previous chapter. It is capable of loading various experimental data, interfacing with models' implementations and using different metrics to simultaneously evaluate and calibrate the models. It has a modular structure, thereby allowing the user to use what is best suited for a particular task.

3.1.1 Modules

This framework is geared towards a user who possesses data and a model and who wants to:

- calibrate the model with respect to the available data and a given metric
- and/or evaluate the model, calibrated or not

The situation can be summed up in the following way:

- inputs:
  - presence of data with the following properties:
    - one file represents a record of a crowd scenario
    - known number of pedestrians
    - consists of successive timestamped frames (points in time)
    - for each frame, each pedestrian's position and velocity are known
  - presence of an implemented model
  - choice of which pedestrians to simulate
  - choice of a framework's metric, or knowledge on how to implement one
  - choice of a calibration algorithm
- wanted outputs:
  - model's score based on the metric

– model calibrated with respect to the data and metric
– statistics on the model's parameters (real distributions)
– model's score after calibration
– visualization of the real and simulated crowds

The inputs and outputs from this summary are taken care of through different modules. The global interactions between these modules are shown in figure 3.1.

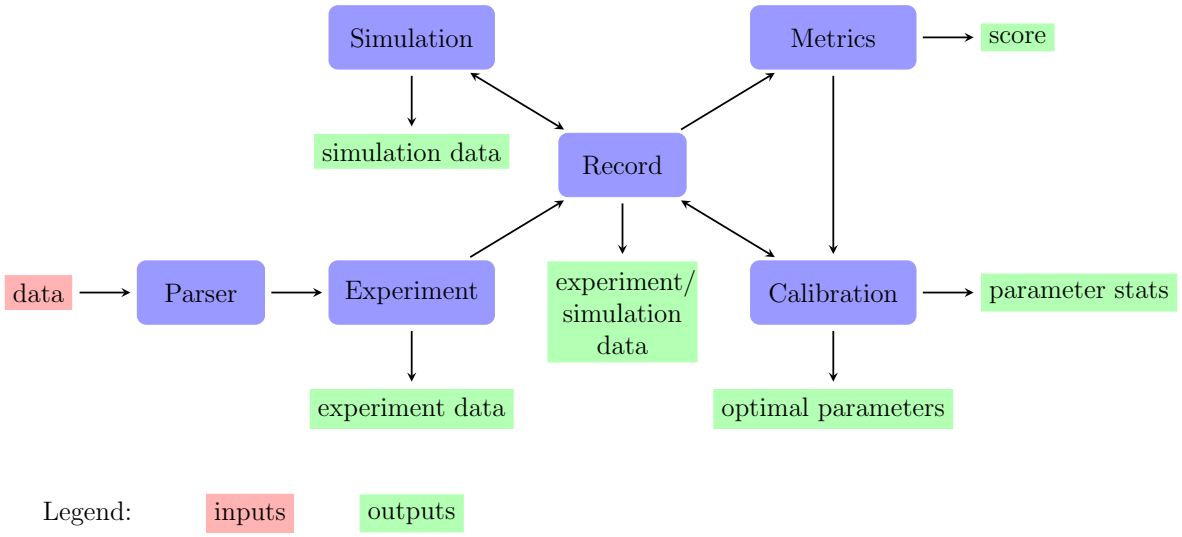


Figure 3.1: Diagram of the framework's module interactions, arrows represent data flow.

The different modules are described individually in the following paragraphs and the accompanying listing 3.1 shows sample code for a program which:

- loads data from the file *cercle6.ana*
- uses the RVO2 (see section 2.2) model
- uses the Difference metric (see section 2.3.6)
- gets the raw metric score
- calibrates the model using the Genetic algorithm (see section 2.4)
- gets the raw metric score after calibration
- displays statistics on the model's parameters (after calibration)

Finally, listing 3.2 shows this program's output.

Experiment module The *Experiment* module is composed of the *Experiment* class as well as a parser class (see appendix 2.1 for more information on the parser). The parser reads information from a data file and transmits it to the *Experiment* class. This latter class can be used to acces the experimental data (number of pedestrians, number of frames, pedestrians' positions, velocities, timestamps...). It is used as shown in steps 0 and 1 in listing 3.1.

Simulation module The *Simulation* module consists of the *Simulation* class. To interface a model, a new class has to be created and must inherit from the *Simulation* class (more information can be found in appendix 2.2). The resulting class (*SimulationRVO2* in step 2, listing 3.1) can then be used to compute the successive states of the simulation as well as access the simulation's data (number of pedestrians, number of frames, pedestrians' positions, velocities, timestamps...).

Record module The *Record* module consists of the *Record* class and is used to synchronize the *Experiment* and *Simulation* modules: initialize the simulation to mirror the experimental setup (number of pedestrians, starting positions and velocities, timesteps, goals...) as well as manage which pedestrians are simulated. It is used in step 3 in listing 3.1 and further information can be found in appendix 2.3.

Metrics module The *Metrics* module is essentially a namespace which regroups all the metrics on a [one metric $\Leftrightarrow$ one function] basis. A metric function is then called with the *Record* object as a parameter and returns a score in the form of a float value. For instance, in steps 4 and 6 of listing 3.1, the Difference metric is used for evaluation (note that in this case, only the raw value of the metric is returned, for comparison with the Dummy model, one would need to perform the same steps using the Dummy model). Further information concerning the definition of a metric can be found in appendix 2.4.

Calibration module

The *Calibration* module regroups the calibration algorithms and can return statistics on the model's parameters. It is first initialized with the *Record* object, the model can then be calibrated by calling the appropriate calibration algorithm and passing the metric function as a parameter (listing 3.1, step 5). The calibration can show statistics on the model's parameters by calling the *printParams()* function (listing 3.1, step 7). It shows the score (if the model has just been calibrated) and the distribution (Gaussian) of each parameter by printing the mean and standard deviation (see listing 3.2). More information on the calibration algorithm methods can be found in appendix 2.5.

Listing 3.1: Example program.

```
#include "CaLib.h"
#include "SimulationRVO2"

int main()
{
        // Step 0: Load data from cercle6.ana
        ParserANA parser;
        parser.parse("cercle6.ana");

        // Step 1: Initialize Experiment object with data
        Experiment experiment;
        experiment.init(parser);
```

```
        // Step 2: Create Simulation object
        SimulationRVO2 simulation;

        // Step 3: Create Record object and synchronize Experiment and Simulation
        Record record;
        record.setExperiment(&experiment);
        record.setSimulation(&simulation);
        record.initSimulation();

        // Step 4: Get and display raw score from Difference metric
        float score = Metrics::differenceMetric(&record);
        std::cout<< "Score: "<< score<< std::endl;

        // Step 5: Calibrate with Genetic algorithm with respect to Difference metric
        Calibration calibration(&record);
        calibration.calibrateGenetic(Metrics::differenceMetric);

        // Step 6: Get and display raw score with optimized parameters
        float betterScore = Metrics::differenceMetric(&record);
        std::cout<< "Score after calibration: "<< betterScore<< std::endl;

        // Step 7: Display statistics on the parameters
        calibration.printParams();

        return 0;
}
```

Listing 3.2: Example program's output.

```
Score: 2633.25
Score after calibration: 431.843
#--BEST PARAMETERS--

%cost
    431.843
%parameter_0
    1.68059 0.280732
%parameter_1
    16.4192 1.69128
%parameter_2
    0.262327 0.0948359
%parameter_3
    4.67494 1.56898
%parameter_4
    5.29142 1.26117
```

3.1.2 Graphical User Interface

Aside from the previously listed core modules, the framework also includes a Graphical User Interface (GUI) which allows the user to evaluate/calibrate models without the need to code a complete program. The user is then only required to create a GUI object (of class *CaLib*) and to "add" any models, additional parsers and metrics that are to be used; listing 3.3 shows an example of the "main" function of such a use-case. The program resulting from this code will be used throughout this section to illustrate what can be done with the GUI module.

Listing 3.3: GUI usage example.

```
#include "CaLib.h"

// include other parsers, models, metrics

int main(int argc, char ** argv)
{
        CaLib calib(argc, argv);

        calib.addParser<ParserANA>("ANA");
        calib.addParser<ParserSeyfried>("Seyfried");

        calib.addModel<SimulationBoids>("Boids");
        calib.addModel<SimulationHelbing>("Helbing");
        calib.addModel<SimulationRVO2>("RVO2");
        calib.addModel<SimulationGeneric<PedestrianTangent> >("Tangent");
        calib.addModel<SimulationGeneric<PedestrianV> >("Vision");

        calib.addMetric(theEntropyMetric, "EntropyMetric");

        calib.run();

        return 0;
}
```

The GUI consists of two panels side by side.

The left panel contains all the controls, it allows to select data, models, metrics and calibration algorithms. This is illustrated in figure 3.2. In this figure, a sample file with two pedestrians crossing ways is chosen, as well as the Vision model. The Difference metric is further chosen and gives a score of 0.863642 with the standard parameters (the scores given in the GUI are a comparison with the Dummy model). In this scenario, the Genetic algorithm is chosen to calibrate the model. After calibration, the score obtained by the Vision model is 0.103674.

The right panel, on the other hand, allows to visualize the real and simulated crowds and has its controls at the bottom of the left panel. The user can choose to show or hide the real crowd, the simulated crowd and a line segment ("Dist" option) which represents the distance between a real pedestrian and the corresponding simulated agent. It is also possible to show a pedestrian's recent path ("Tails" option) as well as the complete path ("Paths" option). Figure 3.3a shows the paths of the real crowd (red) and simulated crowd (blue) with the standard parameters. Figure 3.3b shows the same but after calibration, with the new parameters. Finally, figure 3.3c only shows the trace of the *real-simulated difference* line segment, showing how close are the real and simulated paths (only options "Dist" and "Paths" are enabled).

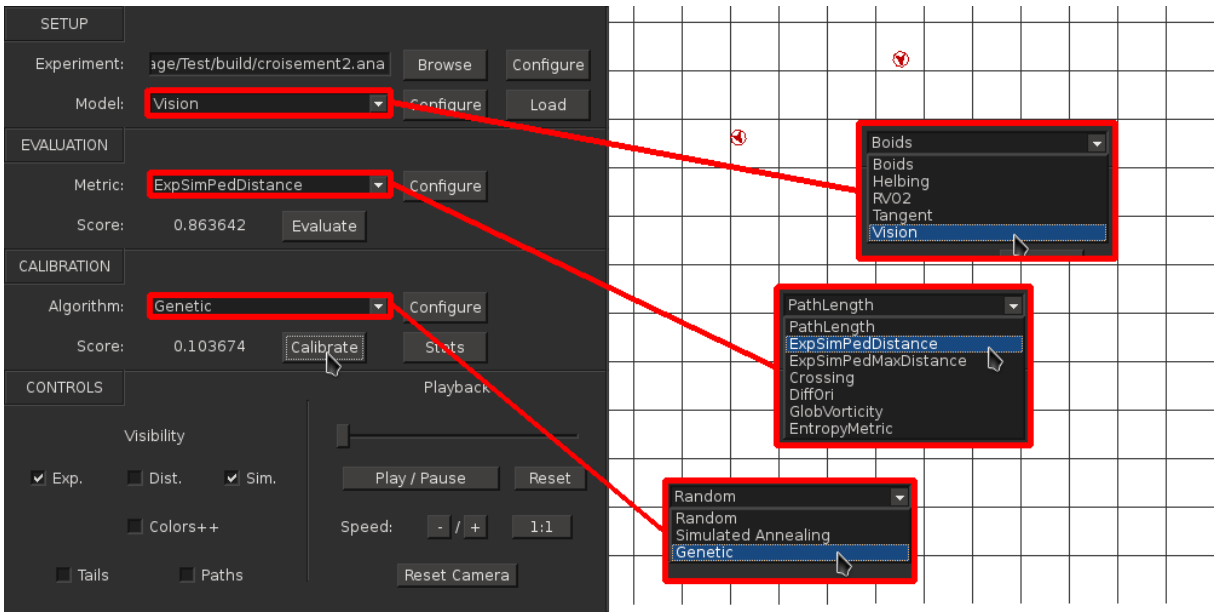
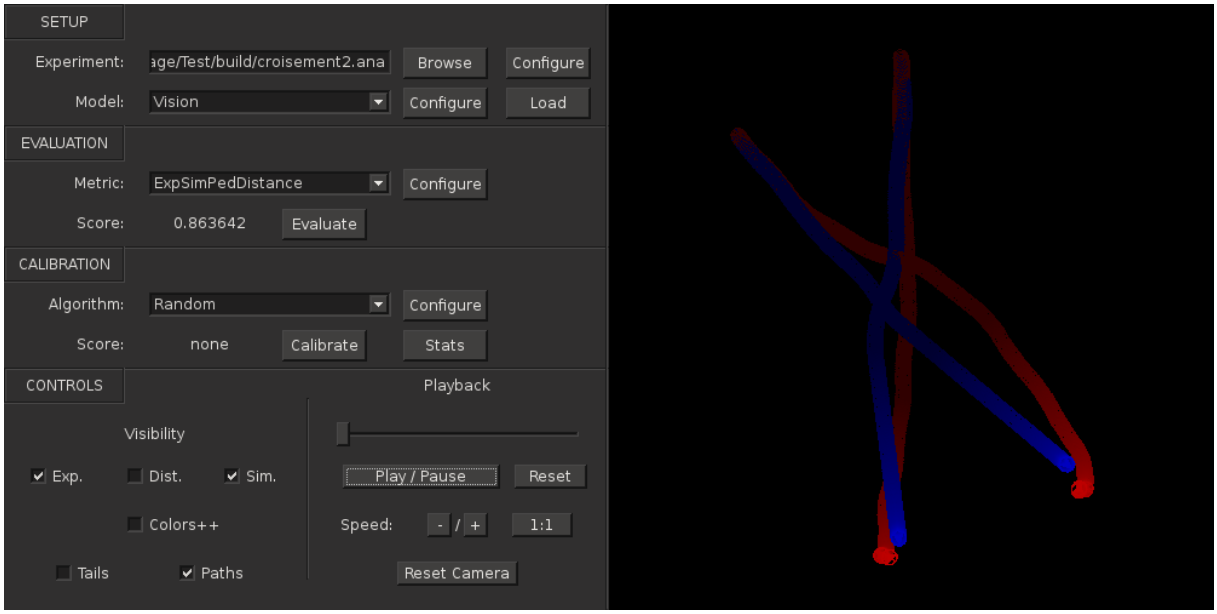
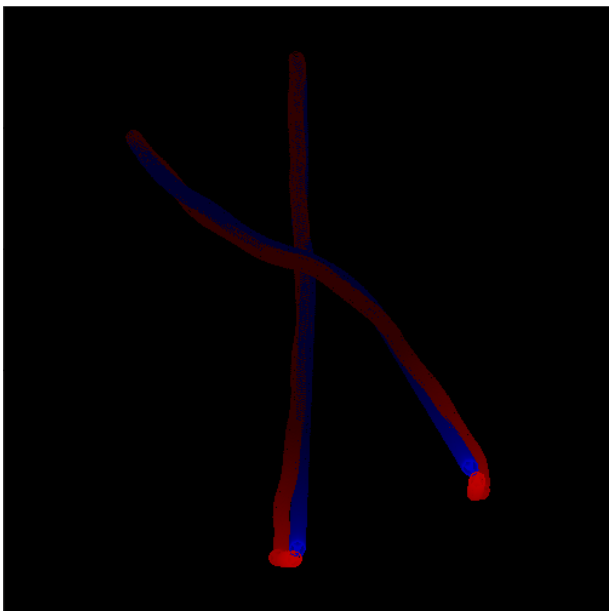


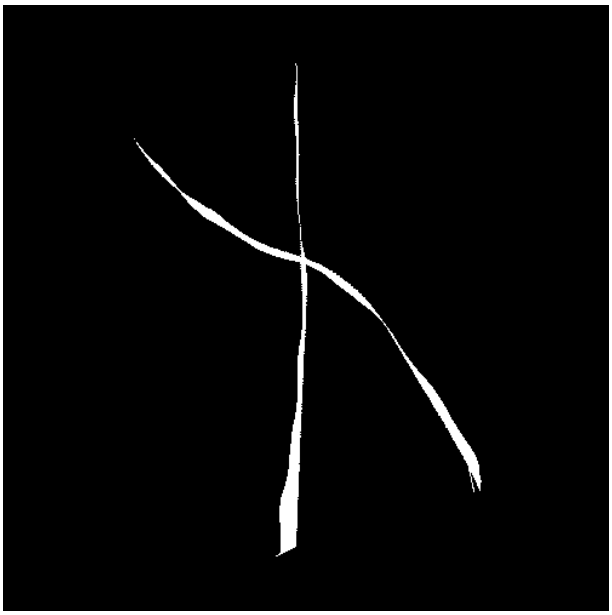
Figure 3.2: GUI, the *croisement2.ana* data file has been loaded, the Vision model, Difference metric and Genetic algorithm have been selected from the drop-down menus. Both the original score (0.863642) and new score (0.103674) are visible.



(a)



(b)



(c)

Figure 3.3: (a) entire GUI, pedestrians' (dissimilar) paths prior to calibration. (b) pedestrians' paths after calibration. (c) difference between the corresponding pedestrians' paths, showing how similar the trajectories have become.

3.2 Validation as a tool and preliminary results

This framework now needs to be validated as a tool. As such, an experiment had to be designed to provide enough data to answer the following questions:

- which calibration algorithms should preferrably be used?
- which metrics are suited to evaluate models based on the available data?

Additionally, if this experiment made the necessary data available, then how would the models compare?

This section presents some preliminary results that were obtained thanks to this experiment: a batch processing program using the framework's modules. It successively presents the program's inputs, results on the validation of the framework as a tool and, finally, some first results on the comparison of crowd simulation models.

3.2.1 Inputs

The batch processing program that was used to generate the results essentially consists of 5 cascading loops, nested in this order:

- a loop on data files
- a loop on models
- a loop on metrics
- a loop on calibration algorithms
- an additional loop for statistical validity

Figure 5 summarizes the program and the following paragraphs give additional information on the number of iterations of each loop.

Algorithm 5: Summary of the setup.

```
foreach datafile (87) do
    foreach model (Boids, RVO2) do
        foreach metric (Difference) do
            foreach calibration algorithm (G, SA, GA, GA+G, GA+SA) do
                foreach try (3) do
                    save result file
                end
            end
        end
    end
end
```

Data The loop iterated over 87 data files:

- 4 files containing 2 pedestrians
- 25 files containing 3 pedestrians
- 46 files containing 4 pedestrians
- 5 files containing 6 pedestrians
- 3 files containing 12 pedestrians
- 4 files containing 24 pedestrians

For more information on the available data, see section 2.1.2.

Models In this experiment, two models were used: Boids [15] and RVO2 [19].

Metrics The batch processing program takes a long time to generate all the results. One previous experiment was started taking into account all the metrics described in section 2.3.6. However this experiment has been ended after 3 days while only 5 data files had been processed. As a result, only the Difference metric was chosen for the next (final) experiment as it performed best. It allowed the experiment to "only" last 4 days in its entirety.

Calibration Five calibration algorithms were tested: the Greedy algorithm (G), Simulated Annealing (SA), the Genetic algorithm (GA) as well as two combinations, Genetic algorithm+Greedy algorithm (GA+G) and Genetic algorithm+Simulated Annealing (GA+SA) (see section 2.4.2 for the motivation behind combining the algorithms).

Additional loop To add some statistical validity to the results, while not prolonging the duration of the experiment too much, 3 was chosen as the number of tries for each combination of the afore-mentioned inputs.

For each combination of inputs, a result file was generated which regrouped the following information: summary of inputs, calibration duration (in seconds), reference score (Dummy model's raw metric score), current model's raw metric scores before and after calibration and statistics on the parameters (see paragraph 3.1.1). It is important to note that by including the raw metric score obtained by the Dummy model, all models' scores used in the following sections are normalized scores $\frac{score_{model}}{score_{Dummy}}$ as defined in section 2.4.1.

A sample result file is shown in listing 3.4. In this file:

- the inputs were:
  - data file *essai001.ana*
  - the Boids model
  - the Difference metric
  - the GA calibration method
  - second try
- the calibration took 6.973343 seconds, the scores were as follows:
  - reference (Dummy model) raw score: 6099.618164
  - Boids model's score before calibration: 4165.604980

  - Boids model's score after calibration: 1741.77217
- the Boids model's parameters after calibration were:
  - first parameter with mean 0.209328 and standard deviation 0.009366
  - second parameter with mean 1.244460 and standard deviation 0.222907

Listing 3.4: Sample output file.

```
%file
        exp-02/essai001.ana

%metric
        ExpSimPedDistance

%model
        Boids

%calibration
        Genetic

%iteration
        1

%calibrationTime
        6.973343

%refScore
        6099.618164

%initScore
        4165.604980

%optiScore
        1741.772217

%parameter_0
        0.209328
        0.009366

%parameter_1
        1.244460
        0.222907
```

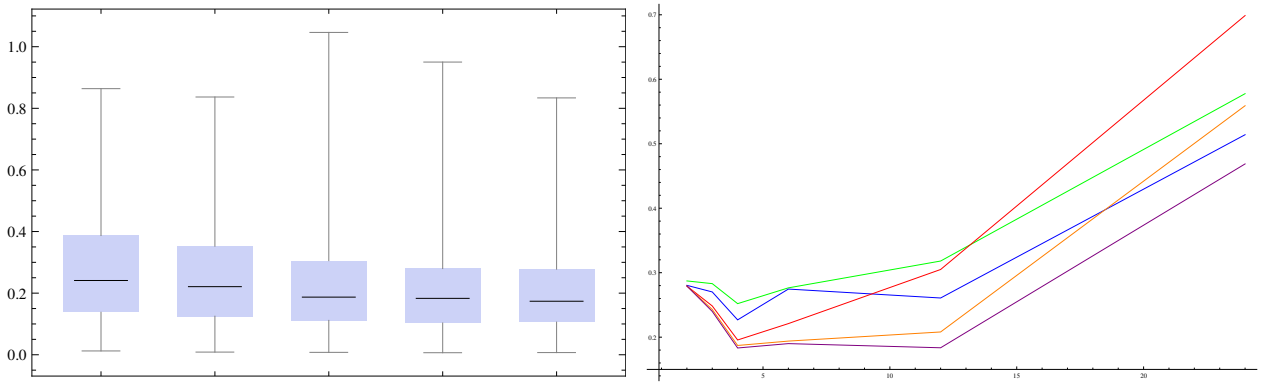
3.2.2 Validation of the framework as a tool

The previously described experiment generated 2610 result files. The first analysis that was made on these results was a comparison of the calibration algorithms. The second analysis focused on deciding if the Difference metric is suited for calibration. Finally, the last analysis is a comparison of the Boids and RVO2 models.

Choosing a calibration algorithm

Optimization levels The first measured quantity is the score minimization obtained by each calibration algorithm. This minimization *imin* is defined as the model's score after calibration divided by the score before calibration: $min = \frac{score_{after}}{score_{before}}$. Figure 3.4a depicts box plots of the distributions of the minimization *min* of each calibration algorithm over all

of the data. According to this figure, globally, all the calibration algorithms seem to give equivalent results. If necessary, one could say that the best algorithm is GA+SA followed by GA+G, GA, SA and G in this order. However, it is not enough to make an informed decision on the relative performance of these algorithms. The same graphs are obtained when only considering the data on the Boids model or the RVO2 model individually, which only suggests that the tendency is similar accross models. In an effort to compare *what is comparable*, the average minimizations *min* of these algorithms have been plotted against the number of pedestrians; figure 3.4b shows this. Here, the first observable tendency is that when the number of pedestrians increases, the scores become less and less *optimizable*. At this point, it is yet unclear if this is due to the dimentionality of the search space (equal to $nb_{agents} * nb_{params}$), or the Difference metric being ill-adapted to very populated scenarios, or the relatively small number of data files containing 12 or 24 pedestrians (3 and 4 respectively). However, the calibration algorithms can still be compared and it appears that with scenarios with up to ∼9 pedestrians, the previous order is still valid: GA+SA best, followed by GA+G, GA, SA, G. The differences are more distinct, however, as GA+G and GA+SA seem to have a great advantage over the remaining algorithms. After this threshold, algorithms containing SA (SA and GA+SA) perform best, followed by GA+G, G and GA.



(a) (b)

Figure 3.4: (a) distributions of the minimization *min* of calibration algorithms (in this order: G, SA, GA, GA+G, GA+SA). (b) average minimization *min* of calibration algorithms over the number of agents (**G**, **SA**, **GA**, **GA+G**, **GA+SA**).

Speed The second important criterion for this comparison is the time it takes to calibrate the models. To this end, the quantity *tng* has been measured. *tng* is defined as the duration of a calibration phase normalized by the same duration (under the same conditions) while using algorithm G: $tng = \frac{t}{t_G}$. Figure 3.5a depicts box plots of the distributions of the normalized duration *tng* of each calibration algorithm over all of the data. Here, it can be seen that G, GA and GA+G are the fastest algorithms, while algorithms containing SA (SA, GA+SA) are slower. To confirm this, the average duration in seconds of each algorithm has

been plotted against the number of pedestrians, as shown in figure 3.5b. On this figure, it can be seen that with more than 6 pedestrians, the difference between SA based algorithms and the rest becomes very important. This is also the case when looking only at values for a number of pedestrians between 2 and 6 (figure 3.5c).

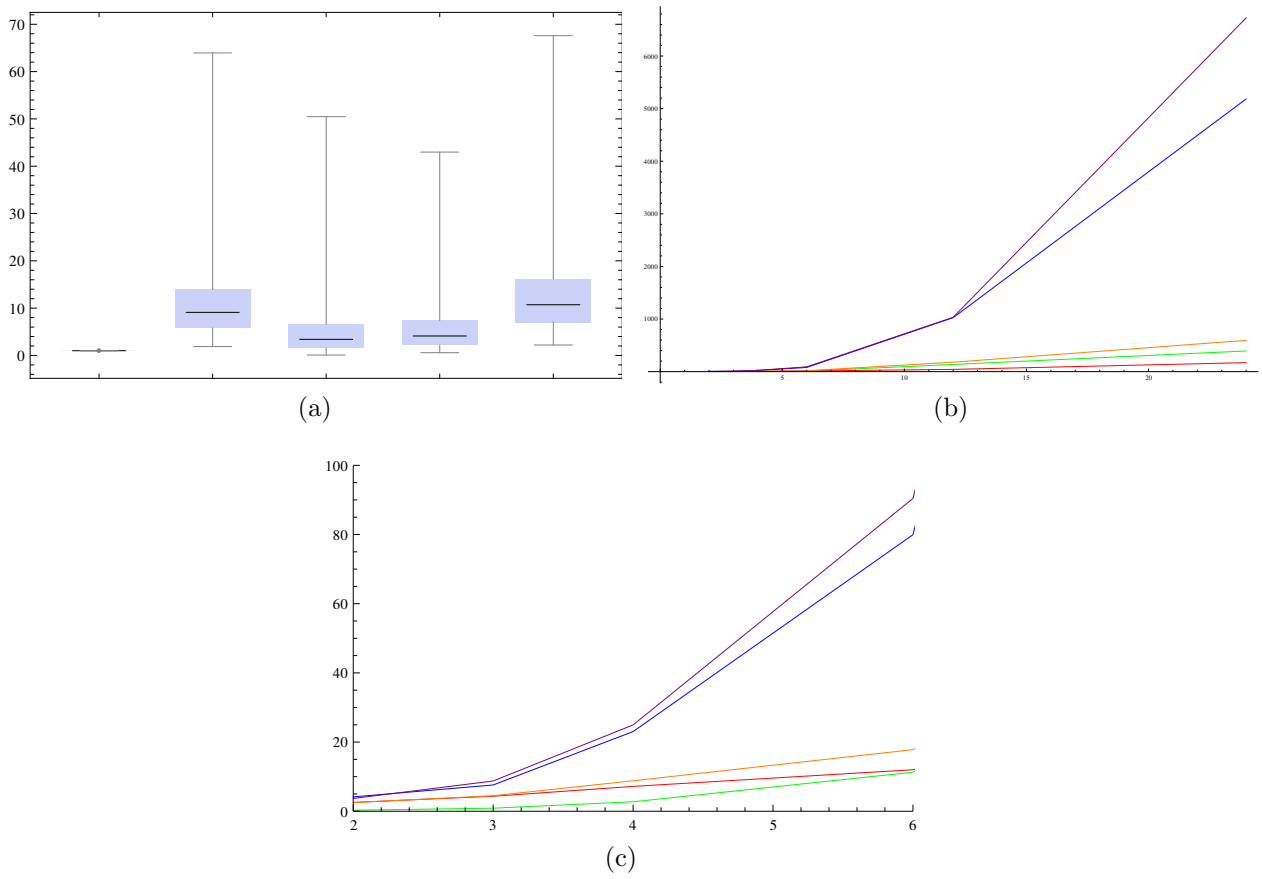


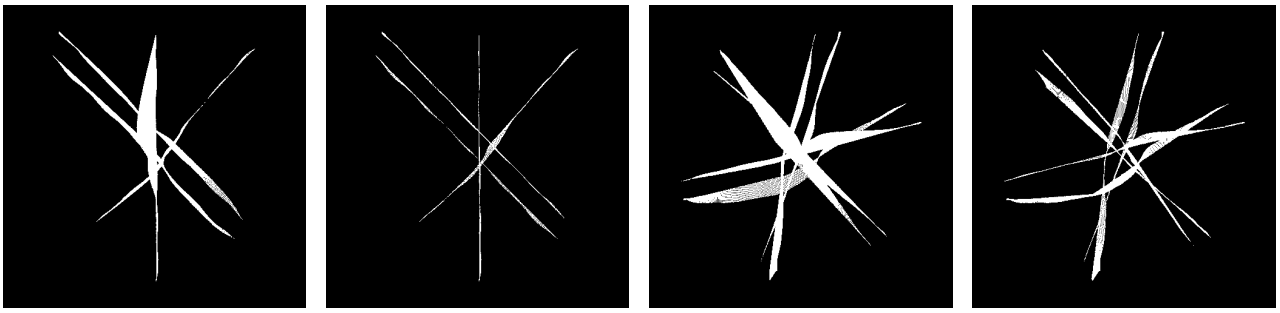
Figure 3.5: (a) distributions of the normalized duration *tng* of calibration algorithms (in this order: G, SA, GA, GA+G, GA+SA). (b) average duration in seconds of calibration algorithms over the number of agents (**G**, **SA**, **GA**, **GA+G**, **GA+SA**). (c) same as (b) only considering 2∼6 agents.

Conclusion When taking into account both the relative performance and the durations of the different calibration algorithms, GA+G can be chosen as offering a good compromise between performance and speed. It is, along with GA+SA, the best algorithm (especially for 2∼6 pedestrians) and, concerning speed, it is on par with G and GA. For these reasons, the GA+G algorithm can be chosen to perform calibration when evaluating and comparing models using the Difference metric. One thing to notice is that combining the GA algorithm with G and SA gives better results than any of the base algorithms (G, SA and GA) which

justifies what was discussed in section 2.4.2.

Choosing a metric

Considering the data generated by the batch processing program, it is difficult to make any definitive conclusions on the validity of the Difference metric. However, by looking at figure 3.4b, it can be said that if this metric is valid, it is most likely to be valid for scenarios including 2∼6 pedestrians. It can also be noted (in the same figure) that the minimization *min*, for 2∼6 pedestrians, is as low as ∼0.16. As it is a minimization of the path difference between real and simulated pedestrians compared to the same situation with default parameters, it is an argument in favor of the Difference metric. Furthermore, it is possible to take into consideration images showing path differences between real and simulated crowds before calibration (figures 3.6a and 3.6c) and after calibration (figures 3.6b and 3.6d) using this metric. These sets of images confirm that the difference between the paths of the real and simulated pedestrians is much reduced and that the final paths are very similar. This metric can be considered a sensible choice when comparing models on data containing 2∼6 pedestrians.



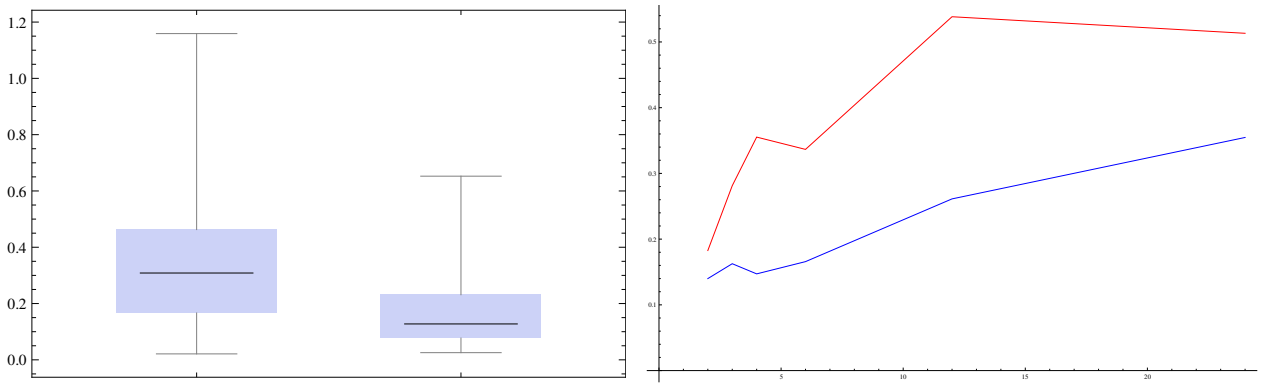
(a) (b) (c) (d)

Figure 3.6: Difference between real and simulated agents' paths. (a) & (b) Boids model before and after calibration - 4 agents. (c) & (d) RVO2 model before and after calibration - 6 agents.

3.2.3 First results on model comparison

In the previous paragraphs it has been shown that the GA+G calibration algorithm is a good choice as it offers a good compromise between performance and speed and that the Difference metric is adapted to the data at hand (especially for 2∼6 pedestrians). From the results generated by the batch processing program, it becomes possible to compare the two models: Boids and RVO2. Figure 3.7a shows the distributions of the scores of the two models over all data and figure 3.7b shows their average score plotted against the number of pedestrians. On both graphs it can clearly be seen that RVO2 gets much better results than Boids, both globally and when considering any number of pedestrians. It can then be

concluded that for this data (especially for 2∼6 pedestrians) and for this metric (and for this calibration algorithm), RVO2 is closer to a real crowd than Boids is.



(a) (b)

Figure 3.7: (a) distributions of Boids and RVO2 scores respectively. (b) average score of models **Boids** and **RVO2** against number of agents.

Conclusion

The work described in this report aimed to fill an empty space that exists in the area of the evaluation of crowd simulation models.

Currently, in the state-of-the-art, much work has been done on simulating crowds, but little has been done on evaluating the resulting models. As mentioned, the biggest effort in this direction has been made with the SteerBench framework. However, even this framework doesn't provide an entirely satisfactory answer. Its first major drawback is the lack of reliance on experimental data. The second, which derives from it, is that the metrics are based on the authors' opinions of what characterizes a crowd. SteerBench's final drawback is the lack of support for the models' parameters which, nevertheless, greatly affect a simulation's outcome.

In the scope of this work, a new modular framework has been developed including specially designed metrics and optimization algorithms. This framework allows to evaluate the models in a data-driven way and calibrate them, thus also taking into account their parameters. Furthermore, some experiments have already been carried out on this framework (and more are to be conducted), allowing to justify the calibration algorithms that it includes as well as the metrics that are used to evaluate the models.

Finally, these experiments allowed to make a preliminary comparison between crowd simulation models. More spcecifically, one of the models that have been tested was the Boids model which is a rule-based, position-based model that is very widely used even today (it was proposed in 1987). The other model was RVO2, which belongs to the category of predictive models. In the recent years, it was much argued that predictive models should be superior to position-based models and the preliminary results obtained here can then be considered as part of an answer to that debate.

As a conclusion to this work, it should be mentioned that the bibliographic study carried out for this internship has contributed to an article which will be presented in the PED2012 [1] conference (June 6-8). The results on the validation of crowd simulation models will be part of a common survey paper (with the Gamma group at UNC) which is in preparation. Finally, more exhaustive results are likely to be the basis for a future article about calibration problems.

Bibliography

[1] Ped2012 conference. `http://www.ped2012.org/`.

[2] The pedigree project. `http://www.pedigree-project.info`.

[3] Seyfried data. `http://www.asim.uni-wuppertal.de/en/database.html`.

[4] Stephen guy's home page. `http://wwwx.cs.unc.edu/~sjguy/new/`.

[5] Stephen J. Guy, Sujeong Kim, Ming C. Lin, and Dinesh Manocha. Simulating heterogeneous crowd behaviors using personality trait theory. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 43–52, New York, NY, USA, 2011. ACM.

[6] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *PHYSICAL REVIEW E*, 51:4282, 1995.

[7] Roger L. Hughes. A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodological*, 36(6):507–535, July 2002.

[8] Ioannis Karamouzas and Mark Overmars. Simulating and evaluating the local behavior of small pedestrian groups. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints), 2011.

[9] Tobias Kretz and Michael Schreckenberg. The f.a.s.t.-model. *CoRR*, abs/0804.1893, 2008.

[10] S. Lemercier, A. Jelic, R. Kulpa, J. Hua, J. Fehrenbach, P. Degond, C. Appert-Rolland, S. Donikian, and J. Pettré. Realistic following behaviors for crowd simulation. 2012.

[11] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C. Lin. Aggregate dynamics for dense crowd simulation. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 122:1–122:8, New York, NY, USA, 2009. ACM.

[12] Jan Ondřej, Julien Pettré, Anne-Hélène Olivier, and Stéphane Donikian. A synthetic-vision based steering approach for crowd simulation. *ACM Trans. Graph.*, 29:123:1–123:9, July 2010.

[13] Sébastien Paris, Julien Pettré, and Stéphane Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Computer Graphics Forum, Eurographics'07*, 26(3):665–674, September 2007.

[14] Julien Pettré, Jan Ondřej, Anne-Hélène Olivier, Armel Cretual, and Stéphane Donikian. Experiment-based modeling, simulation and validation of interactions between virtual walkers. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 189–198, New York, NY, USA, 2009. ACM.

[15] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. ACM.

[16] Andreas Schadschneider. Cellular automaton approach to pedestrian dynamics - theory. page 11, 2001.

[17] Shawn Singh, Mishali Naik, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinman. Watch out! a framework for evaluating steering behaviors. In *MIG'08*, pages 200–209, 2008.

[18] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Trans. Graph.*, 25:1160–1168, July 2006.

[19] Jur van den Berg, Stephen J. Guy, Ming C. Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *INTERNATIONAL SYMPOSIUM ON ROBOTICS RESEARCH*, 2009.

[20] Jur van den Berg, Sachin Patil, Jason Sewall, Dinesh Manocha, and Ming Lin. Interactive navigation of multiple agents in crowded environments. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, I3D '08, pages 139–147, New York, NY, USA, 2008. ACM.

[21] Jur P. van den Berg, Jamie Snape, Stephen J. Guy, and Dinesh Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *ICRA'11*, pages 3475–3482, 2011.

Appendix

1 Genetic algorithm example

Figure 8: Genetic algorithm example.

2 Framework sepcification details

This section aims to briefly describe the specifications a potential user needs to be aware of during "normal" use of the framework (which is illustrated in listing 3.1 from section 3.1.2). These include:

- defining a parser
- interfacing a model
- deciding which pedestrians are simulated
- defining a metric
- calibrating

2.1 Defining a parser

Assuming the user wants to define a parser for ".example" files, a class *ParserExample* can be defined which inherits from the *Parser* class and implements two methods; see listing 5.

Listing 5: The definition of the *ParserExample* class.

```
#include "Parser.h"

class ParserExample : public Parser
{
public:
        ParserExample()
        {
                g_extension = "example"; // tells what the files' extension is
        };
        virtual ~ParserExample();

        virtual void parse(const char * filename);

        virtual void fill(int * nPed, int * nMes,
                float ** centerx, float ** centery,
                float ** centerVelocityx, float ** centerVelocityy,
                float ** times);
};
```

The *parse(filename)* method should parse the file *filename*. The method *fill(..)* is used to transfer the data to the *Experiment* module, therefore it should modify the passed parameters in the following way (example in listing 6):

- *nPed* and *nMes* should be set to the number of pedestrians and number of frames
- *centerx* should be set to contain an array of X-axis pedestrian coordinates; the size of array *(*centerx)* should be *nPed*nMes* and *(*centerx)[i*nMes+j]* should hold the j^{th} X-axis coordinate of the i^{th} pedestrian (similarly *centery* for Y-axis coordinates)

- *centerVelocityx* and *centerVelocityy* are similar to *centerx* and *centery* except they should hold the X-axis and Y-axis components of the pedestrians' volocities
- *times* should be set to contain an array of timestamps; the array *(*times)* should be of size *nMes* and *(*times)[i]* should hold the timestamp of the i^{th} frame

Listing 6: Example of an implementation of *fill()*.

```
virtual void fill(int * nPed, int * nMes,
        float ** centerx, float ** centery,
        float ** centerVelocityx, float ** centerVelocityy,
        float ** times)
{
        // 2 pedestrians
        *nPed = 2;

        // 100 frames
        *nMes = 100;

        // create arrays
        *times = new float[100];
        *centerx = new float[2*100];
        *centery = new float[2*100];
        *centerVelocityx = new float[2*100];
        *centerVelocityy = new float[2*100];

        // fill arrays...
}
```

2.2 Interfacing a model

Assuming a user wants to interface a model (e.g. *Example* model) with the framework, he should define a new class (e.g. *SimulationExample*) which inherits from the *Simulation* class. Listing 7 shows how such a model is interfaced.

Listing 7: The definition of the *SimulationExample* class.

```
#include "Example.h"

class SimulationExample : public Simulation
{
public:
        SimulationExample();
        virtual ~SimulationExample();

        virtual void init();
        virtual void reset();

        virtual void setPosition(int s_indPedestrian, float s_x, float s_y);
        virtual void setVelocity(int s_indPedestrian, float s_x, float s_y);
        virtual void setGoal(int s_indPedestrian, float s_x, float s_y);

        virtual void doStep(float s_dt);
};
```

- in the constructor, the user adds parameters thanks to the *addParam()* function; an example is given in listing 8
- in *init()*, the user initializes the *Example* model with *g_nPedestrian* agents; this depends on *Example*'s specfications
- in *reset()*, the user copies the pedestrians' parameters to *Example*'s agents (see listing 8 for an example)
- *setPosition(..)*, *setVelocity(..)* and *setGoal(..)* are used to set *s_indPedestrian*'s position, velocity and goal
- *doStep(s_dt)* should compute the *Example* model's agents' next positions and velocities; at the end of this function the user should call the *setNextState()* function for each agent, an example is shown in listing 8

Listing 8: Examples.

```
// add a parameter: preferred speed (in m/s): default value 1.6
// base distribution: Normal with mean 1.5, standard deviation 0.5
// clamped between 1 and 2
addParam(1.6, PDF(1, 2, PDF::NORMAL, 1.5, 0.5));

// set an agent's parameter
example.agent(i).parameter(j) = g_params[i*g_nbParams+j]

// setNextState for agent number i:
setNextState(i,
             example.agent(i).Xcoordinate,
             example.agent(i).Ycoordinate,
             example.agent(i).XvelocityComponent,
             example.agent(i).YvelocityComponent);
```

2.3 Deciding which pedestrians to simulate

Deciding which pedestrians are simulated is done by using the *Record* object. For a given agent, the user calls the *setMask()* method. When the user is done, *saveMask()* should be called. Listing 9 gives an example.

Listing 9: Choosing which pedestrians to simulate.

```
// initialize a Record object called record, see section 3.1.1

// assume we don't want to simulate some pedestrians number 2, 3 and 7
record.setMask(2, false);
record.setMask(3, false);
record.setMask(7, false);

// save changes
record.saveMask();
```

2.4 Defining a metric

Each metric is defined as one function inside the *Metrics* namespace. An example metric can be defined as shown in listing 10.

Listing 10: Example metric.

```
namespace Metrics
{
        float exampleMetric(Record * record)
        {
                // we want the complete path to be pre-computed
                record->computePath();

                // get number of pedestrians and measures
                int nbPeds = record->getNPedestrian();
                int nMeasures = record->getNMeasure();

                // get last known coordinates of real pedestrian number 0
                float X1 = record->getExpPositionx(0, nMeasures-1);
                float Y1 = record->getExpPositionx(0, nMeasures-1);

                // get first coordinates of last simulated pedestrian
                float X2 = record->getExpPositionx(nbPeds-1, 0);
                float Y2 = record->getExpPositionx(nbPeds-1, 0);

                // compute the score based on a weird formula
                float score = (X1-X2) * (Y1-Y2);

                return score;
        }
}
```

2.5 Calibrating

Currently, three algorithms are available to calibrate the models, they are summed in listing 11. These algorithms can be combined with the exception that the Genetic algorithm won't take into account any previous calibration, so if it is to be used, it should be used first.

Listing 11: Calibration example.

```
// initialize a Record object called record, see section 3.1.1

Calibration calibration(&record);

// the user can calibrate (with respect to someMetric) using:
// the Greedy algorithm:
calibration.calibrate(Metrics::someMetric);

// or the Simulated annealing:
calibration.calibrateSimAnneal(Metrics::someMetric);

// or the Genetic algorithm:
calibration.calibrateGenetic(Metrics::someMetric);
```