ANNÉE 2016

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale Matisse

présentée par

David Wolinski

préparée à l'unité de recherche UMR 6074 IRISA
et au centre INRIA - Rennes Bretagne Atlantique
ISTIC

Microscopic Crowd Simulation: Evaluation and Development of Algorithms

Thèse soutenue à Rennes
le 22 janvier 2016

devant le jury composé de :

Yiorgos Chrysanthou
Professeur, University of Cyprus / rapporteur

Pierre Degond
Professeur, Imperial College London / rapporteur

Ming Lin
Professeur, University of North Carolina / examinateur

Armin Seyfried
Professeur, Jülich Supercomputing Centre / examinateur

Kadi Bouatouch
Professeur, Université de Rennes 1 / examinateur

Julien Pettré
Chargé de recherche, Université de Rennes 1 / directeur de thèse

Contents

List of Figures

List of Tables

Introduction 1

The demand for crowd simulation has sky-rocketted in recent years, with entertainment and safety at the forefront of its applications. Evermore ambitious and epic movies and video games call for ever larger armies or background crowds, while ever stricter safety rules require urban designers and architects to make increasingly accurate predictions of crowd behaviors.

Blockbuster movies in particular have recently made the transition from calling upon large numbers of "extras" to using computer-generated crowds ("crowds" here loosely refer to any collection of on-screen entities) in order to populate their scenes. Thus, one can now watch synthesized crowds in varied works and contexts. For instance, very large armies have been a central aspect of movies or television shows such as *Lord of The Rings*, *300*, *Game of Thrones* etc. Computer-generated crowds can also be found in other forms, such as large amounts of swarming zombies in *World War Z*, apes (and humans) in *Dawn of the Planet of the Apes*, minions in the *Despicable Me* franchise etc. The main requirement in bringing these animated crowds to life (in addition to the quality of motion), is a high level of control of the artist over the simulation, directing the behaviors and styles as needed to fit a particular project.

In video games, crowd simulators are tasked with the steering of every dynamic (interactable) non player character, ranging from pedestrians roaming the streets of *Assassin's Creed* games to the countless soldiers of the *Dynasty Warriors* franchise. While both video games and movies require a high level of control over the simulations as well as a high level of realism/believability, video games (and any other interactive experience) additionally need the synthesized characters to be interactable: agents need to be autonomous and the simulation needs to be real-time.

In terms of security and urban planning, crowd simulation is similarly widely applicable. For instance, organizers of large "open" events (e.g. concerts) can use simulators to make statistical predictions on the crowd's behavior, in order to improve the layout of their installations and/or intended routes (hopefully avoiding disasters such as the one during the *Love Parade* music festival in Duisburg, Germany, in July 2010). Obviously, similar approaches can also be adopted when designing a new building/structure, such as an airport/train station, a mall, offices, cruise-ships etc. Additionally, in the case of such buildings it is also possible to track people during an evacuation in order to make predictions on the availability of exit points, and ultimately to guide the evacuees along optimal routes, avoiding congestion. As a more specific application, one can also use queueing simulators to optimize waiting lines at amusement parks, airports/train stations etc. Overall, in these cases, the most important aspect is the (mostly statistical) accuracy of the simulated crowds as well as their ability to predict situations' outcomes, with some situations such as evacuations also requiring real-time performance.

With this wide variety of applications, many algorithms have been developed, often for specific purposes. Consequently, the choice of which algorithm to use for a given task is not an easy one.

1 Problem

Assuming a pool of available algorithms to simulate crowds in a user's target application, the task of such a user when choosing the appropriate algorithm is to answer a few fundamental questions.

Performance How fast does the algorithm need to be? Interactive experiences (e.g. video games) require real-time crowds while movies can make do with more time consuming offline computation. Similarly, a system which directs pedestrians along the optimal path during an evacuation needs to adapt quickly to changing situations, while the validation of new building's design is less urgent.

Autonomy The next question is: how much user intervention is required for the algorithm to satisfactorily achieve the desired effect? For instance, can the simulator produce the right result with the user simply specifying "1000 humans, from here to here, moving fast, scared", or is the user required to manually check and correct the behavior of each simulated individual (keeping in mind that each individual change could impact the whole simulation)?

Control Another facet is the issue of control (or flexibility), i.e. can the user profitably direct the simulation algorithm to fit another particular need? Could the same simulator produce a different crowd with another specification such as "1000 tourists, from here to here, moving slowly, curious" or does it have a limited domain of application?

Realism The last question is also an immediate one: how accurate/realistic will the simulation be? An algorithm would not be of much use to an artist if the simulated characters broke the audience's immersion, nor would it be of much use to an architect studying evacuation cases if the simulated pedestrians did not sufficiently behave like humans. As a simulator could be more suited for certain applications than others, this is also related to the question of the validity domain of an algorithm: assuming a pool of available algorithms, which one is most suitable for a given application?

While all these questions need to be considered when choosing (or developing) an algorithm, they are not equally easy to answer. The question of performance is an easy one, the theoretical algorithmic complexity of a simulator is not difficult to assess, and as a last resort simply running the simulator gives an idea of its performance and scaling ability. The questions of autonomy and control are trickier, as they require a deeper knowledge of the considered algorithm's capabilities: mainly its applicability domain (possible instabilities at certain densities, holonomic/non-holonomic agents etc.) and the effects of parameters (affecting for instance how cautious simulated characters are

of each other). Finally, the question of realism is by far the most difficult to answer as: (1) in many cases there is no clear definition to what "realistic" means, and (2) some algorithms could in theory achieve the target result but their tuning to do so is not known nor trivial.

Thus the main objective of this thesis is to globally improve crowd simulation algorithms' realism by: (1) designing a generally applicable scheme to evaluate the realism of simulation algorithms (while keeping in mind the questions of autonomy and control), and (2) further developing more capable simulation algorithms.

2 Approach

As a basis for our work, we chose to approach this question of realism from the perspective of microscopic, agent-based crowd simulators (detailed in Section 2) as they are a widely used class of algorithms, thanks to their ease of use and implementation as well as their flexibilty.

From this perspective, we first focused on the evaluation of crowd simulation algorithms. The two main objectives of this work were to validate existing algorithms and to develop a framework which we could later use to validate future ideas. We based this framework's evaluation scheme on real-world data, using various metrics to compare tracked pedestrians' trajectories and the corresponding ones from the simulated agents'. As a second component, we incorporated parameter estimation into this framework, allowing us to do two things. The first is a fair comparison between algorithms, as each algorithm is optimally tuned during the testing, an the second is the exploration of the question of control, as the tuning of parameters according to different criteria can be used for instance to help artists adapt a given algorithm to varied situations.

During this work, it became clear that even with recent progress on microscopic, agent-based algorithms, many artifacts and simulation errors persist. This led to the second main piece of work presented in this document, concerning the development of crowd simulation algorithms. Broadly, while "first-order" algorithms (see Chapter 2, Section 2) are intuitive to implement, extend and use, their simulation results are not as good as that of "second-order" algorithms (algorithms which anticipate collisions by linear trajectory extrapolation), which in turn are however more difficult to extend, and also still produce noticeable artifacts. In order to further improve simulation results, we designed an easily extendable algorithm which works in the 3-dimensional space (2D positions plus time) and where agents *perceive* probabilities of colliding with each other. These probabilities are computed on the basis of an *intrinsic field* which represents agents' collision probabilities that are due to their co-existence (i.e. agents are not reduced to a point), and every source of information that we wish to take into account further *warps* this field. Thus, we implemented *warp operators* which model: agents' perception error (agents perceive imminent situations better than ones further into the future), agents' radius, agents' future trajectory prediction based on velocity (linear), perception errors due to velocity, prediction based on the environment layout (non-linear), prediction based on agents' past trajectories (non-linear) and prediction based on agents' interactions with obstacles (i.e. the impossibility for an agent to traverse obstacles; also non-linear). Furthermore we can easily visualize these collision probabilities when checking simulation

results, thus making it easier to analyze and extend the algorithm.

In parallel, while exploring and looking for available ground-truth data, the work on evaluation and parameter estimation has lead to two further applications. In these applications we approach the question of the validity domain of each simulation algorithm. Thus, we use our work as a simulation algorithm selector in order to determine which simulator is most suitable for a (1) simulation or (2) an instant during a simulation:

The first application concerned the simulation of swarms of insects, as data on individual flying insects' trajectories had recently been made available. Consequently, we applied our work in order to design a data-driven insect swarm simulator. We used the parameter estimation to select the most appropriate collision-avoidance algorithm and tune it in order to reproduce low-level insect behaviors, and then completed the approach with additional statistical mechanisms taking care of the "zig-zaggy" nature of each insect's trajectory and their high-level swarm behavior respectively.

The second project prompted by the work on evaluation and parameter estimation concerned tracking. Here, we based ourselves on an already explored approach where a tracker is paired with a simulation algorithm which helps the tracker by predicting where pedestrians are likely to move. We worked on perfecting this tracker-simulator approach in a general, simulator-agnostic way. As confirmed during our work on parameter estimation, it is easier to fully reproduce crowd behaviors at all times and under all circumstances with several, well tuned algorithms than with a single simulator with a general parameter set. Thus, we use already tracked portions of tracked pedestrians' trajectories to estimate the optimal parameters of several, concurrently running simulators. Then, we select the (tuned) simulator which best matches these already-known portions to predict the pedestrians' next positions to help the tracker, thus much improving the overall tracking accuracy.

3 Contributions

As per the title of this thesis "Microscopic Crowd Simulation: **Evaluation** and **Development** of Algorithms", our two main contributions (as first author) are as follows:

Evaluation and Parameter Estimation, Chapter 3 We propose a framework/method which aims to provide an objective and fair evaluation of the realism of crowd simulation algorithms. "Objective" here means the use of various metrics quantifying the similarity between simulations and ground-truth data acquired with real pedestrians. "Fair" here means the use, along with the similarity metrics, of parameter estimation to automatically tune the tested algorithms in order to match the ground-truth data as closely as possible (with respect to the metrics), thus effectively allowing to compare algorithms at the best of their capability. We also explore how this process can increase the level of crontrol a user has on the simulation while simultaneously reducing the amount of necessary user intervention.

Collision Avoidance, Chapter 4 We propose a new collision-avoidance algorithm, solving artifacts which still persist in current approaches. In order to achieve this, where current algorithms predict collisions by linearly extrapolating agents' trajectories, we

introduce an algorithm which better predicts agents' future motions in a probabilistic, non-linear way, taking into account the environment layout, agent's past trajectories and interactions with other obstacles among other cues. The resulting simulations do away with commonly observed artifacts such as: slowdowns and visually erroneous agent agglutinations, unnatural oscillation motions, or exaggerated/last-minute/false-positive avoidance manoeuvres.

Applications to Evaluation and Parameter Estimation Additionally, we also present two other contributions in the form of two applications to the parameter estimation and evaluation framework (as second author):

Insect Simulation, Chapter 5, Part 1 First author: Weizi Li, PhD student at the Gamma Group from the University of Carolina at Chapel Hill, NC, USA. In this work, we simulate biologically-inspired insects, i.e. insects leading to swarming behaviors as close as possible to ones observable in nature. Thanks to multiple available simulation algorithms, local similarity metrics and parameter estimation, agents' local interactions are as close as possible to real insects', while their noisy/zig-zaggy trajectories and global swarm structures are also learned from collected data. The resulting simulations reproduce real swarm behaviors as measured by metrics classically-used in the field of insect simulation.

Pedestrian Tracking, Chapter 5, Part 2 First author: Aniket Bera, PhD student at the Gamma Group from the University of Carolina at Chapel Hill, NC, USA. In this work, we track pedestrians using a method which benefits from motion prior. The approach is itself not new: a tracking algorithm provides position information to a crowd simulator which predicts the pedestrians' next positions and informs the tracking algorithm, which in turn improves its accuracy. The novelty of our work lies in the prediction of the pedestrians' next positions: instead of using a single algorithm, we use similarity metrics and parameter estimation in real-time to tune several simulation algorithms in order to match the already observed trajectories as well as possible and then select the algorithm that matches them best. This way, we effectively optimize our prediction capability and, consequently we much improve tracking accuracy in challenging conditions of lighting and inter-agent occlusions.

Background

2

Contents

While the following chapters present work touching upon various topics including evaluation, insects, pedestrian tracking, and collision avoidance, this chapter aims to provide some common ground for the rest of this thesis in the form of related work in the field of crowd simulation. Thus, we here give an overview of the various algorithms that have been devised over the years to simulate crowds, that is collections of agents, possibly ranging from insects to humans.

1 Introduction

Due to the numerous and varied applications to crowd simulation, and thus due to the numerous and varied properties required by each of these applications, a large number of simulation algorithms have been proposed. These algorithms vary in their scope: ranging from microscopic (crowds formed of interacting agents) to macroscopic (crowds simulated as a whole, at the expense of individual agents). They vary in their intention: either trying to replicate real, observed crowds or trying to build them from the ground up. And of course, most of all, they vary in their approaches: they can be physics-derived, vision-based, geometric, rule-based, probabilistic etc.

Thus, we give an overview of crowd simulation algorithms grouped by approach type. First we detail microscopic, autonomous agent-based algorithms by increasing level of complexity in Section 2, as they are the focus of this thesis, then we list other types of algorithms according to their scale in Section 3. Overall, we start with microscopic algorithms and end with macroscopic algorithms.

2 Autonomous Agent-based Algorithms

Agent-based algorithms, which are the focus of this thesis, aim to build crowds as a result of combining interacting agents. As already mentioned, to design such an algorithm, one needs to define three things: agents, interaction rules between agents, and the mechanism which combines interactions between several agents. Traditionally, these interaction rules focus on collision avoidance, as it has the biggest influence on agents' trajectories, and is still challenging in certain conditions. Nonetheless, the advantage of agent-based algorithms is that even simple rules may lead to surprisingly complex crowds. This means that one can design an algorithm which can simulate a crowd well at the global level all the while focusing on making interactions between individual agents as realistic as possible.

Thus, the complexity of interaction rules in agent-based algorithms has slowly increased with each new proposed approach. Broadly, one can identify *first-order* algorithms, where the dominant cue for interaction is the position of the agents, and the more recent *second-order* algorithms, which started predicting where and when collisions will take place in order to avoid them with anticipation.

2.1 First-Order Algorithms

To define the interaction rules, agent-based algorithms have started by focusing on agents' positions. In his seminal work, Reynolds [Rey87] builds flocks of *Boids* by defining three rules:

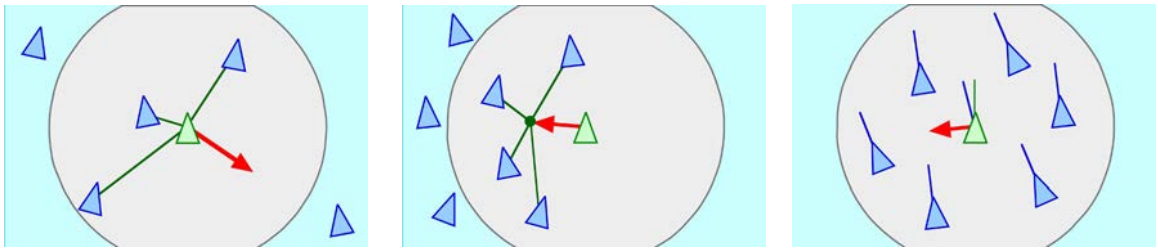
1. **Separation** pushes boids away from each other as repulsive forces between neighbors (Figure 2.1(a)).
2. **Cohesion** keeps boids from dispersing, by making them attempt to move towards the center of the flock (Figure 2.1(b)).
3. **Alignment** keeps boids flying in the same general direction, in order to have a coordinated flock behavior (Figure 2.1(c)).

Vicsek and colleagues similarly define self-driven particles [VCBJ+95] where the speed of each particle is fixed but the orientation is set as the average orientation of neighbor particles with some random perturbation (similar to the *Alignment* rule from *Boids*).

Repulsive forces between agents were further investigated by algorithms such as Social Forces [HM95, HFV00]. This formulation models agents as particles subjected to various forces, such as repulsion from neighbor agents and attraction to their destination. The moton of an agent is thus defined by an equation analogous to Newton's second law of motion:

$$\frac{d\overrightarrow{v}_\alpha}{dt} = \overrightarrow{F}_\alpha(t) + fluctuations, [\text{HM95}] \tag{2.1}$$

where the agent's velocity $\overrightarrow{v}_\alpha$ is computed from various forces $\overrightarrow{F}_\alpha$. Additionally to repulsion from neighbor agents (collision avoidance) and attraction to destinations, other



(a) Separation (b) Cohesion (c) Alignment

Figure 2.1 – Boids flocking rules [Rey99].

forces can be defined, such as time-varying attraction forces to friends, store fronts or other points of interest. For instance, this algorithm has been extended to take into account the speeds of the agents as well as their relative angles in the computation of the forces [JHS07].

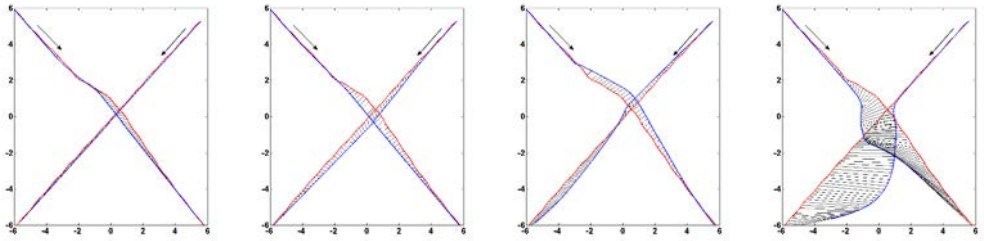
The main advantages of these algorithms are their ease of implementation and extendability, justifying their wide use. Implementing these interaction rules can be done in a matter of minutes, and additional rules and forces can be quickly and intuitively defined and integrated into the algorithms. Additionally, even with these simple rules, one can simulate flocks and crowds which display convincing group behaviors. However locally, individual agents' trajectories are not always very convincing and many pathological situations must be taken care of for a robust implementation: at higher densities for instance, Social-Forces based algorithms (akin to a physics simualtion) can become unstable.

2.2 Second-Order Algorithms

Solving many artifacts found in these previous approaches, algorithms predicting future collisions have been investigated. As opposed to previous methods where interactions are based on positions (first-order), these algorithms consider agents' instantaneous velocities (second-order) to linearly extrapolate their future trajectories, and thus to predict collisions. Some effects of second-order algorithms over first-order ones can be seen in Figure 2.2.

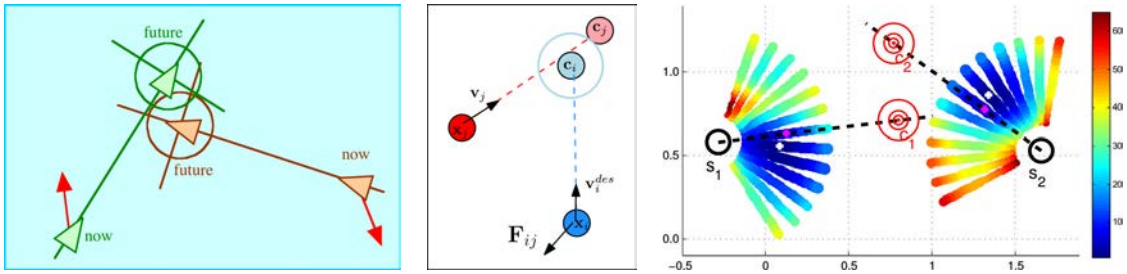
2.2.1 Repulsive-Forces from Future Collisions

Some algorithms, such as those found in [Rey99] (an extension to the Boids algorithm), [KHBO09] or [PESVG09], predict where a collision between two agents will take place and steer agents away from it, through various repulsive forces (Figure 2.3). Lamarche and Donikian similarly compute future collisions, but then discretize them and identify if those are rear, front, back, or static collisions, with a velocity-adaptation module defined for each type [LD04].



(a) [POO+09] (b) [vdBLM08] (c) [Rey99] (d) [HM95]

Figure 2.2 – Simulation results of two interacting agents [POO+09] with second-order algorithms (a-c) and a first-order algorithm (d), as compared to ground-truth data. Real trajectories are in red and simulated ones are in blue, the hatched area thus represents the simulation error. Second-order algorithms have a lower error than the first-order one.



(a) [Rey99] (b) [KHBO09] (c) [PESVG09]

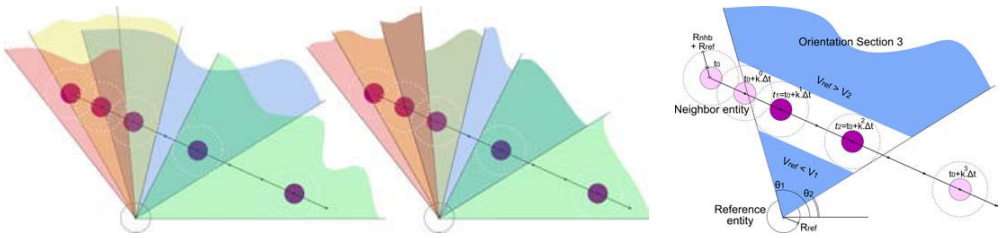
Figure 2.3 – Steering away from future collisions. (a) and (b) Repulsion forces are directed similarly, F_{ij} is directed by vector $\overrightarrow{c_j c_i}$ in (b). (c) Agents are subjected to repulsive forces originating from each other's future point of closest approach (e.g. force on agent $s1$ directed by $\overrightarrow{c_2 s_1}$).

2.2.2 Collision-Free Velocities

Other algorithms solve collisions in a different way, by searching for a velocity (both the orientation and speed of and agent, usually a two-dimensional vector) leading to a collision-free trajectory. Although the problem is three-dimensional (trajectories are sets of points with a two-dimensional position and a time component), by using agents' instantaneous velocities to linearly extrapolate their future trajectories, the problem can easily be simplified to a two-dimensional one. Consequently, many approaches to collision avoidance have proposed such a simplification, followed by a corresponding solution. For instance, Feurtey defines a disc of points that are reachable with specific velocities in Δ time [Feu00], and chooses agents' velocity adaptations from this disc based on a cost (cost of colliding, changing direction and changing speed).

Overall, certain velocities lead to a collision while others are collision-free: they are qualified as inadmissible and admissible velocities, respectively. Consequently, Paris et al. [PPD07] divide the space in front of an agent into sections, corresponding to different orientations of said agent (different colors in Figure 2.4(a)). Then for each section, they determine the speeds that lead to a collision-free motion (in blue in Figure 2.4(b)).

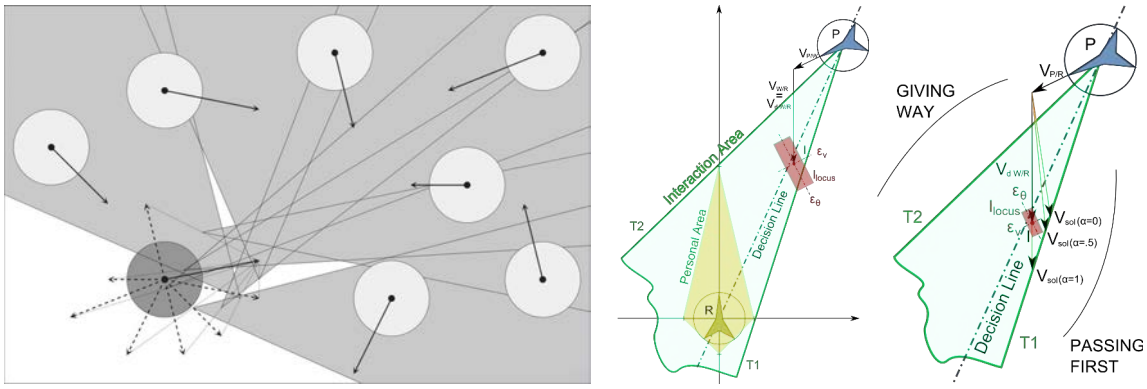
Finally, the agent chooses among these sections and speeds (combined they form the admissible velocities) those which minimize the velocity adaptation.



(a) Dividing orientations. (b) Dividing speeds.

Figure 2.4 – Agents divide their possible orientation changes (left) and corresponding speed changes (right) during collision avoidance [PPD07]. Purple discs represent the other agent's positions at diffirent moments in the future. The blue ares on the right represent admissible velocities four one section of possible orientations (each colored area on the left is such a section).

Later algorithms look for admissible velocities in a different way [vdBLM08, GCK+09, POO+09, GCLM12], and choose instead to reason in the two-dimensional velocity-space (see Figure 2.5), where admissible velocities can be quickly found amongst linear constraints.



(a) [vdBLM08] (b) [POO+09]

Figure 2.5 – Algorithms reasoning in velocity-space. Left: the main agent (dark gray) is looking for collision-free velocities (white areas) in velocity-space. Light gray areas represent inadmissible velocities leading to collisions with other agents. Right: two agents interacting from the perspective of the bottom left one, with the green area representing inadmissible velocities. The goal of that agent is to move its current velocity (in red) out of the green area to make it an admissible velocity.

2.2.3 Other Predictive Approaches

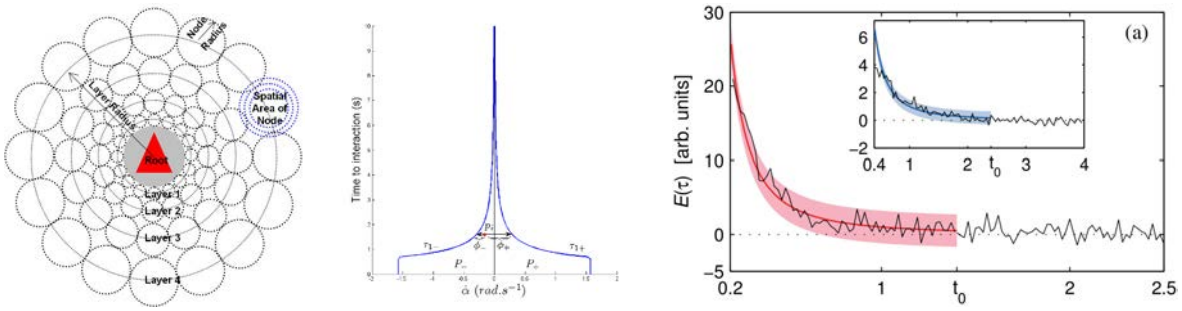
Finally, some other algorithms have also been proposed, using very different approaches but which are also predictive in nature; the remainder of this section details three such examples.

Kapadia and colleagues propose a model of ego-centric affordance [KSHF09], where space around an agent is discretized (Figure 2.6(a)) into cells containing information about neighbor agents visiting them in the future; collisions are then solved by choosing an orientation and speed which steer the agent towards the most neighbor-free cells.

Moussaïd and colleagues [MHT11] compute for every direction an agent could follow, a cost based on the time to collision (assuming the agent follows this direction at the current speed, and other agents continue with their current velocity) and the deviation from the destination. They then choose the direction with the lowest cost and adapt the speed of the agent so that the time to collision falls under a certain threshold.

A vision-based algorithm has also been proposed [OPOD10] which simulates agents' optic flow, and computes for each pixel, the time to closest approach (if there will be a collision, this would then be the time to collision) and the time-derivative of the bearing angle (the angle between the agent's orientation and a neighbor). Essentially, as the time to interaction decreases (but remains positive) and the derivative of the bearing angle stays close to zero, agents are headed for collision. Thus an agent steers as to keep these two quantities around certain preferred values (see Figure 2.6(b)).

In the last example [KSG14], agents are subjected to forces (computed based on linear extrapolations of each agent's velocity) which have been empirically defined by analyzing recorded ground-truth data, and finding a predictive *power law* which governs pedestrians' motions (Figure 2.6(c)).



(a) [KSHF09] (b) [OPOD10] (c) [KSG14]

Figure 2.6 – Other predictive algorithms. (a) Illustration of the egocentric discretization of space around an agent, future interactions are stored in these cells. (b) A trajectory is considered collision-free while values of time to closest approach and the time-derivative of the bearing angle stay outside the blue curves. (c) Interaction energy as a function of time to closest approach, extracted from a specific data-set.

2.3 Summary

As a common assumption, these second-order algorithms all linearly extrapolate agents' future motions from their positions and instantaneous velocities, making it possible to anticipate collisions up to a certain time horizon and improve simulations over first-order algorithms [OMCP12, GvdBL+12, WGO+14].

However this remains a linear extrapolation, which in many more challenging situations does not always yield truly satisfactory results, prompting additional work to

strengthen the quality of simulations. Kim et al [KGL+14]. introduced a probabilistic component to the algorithm presented in [vdBLM08], while Golas and colleagues [GNL13] added look-ahead to adaptively increase the time horizon in an efficient way for large groups of agents. Finally, this same algorithm has been extended to include agents' acceleration constraints [vdBSGM11]. All three aproaches have thus extended this algorithm to further improve results.

3 Centralized Algorithms

3.1 Cellular Automata

Cellular automata [Sch01, KS08] discretize the simulation space into a grid where each cell can be either free or occupied by one agent (or obstacle). In order to steer agents, these algorithms define rules dictating the probabilities agents have of transitioning into neighbor cells.

The simulation is then carried out as a series of *rounds* where all agents act in parallel. Each agent first chooses its preferred *exit* which is the neighbor cell it would attain following its preferred velocity during that round (Figure 2.7). Afterwards, agents attempt to transition into their preferred exits, while obeying all the rules they are subjected to.

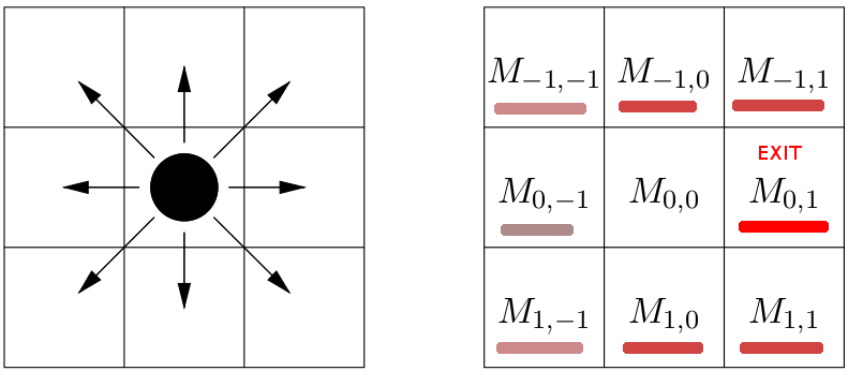


Figure 2.7 – Matrix of preferred transitions for an agent [Sch01].

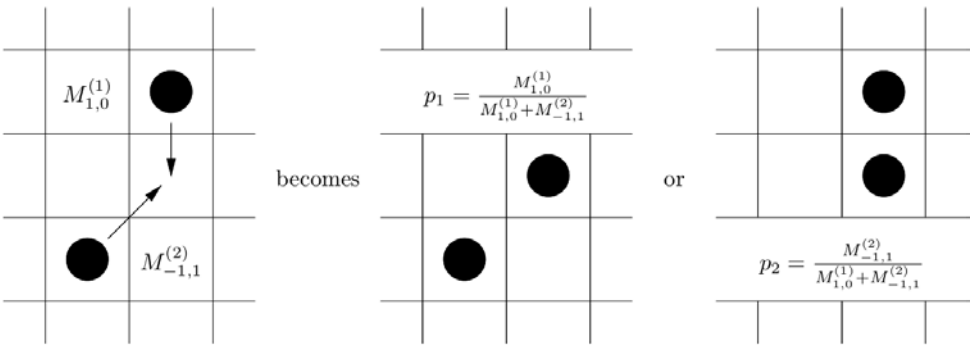


Figure 2.8 – Conflict between two agents, solved by comparing relative transiton probabilities [Sch01].

Transition rules can be local: an agent can't transition into a cell that is already occupied by another agent and if two agents were to attempt transitioning into the same cell, the one with the highest relative probability is given priority, thus solving extreme congestion cases (Figure 2.8). Rules can also be defined at a larger scale. For instance, it has been observed that two large groups crossing ways lead to the formation of lanes where pedestrians tend to follow the ones in front of them; similarly, pedestrians tend to follow trodden paths rather than unexplored routes. Cellular automata can easily be extended with floor fields to display these phenomena: by increasing agents transition probabilities to recently visited cells, lanes and previously paths will more easily be followed by subsequent agents. Additionally, these probability increases can be made to decay over time, thus allowing the patterns to change, and preventing the crowds from reducing to a few lanes.

Due to their probabilistic nature, cellular automata allow to evaluate possible outcomes of a given situation. Additionally, thanks to the simplicity of their governing rules, these algorithms scale well. This ability to scale well allows to study the coupling/transition from microscopic properties to macroscopic phenomena, such as the formation of diagonal patterns between two intersecting traffic flows [CARH13]. For these reasons, they are often used in the field of evacuation dynamics. Due to their discrete nature, these algorithms are not however well suited for computer graphics applications.

3.2 Data-Driven Algorithms

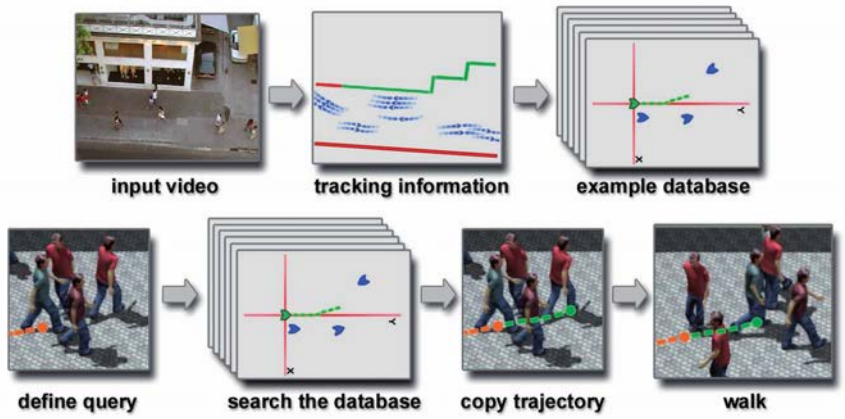
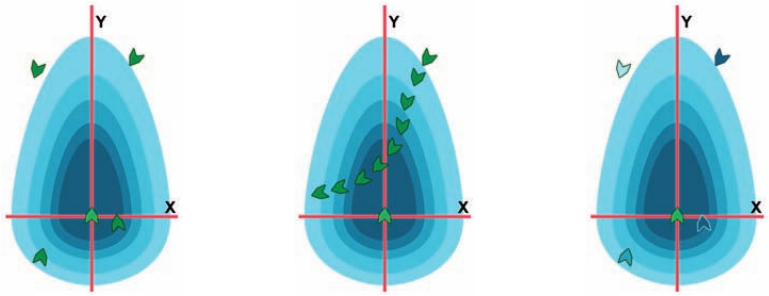


Figure 2.9 – Data-driven algorithms' usual workflow; from [LCL07].

Data-driven algorithms attempt to learn crowd behaviors from real-world sources such as motion capture or camera footage, hugely benefitting from recent advances in tracking. With large data-sets of real-world pedestrian trajectories available to them, these methods construct data-bases of motions which they then query to synthesize realistically behaving crowds. This usual workflow of data-driven algorithms is shown on Figure 2.9.

In their work, Lerner at al. [LCL07] aim to determine what influences (and how) pedestrians' trajectory decisions. Their data-base then consists of pedestrian-interaction examples (see Figure 2.10). Each example focuses on one *subject* pedestrian and captures

a short moment around a time t. Examples have two components: a portion of the trajectory of the *subject*, as well as a portion of the trajectories of nearby pedestrians (and other dynamic/static gemoetry such as walls). As seen on Figure 2.10, examples are centered on the *subject* and all information is recorded in this referential, additionally, each point in this referential has a certain influence value (colored areas in Figure 2.10). Each influencing pedestrian (and geometry) is given an influence value corresponding to the highest influence value along their trajctory portion. At run-time, for every agent, the simulation algorithm computes a query and thanks to a similarity function using the influence values, finds the closest example. The trajectory of the *subject* in the closest example then determines how the queried agent will behave in the simualtion.



(a) (b) (c)

Figure 2.10 – An example as defined in [LCL07]. (a) Configuration of an example, with the *subject* in the center and nearby influencing pedestrians. Colored areas represent various levels of influence (darker is more influent). (b) The top-right pedestrian's trajctory portion, the most influential position determines the pedestrian's influence. (c) All pedestrians colored according to their influence on the *subject*.

Other data-driven algorithms follow similar principles. Lee et al. [LCHL07] construct data-bases of state-actions which record information about the *subject* pedestrian (state: such as speed, neighbors, pivots aka. special objects near which special behaviors take place etc.) and what the pedestrian did. Again, at run-time, this data-base is queried to determine agents' trajectories.

Ju et al. [JCP+10] developed an algorithm which learns crowd formations (as distributions of the four closest neighbors' positions) and motions (as trajectory segments) in order to learn different crowd styles. As a result, it is possible to interpolate between different models, thus leading to visually heterogeneous crowds as seen on Figure 2.11.

More recently, Charalambous and Chrysanthou [CC14] constructed their data-base in the form of a graph. In this graph, nodes represent the current state of an agent, defined as a series of temporally consecutive visibility patterns (free space in an agent's field of vision), and edges are actions which allow to transition from state to state. Thanks to this graph structure, run-time queries are easier (the query result is very probably in the agent's current node) thereby reducing the computational cost as compared to previous algorithms. Additionally, since agents traverse the graph with limited jumping between non-connected nodes (unless they drift too much from the states stored in the current node), their behaviors are more consistent.

Overall, data-driven algorithms are a great method to populate areas. Crowds with

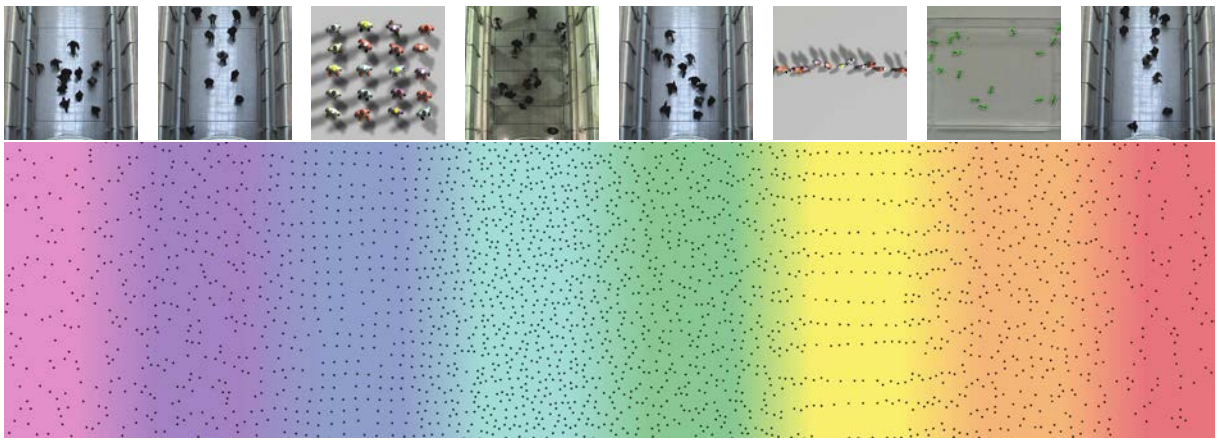


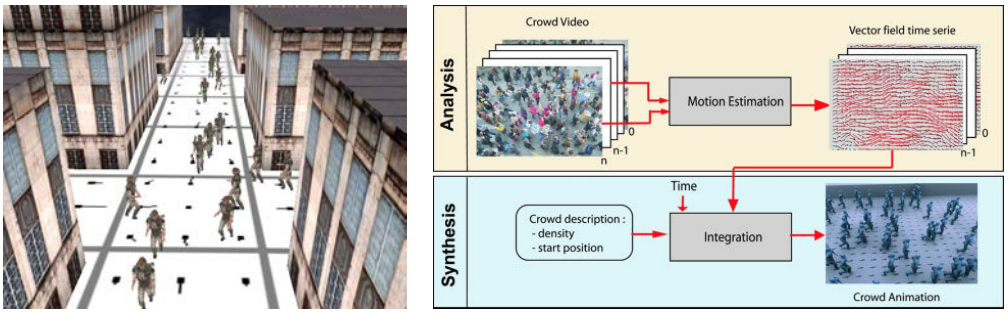
Figure 2.11 – Heterogeneous crowd obtained by interpolating between various styles [JCP+10].

specific styles as well as artist-generated ones can be recorded and used to build the data-bases allowing to synthesize a wide variety of crowds. Additionally, crowds can be randomly initialized (or with some guidance as in [JCP+10]) and the agents would continue moving according to the learned behaviors seemingly indefinitely without the need for the animator to specify individual goals for each agent. But this lack of explicit goal-management makes agents more difficult to control during run-time. Moreover, the resulting simulations heavily depend on the quality and situation-coverage of the source data. Obviously, situations which are not in the data-base cannot be reproduced and the algorithms can have a hard time when the simulations start straying away from what is learned. Finally, although [CC14] improved run-time performance, these algorithms remain computationally expensive, and are thus better suited for non-interactive applications.

3.3 Tiles and Patches

Tile-based algorithms essentially allow to precompute motions. Using this approach, a virtual environment is assembled from a library of tiles which contain velocity information, or flows. During simulation, pedestrians are modeled as particles which are advected along these flows, resulting in simulations with a low computational cost. The first technical question then revolves around the construction of these tiles. For instance, Chenney proposes a method [Che04] to create divergence-free tiles, respecting boundary conditions imposed by the environment (Figure 2.12(a)), while Courty and Corpetti propose to learn the flows from video data [CC07](Figure 2.12(b)).

Since all particles (simulated pedestrians) are advected following the same flow fields, these algorithms forgo local collision-avoidance mechanisms. In theory, if particles are initialized as non-intersecting, and allow a certain buffer distance between themselves accounting for the possible compression resulting from the flow fields, collisions should be avoided. In practice however, especially at higher densities collisions do occur. Additionally, the particles have no individual goals and it is hard to simulate intersecting flows (unless explicitly stacking tiles on top of each other, applicable to specific particles,



(a) [Che04] (b) [CC07]

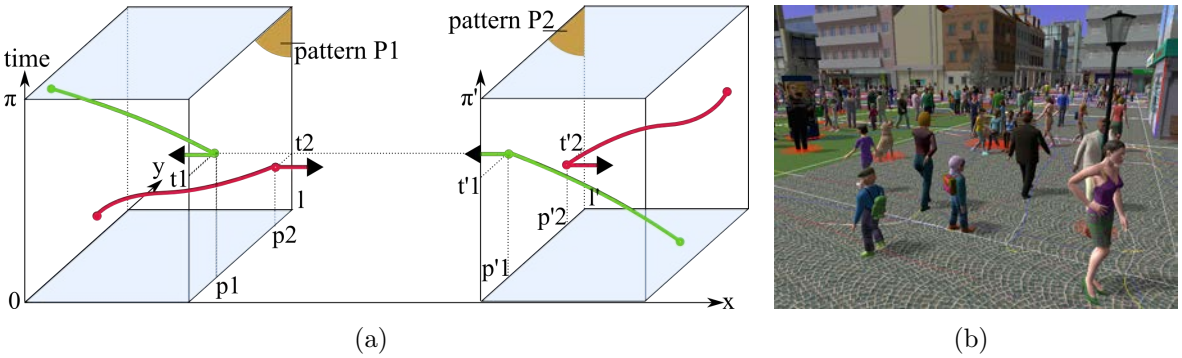
Figure 2.12 – (a) Virtual pedestrians walking on tiled streets in a city. (b) Overview of a data-driven tile-based crowd simulation algorithm.

as stated in the future work for these algorithms).

Consequently, Patil and colleagues propose to use a unified solution [PvdBC+11] combining a global planner, flow tiles and a local collision-avoidance algorithm. As a result, each agent can select a route along the flow fields (where two opposing flows can intersect) and follow the flow fields, with collisions being solved by the local collision-avoidance algorithm.

While tile-based algorithms create "shared tracks" which advect simulated pedestrians, patch-based algorithms take this principle further by creating a track per pedestrian and later assembling them. Thus, Kim et al. build motion patches capturing actors' interactions (hand-shaking, carrying objects etc.), segment them and assemble them during simulation to create crowds of interacting characters [KHHL12]. While these whole-body motion patches allow to create crowds of interacting characters, the assembly of patches remains computationally expensive, especially if one wants to keep extending the synthesized scenes. Yersin and colleagues solve this problem [YMPT09] by defining patches which contain periodic character trajectories. By building a library of patches which have matching periodicity and boundary conditions, they are able to effortlessly and endlessly combine patches in all directions, while guaranteeing that the simulation will continue playing forever thanks to its periodicity (Figure 2.13).

The main advantage of tile- and patch-based algorithms is their ability to populate virtual areas with large crowds obeying easily controllable pedestrian flows, all at a lesser computational cost during simulation. This property comes at a price however, which is the static nature of the resulting simulations. As all simulated pedestrians are "on tracks" and have no individual navigation (except in the work found in [PvdBC+11] where the cost increases due to the local collision-avoidance algorithm), the synthesized crowds are non-interactable. Furthermore in the case of tile-based algorithms (and the method from [KHHL12]) on the one hand, edits made to the environment require a re-computation of the tiling, at least to solve any boundary conditions, divergence or singularities that may arise. For the algorithm from [YMPT09] on the other hand, while patches can easily be changed by picking other patches with matching boundary conditions, the efficient (pre-)computation of these patches remains difficult.



(a) (b)

Figure 2.13 – Crowd patches algorithm [YMPT09]. (a) Example of two patches (one period of animation) that can be assembled thanks to matching boundary conditions. (b) Scene assembled from Crowd Patches.

3.4 Macroscopic Algorithms

Crowd simulation algorithms based on fluid-dynamics aim to control very large crowds at the global - or macroscopic - level. Treating the crowd as a flowing continuum, these methods rely on continuous equations, defining flows that the agents (modeled as particles) then follow.

In an extension to the work found in [Hug03], Treuille et al. [TCP06] simulate large, moderately dense crowds of particle-like agents. This method works by combining several fields (discretized into grids), such as goals, speed, density etc. and using the resulting potential field to move agents (see overview Figure 2.14). Inter-agent interactions are solved thanks to the density field: when the density increases it dominates the field equations, thereby pushing agents away from each other (note that this is only ensured down to the level of the grid, meaning that two agents in the same cell can intersect).

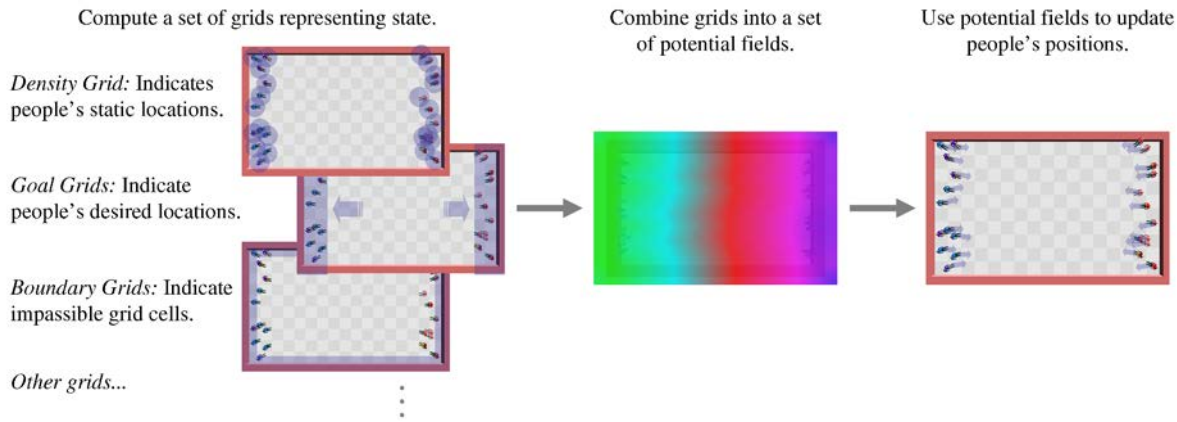


Figure 2.14 – Overview of the method from [TCP06].

To improve the agents' local behavior, Narain et al. [NGCL09] introduced a hybrid method combining fluid-dynamics and a local collision-avoidance method. For each

agent, a global planner determines the desired velocity (direction and speed), this velocity is then tranformed along the agent's position into flow information contained in a grid. Compressibility constraints are then applied to this flow, so that the crowd is not compressed beyond a threshold. Finally, the corrected flow information is transferred in the form of updated velocities to the agents where pair-wise interactions are solved by the local collision-avoidance algorithm. Figure 2.15 shows an overview of this algorithm (left) as well as a resulting crowd (right).

These simulators result from a "top-down" process: fluid dynamics-like equations are used to move aprticles, thus defining the local behavior. Some work has also been done from the opposite perspective, as a "bottom-up" process where macroscopic laws are derived from microscopic algorithms (e.g. [BMP11, DARM+13, DARPT13] repectively derive macroscopic models from (1) cellular automata, and algorithms from (2) [MHT11] and (3) [OPOD10]). This type of derivation can then be used to analyze various aspetcs of crowd motion, such as the formation of clusters of agents for instance.

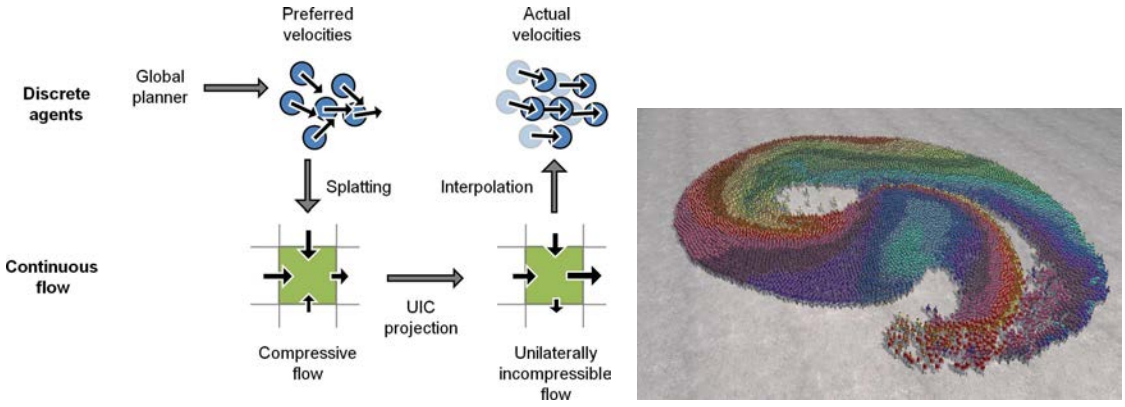


Figure 2.15 – Algorithm from [NGCL09]. (a) Workflow overview. (b) Resulting crowd.

As they steer crowds as a whole and with simplifying assumptions such as shared goals, these macroscopic crowd simulation algorithms scale very well (Figures 2.15(b) and 2.16(a)) into the tens of thousands of agents at interactive frame-rates. Additionally, the resulting simulations seem realistic at the global level and allow the emergence of patterns (e.g. formation of a vortex for four groups crossing ways, Figure 2.16(b)). They also allow controlling a crowd's density and discover related phenomena such as bottlenecks (Figure 2.16(c)). However, these algorithms rely almost exclusively on terrain/environment cues to determine the motion of the agents. Additionally, agents are mostly modeled as particles carried by the flow. As a result, agents can have erratic trajectories (going forward/backward and sideways, equally), complex interactions between individual agents are lacking at the local (or microscopic) level, and individual trajectories have little impact on the crowd.

(a) (b) (c)

Figure 2.16 – Crowds generated with macroscpic crowd simulation algorithms. (a) A large army chasing another one. (b) A vortex forms as four groups cross ways. (c) Office evacuation simulation.

4 Conclusion

The first conclusion is that the pool of available algorithms is very large, and the algorithms themselves can vary significantly. Each category of algorithms results from a certain need: macroscopic algorithms allow to simulate large crowds observable from a distance/as a whole, tiles and patches allow to easily direct agents with low performance impacts, data-driven algorithms allow to populate areas by "copy-pasting" crowds, cellular automata allow to simulate individual agents for statistical purposes, while agent-based algorithms are used for their versatility and general ease of use.

Overall, among all these types of simulation algorithms, each with their strengths and preferred application domains, agent-based algorithms remain the most popular, thanks to their ease of implementation and use but also their flexibility, being intuitively extendable with additional interaction rules and scripting. Agent-based algorithms also yield interactive crowds with consistent levels of realism at all scales: local interaction rules allow to convincingly simulate local behaviors while the resulting crowds display emergent global phenomena observable in real life.

However, despite all the available agent-based algorithms and successive improvements, some artifacts still persist in certain scenarios, such as: slowdowns and visually erroneous agent agglutinations, unnatural oscillation motions, or exaggerated/last-minute/false-positive avoidance manoeuvres. Furthermore, one can observe that there are many different simulation algorithms, and that each simulator has its own strengths and weaknesses. To quantify these differences and be able to further develop and improve on existing work, one needs a robust evaluation process. The next chapter (Chapter 3) deals with this topic: after a review of previous work on evaluation in Section 2, we propose a general evaluation framework, which served as the first step in the work presented in this document.

Craal: Parameter Estimation and Comparative Evaluation of Crowd Simulations

3

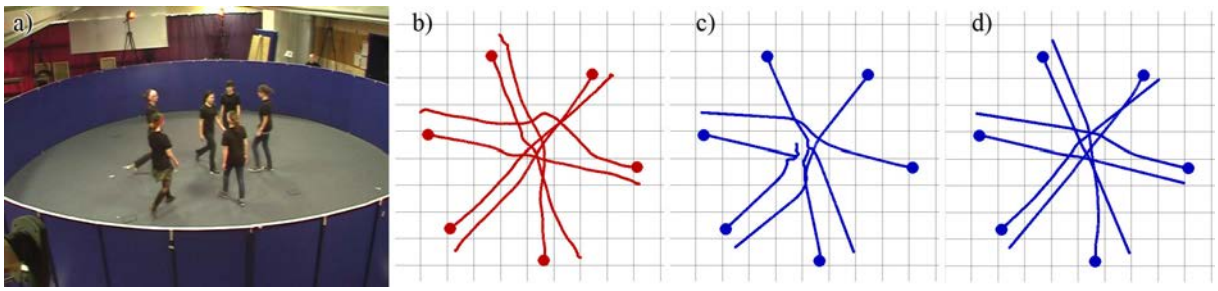


Figure 3.1 – Parameter Optimization Applied to Crowd Data (a) motion capture session for recording reference trajectories for six human agents (b) reference data plot (circles are initial positions) (c) paths taken by simulated agents with default parameters (d) paths taken by simulated agents with optimized parameters. The stock parameters of a simulation model often do not match closely with actual paths humans take in the same situation. Using our parameter optimization technique, the resulting simulation can be made to better match the human trajectories.

Contents

1 Introduction

Creating simulation models of crowds has recently received considerable attention in computer animation, pedestrian dynamics, and virtual reality. As already mentioned in the previous chapter (Chapter 2), many approaches have been investigated that suggest different techniques to simulate crowds, and a variety of simulation algorithms are known in the literature. These include multi-agent simulation algorithms that are widely used in computer games, virtual reality, animation, and pedestrian dynamics.

A key research issue in this area is to perform a formal or rigorous evaluation of these algorithms. One widely used criterion is to perform comparative evaluation of simulation algorithms against some real-world reference datasets. However, a major challenge is to estimate the best set of parameters for a given algorithm that would result in the *optimal match* with the reference data.

The issue of optimal parameter selection is critical, because most of existing crowd simulation algorithms depend on various parameters and the resulting trajectories or behaviors can vary noticeably based on the choices of parameters. There is no standard way to make comparative evaluation of simulation algorithms. At the same time, data capture of real-world human crowd motion is becoming increasingly ubiquitous. Such datasets can in fact help in describing and analyzing specific crowd phenomena, as well as in calibrating and evaluating crowd simulation models. Given the increase in the number of crowd simulation algorithms and real-world datasets, we need rigorous and automatic techniques to evaluate them.

In this work, we present a novel framework that can be used to evaluate different crowd simulation algorithms against reference datasets. In this context, we address the problem of computing optimal parameters for a crowd simulation algorithm and present a general scheme that is applicable to a broad class of algorithms and reference datasets. We formulate the evaluation of a simulation algorithm as an optimization problem. First, we find a set of parameters that enables the best match between each simulation algorithm and the reference data. Second, we compare the objective function scores (i.e., distance to reference data) for the given set of algorithms. Our framework is general and capable of supporting a wide range of comparison metrics and simulation techniques.

We illustrate the benefits of our evaluation framework over several existing multi-agent crowd simulation algorithms. Moreover, we consider heterogenous types of reference datasets: recorded individual trajectories, macroscopic quantities, or even animation sketches. We gather a set of relevant metrics to compare simulated crowds with reference data. We highlight the benefits of parameter estimation by demonstrating its application to example-based simulation and behavior modeling with cultural variation. Our framework is available as an open-source package and can be used by others to evaluate different simulation algorithms and metrics. We demonstrate its performance on many widely-used multi-agent simulations and consider different scenarios with a varying number of agents. In our benchmarks, we observe that velocity-based crowd simulation algorithms (e.g. RVO2, [POO+09], etc...) result in lower errors as compared to techniques based on Boids or social forces.

The rest of the chapter is organized as follows. In Section 2, we give an overview

on related work in parameter calibration and crowd simulation algorithm evaluation, as a complement to the related work on crowd simulation algorithms from Chapter 2. Section 3 describes our parameter estimation framework and its key components: algorithms, metrics, reference data, and optimization techniques. A wide range of concrete examples and applications are presented in section 4 to demonstrate the benefits of our solution. Finally, we conclude in Section 5.

2 Related Work

The necessity of evaluating crowd simulation algorithms arises both in the case of a prospective user when choosing an appropriate simulator, and in the case of researchers and developers when justifying the development of a new one.

The main observation from related work in this aspect is that evaluation is fragmented across many works and that, while some efforts have been made, no standardized benchmarks are defined and widely accepted in the field of crowd simulation. The reason for this lack of standardized testing is the variety of applications for crowd simulators, leading to the emergence of a wide range of works focusing on very different aspects: by necessity the amount of test cases needs to be large and not all test cases always apply. Section 2.1 summarizes existing evaluation approaches.

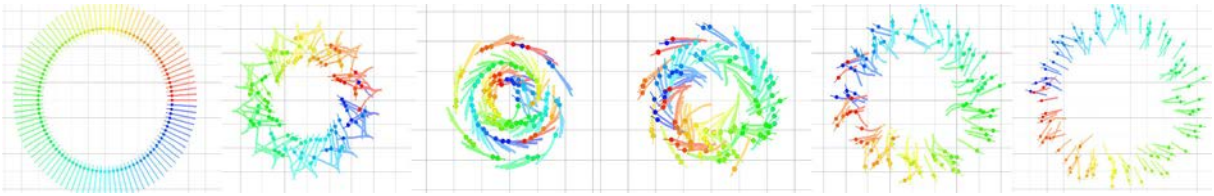
In addition to the lack of a standardized evaluation scheme, one can observe the lack of a standardized evaluation protocol. Mainly, the issue of the simulators' parameters is rarely taken into account. Thus, resulting comparisons between algorithms can in theory be contested, as "some parameter tweaking" might considerably change the simulations and consequently the comparison results. Related work on the issue of parameters is presented in Section 2.2

2.1 Evaluation

In this section we present related work on evaluation, first dealing with the fragmented approaches to this issue and then moving onto attempts at defining standardized testing schemes or benchmarks.

The most direct form of evaluation is the qualitative observation of the simulated crowds. Thus, a number of scenarios are typically chosen to display the ability of a simulator to reproduce known emergent crowd behaviors (Figure 3.2). Prominent examples of such behaviors include the circular aggregation of pedestrians around a door/passage, a flow of pedestrians passing throught such passages, and the formation of lanes between two or more flows of pedestrians [HM95, HFV00, PPD07, OPOD10, GCLM12]. In contrast to these scenarios, which reproduce situations that are readily observable in the real world, other (more artificial) scenarios have subsequently been designed to test how agents interact. In particular, one can force simulated pedestrians to deal with increasing numbers of neighbors by arranging them into a circle and then setting their destinations as the antipodal positions on this circle, effectively maximizing the number of exected interactions. This specific situation, while rarely observed in the real world (and thus lacking ground-truth observations), allows to look for collisions between pedestrians in progressively changing density conditions [vdBLM08, GCK+09, OPOD10].

(a) [HFV00] (b) [GCLM12] (c) [PPD07] (d) [OPOD10]



(e) [OPOD10]

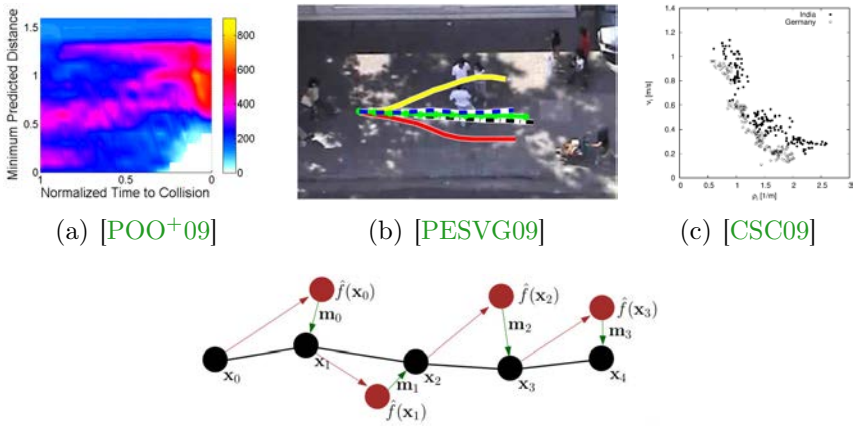
Figure 3.2 – Examples of test scenarios. (a)-(b) Examples where pedestrians are known to agglutinate in a circular way around a door/passage. (c) Pedestrians traversing a funnel-shaped passage. (d) Lanes forming between two opposing flows of pedestrians. (e) Circle situation where pedestrians attempt to reach the antipodal positions.

In addition to these typical scenarios, some works have also proposed quantitative evaluations by comparing simulated pedestrians' trajectories to ones recorded on humans [PPD07, POO+09, PESVG09] (Figures 3.3(a) and 3.3(b)). In these cases, metrics included differences between simulated and recorded pedestrians and the minimum predicted distance (distance between two interacting pedestrians when they will be closest to each-other). Guy and colleagues further developed a more complex *entropy* metric which compares decisions taken by simulated and recorded pedestrians at each step of their trajectories, resulting in a statistical similarity measure between ground-truth and simulation (Figure 3.3(d)). Finally, the *fundamental diagram* (speed of agents in a flow as a function of their surrounding density) can be used (from the field of pedestrian dynamics and evacuation) to evaluate simulated crowds (Figure 3.3(c)).

Finally, Singh and colleagues introduced a framework [SKFR09] as an attempt to standardize the evaluation of crowd simulation algorithms. This framework contains a collection of situations destined to cover the range of possible, expected simulation situations (with the question of coverage being explored more in [KWS+11]). It also contains a collection of metrics measuring for instance the time an agent takes to reach its goal, the energy that has been spent in doing so etc. Similarly, Lerner and colleagues [LCSCO09, LCSCO10] propose a system where a data-base is constructed from trajectories extracted from video clips and a set of metrics is used to rate simulated crowds.

2.2 Parameters

When using crowd simulation algorithms, one must keep in mind the algorithms' parameters, which can range from common ones, such as agent radius or time horizon (if the algorithm is predictive), to algorithm-specific ones such as force magnitude (Social-Forces [HM95] algorithm and similar), various thresholds (e.g. for the time derivative of



(a) [POO+09] (b) [PESVG09] (c) [CSC09]

(d) [GvdBL+12]

Figure 3.3 – Examples of ground-truth comparisons. (a) Metric: minimum predicted distance. (b) Comparison of real and simulated trajectories. (c) Fundamental diagram (in this case, difference between German and Indian people for single-line following). (d) The entropy metric compares possibly erroneous simulation decisions (red) to recorded positions (black) at every step.

the bearing angle in the case of the vision-based algorithm [OPOD10]) etc. These parameters are very important as they can have a large impact on the resulting crowds, for instance the wrong value for the force magnitude in the Social-Forces algorithm could lead to agents either colliding (too low value) or walking too far apart/being unstable (too high value). Consequently, this issue greatly influences the evaluation of simulation algorithms, as it is natural to select values for simulators' parameters that would lead to the best possible results.

The most direct way of setting parameter values is via manual tweaking: the user repeatedly uses the simulation algorithm on various scenarios and gradually settles on values which lead to mostly satisfactory results (either by plotting results or by rule of thumb) [OPOD10]. Paris and colleagues [PPD07] manually set parameter values by checking their results on ground-truth data of two recorded interacting human pedestrians. Karamouzas and colleagues [KHBO09] plotted results of their algorithm on the various scenarios from the framework presented in [SKFR09] and set their values accordingly.

However, this process of manual tweaking is very tedious and some works have automated this process. Pellegrini and colleagues [PESVG09] (seeking to apply their algorithm to tracking of humans from video) "train" their algorithm on recorded trajectories using gradient descent and a genetic algorithm. Pettre and colleagues [POO+09] set their parameter values using Maximum Likelihood Estimation based on recorded trajectories of two interacting humans.

More recently, Karamouzas and colleagues [KSG14] derived their algorithm from the power law they had found from collections of real-world data.

2.3 Discussion

Overall in terms of evaluation, one can observe the emergence of test cases commonly used to validate simulation algorithms (such as opposing flows of pedestrians, doors/passages etc.) as well as phenomena commonly sought in simulation results (lanes, vortices etc.). Following this trend, a framework has also been proposed regrouping many such test cases and metrics. It is difficult however to quantify how realistic analgorithm is based on these examples alone. Thus, some works have proposed using real-world in the form of pedestrian trajectories or other statistics (e.g. fundamental diagram). One can also observe that parameters play a role in simulation results (and thus evaluation) as they have been explicitly dealt with in a number of works.

However, one can also observe that, there is no unified evaluation scheme:

- based on real-world data, or more generally, arbitrary data
- independent of the simulation algorithm
- involving *automatic* parameter estimation

3 Optimization Framework

Our framework brings together three components: crowd simulation algorithms, reference data, and a set of metrics for measuring how well the simulation algorithm matches the data. This section provides the details of our parameter optimization algorithm and gives an overview of different simulation models, data sources, and comparison metrics used in our evaluations. We also give a brief summary of the global optimization techniques used in our work. Additional information, including a more detailed description of our metrics and optimization algorithms as well as a comparison of these optimization algorithms can be found in Appendix A.

3.1 Overview of Approach

We define a crowd simulator as an algorithm which takes a collection of agent states (i.e., positions and velocities of agents in the crowd) and produces a new set of agent states representing the movement of the crowd over a timestep Δt. We introduce the following notation for specifying a crowd simulation algorithm: let k be a given timestep, then $\mathbf{x}_k$ will represent the positions of all agents at timestep k, $\mathbf{v}_k$ the velocity of all agents, and $\mathbf{g}$ the goals of all agents. We can then formally define a crowd simulation algorithm f as follows:

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ \mathbf{v}_{k+1} \end{bmatrix} = f(\mathbf{x}_k, \mathbf{v}_k, \mathbf{g}). \tag{3.1}$$

In general, a crowd simulation algorithm may have several tunable parameters that affect the behavior of an agent computed by the simulator. Common examples of parameters include an agent's preferred speed or some notion of personal space. While the exact

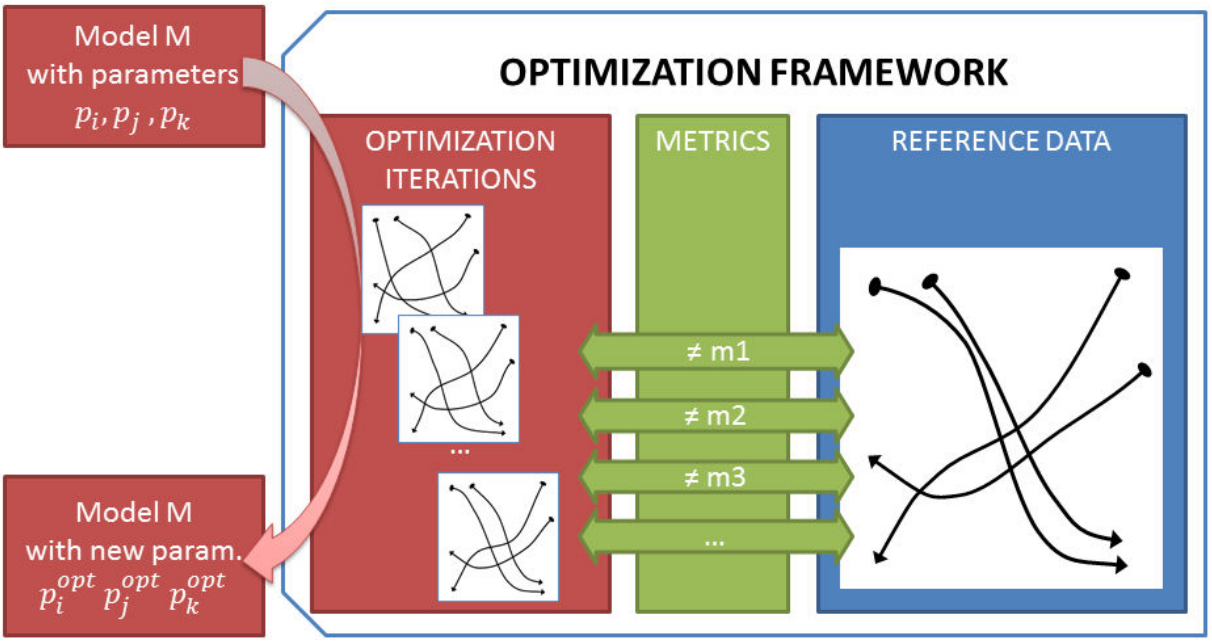


Figure 3.4 – System Overview Our approach optimizes simulation parameters to match target data. Our framework has 3 components: an optimization technique, metrics, and reference data.

nature of the agent parameters are specific to each algorithm, our framework assumes that these parameters can be defined separately for each agent. Given an agent i, we use $\mathbf{p}_i$ to denote the current parameter set for that agent, and $\mathbf{p} = \{\mathbf{p}_1 \cdots \mathbf{p}_n\}$ to denote the vector of parameters over all n agents.

We can now introduce the notion of a *Parameterized Crowd Simulation* as an algorithm f where crowd parameters are part of the input. Formally, for each timestep k:

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ \mathbf{v}_{k+1} \end{bmatrix} = f(\mathbf{x}_k, \mathbf{v}_k, \mathbf{g}, \mathbf{p}). \tag{3.2}$$

Simulation Models Several common crowd simulation models fit the form described by Equation (3.2). We focus on five widely used agent-based simulation algorithms:

1. In the *Boids* model [Rey99], f is a function of the agents' position at some specified future time (current time plus constant). When the predicted distance between agents gets too low, a separation force is computed and added to an attraction force which is pulling towards the agent's goal. Parameters are: radius (size of 2D circle agents) and comfort speed (i.e., speed when no interactions occur).

2. In the *Helbing* Social Force model [HFV00], f is a function of the agents' positions. Repulsive forces are computed between agents and combined with attraction forces toward goals. Parameters are: radius and comfort speed.

3. In the *RVO2 model* [vdBLM08], which computes an agent's admissible velocity space (space which remains collision-free in a future time window), f returns the

optimal admissible velocity. Parameters are: comfort speed, neighbor distance (only agents within this distance are considered for local interactions), radius, and time horizon (only future collisions within this horizon are considered for local interactions).

4. In the *Synthetic Vision* model [OPOD10], which is based on principles from human cognition and visual navigation, f is a function of perceptual variables derived from synthetic optic flow. Parameters are: comfort speed and (a, b, c), which define a threshold function; perceived values under this threshold are considered for local interactions.

5. In the *Tangent* model [POO+09], which works in the velocity space and considers possible perception errors, f returns the optimal admissible velocity. Parameters are: comfort speed, radius and two error-quantifying parameters.

The parameter set $\mathbf{p}$ for each of these models can be found in Appendix A Section 4.

Given a parameterized crowd simulation (Eqn. (3.2), our goal is to find a parameter set $\mathbf{p^{opt}}$ which leads to the closest match between a model and some user-defined reference data, which can vary per timestep $\mathbf{z}_k$. Over all timesteps m, we can define the reference data as follows:

$$\mathbf{z} = \bigcup_{k=1}^{m} \mathbf{z}_k. \tag{3.3}$$

In the same way, we can define a complete simulation as all states of a simulator initialized with the reference data:

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix} = f(\mathbf{z}, \mathbf{p}) = \bigcup_{k=1}^{m} f(\mathbf{x}_k, \mathbf{v}_k, \mathbf{g}, \mathbf{p}), \tag{3.4}$$

initialized with $\mathbf{x}_1 = \mathbf{z}_1$, $\mathbf{v}_1 = speed(\mathbf{z}_1)$ and $\mathbf{g} = \mathbf{z}_m$.

Given this complete simulation and reference data, plus some user defined distance metric, $dist()$, we can formally define our framework as computing

$$\mathbf{p}^{opt,f} = \underset{\mathbf{p}}{\operatorname{argmin}}\, dist(f(\mathbf{z}, \mathbf{p}), \mathbf{z}), \tag{3.5}$$

where $\mathbf{p}^{opt,f}$ is the parameter set which matches the reference data closest for a given simulation method f.

In general, the optimization problem that we propose in Equation (3.5) is very high-dimensional ($dim(\mathbf{p}) = \sum_{i=1}^{n} dim(\mathbf{p}_i)$), making it difficult to optimize consistently across a wide range of similarity metrics, reference data, and simulation methods. We describe our optimization algorithm in Section 3.3, which is designed to deal with such high-dimensional problems.

Once an optimal parameter set $\mathbf{p}^{opt}$ has been computed, we can fairly compare two different simulation methods, f_1 and f_2, by examining their optimal distance from the reference data. Formally, we declare simulation method f_1 better than simulation method f_2 if and only if:

$$dist(f_1(\mathbf{z}, \mathbf{p^{opt,f_1}}), \mathbf{z}) < dist(f_2(\mathbf{z}, \mathbf{p^{opt,f_2}}), \mathbf{z}). \tag{3.6}$$

3.2 Optimization Metrics

The role of reference data is to provide a description of the desired behavior or motion trajectories that the simulation should generate. This data can either come from measurements of real motion (e.g., from an overhead camera or motion-capture devices) or can be generated. Generated data can come from artists (flow fields) or some other high-level simulation algorithms. The function *dist*() in Equation (3.5) should capture how close a simulation state comes to matching the reference data. The exact representation of *dist*() depends on the nature of the reference data and on the features of the data which the user considers most salient for his or her application.

At a high level, there are two fundamentally different types of reference data that can be used in Equation (3.5):

- *microscopic data*, which specifies the exact trajectory of each agent in the data, and
- *macroscopic data*, which describes aggregate measures of the overall crowd motion.

Below we briefly describe various metrics that can be easily used in our framework, a more detailed description as well as their mathematical representation can be found in Appendix A Section 1.

3.2.1 Microscopic Data Metrics

- *absolute difference metric (D)* computes the total distance in position over all agents over all timesteps,
- *path length metric (L)* compares the difference in total length traveled between agents in the reference data and the simulated agents,
- *inter-pedestrian distance metric (I)* compares the difference in average distance (as a 2-norm) between every pair of agents,
- *progressive difference metric (P)* measures the absolute difference between the simulated agents and the reference data *when the simulation is reinitialized at each timestep.*

3.2.2 Macroscopic Data Metrics

- *vorticity metric (V)* measures the vorticity (as defined in fluid mechanics) of the crowd flow,
- *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]).

3.3 Optimization Techniques

Once a user has chosen the reference data and an appropriate optimization metric, Equation (3.5) can be optimized using different combinatorial optimization algorithms. Because several parameters need to be chosen for every agent in the simulation, the result is a very high-dimensional search problem (hundreds of dimensions) and the complexity of finding an optimal solution is very high. We have analyzed three different widely-used optimization techniques that can be applied to these high-dimensional search spaces.

Each optimization technique uses a different strategy to sample plausible parameters for each agent, maximizing the match of the simulation algorithm to the reference data. All three methods proceed by choosing perspective values for the per-agent simulation parameters from a user-specified domain of reasonable values. A list of the parameter distributions used in our experiments is given in Appendix A Section 4.

3.3.1 Greedy approach (G)

This approach works by replacing one parameter from $\mathbf{p}$ at a time for each agent. If this replacement lowers the optimization function, the new parameter value is chosen; if not, the previous value is restored. This method can get stuck in local minima.

3.3.2 Simulated annealing (SA)

A variant of the greedy approach, this approach attempts to avoid local minima by occasionally using new parameter sets that are "worse" (have higher value of the optimization function) than the old ones [KJV83]. The likelihood of accepting a worse parameter set decreases over time. Given an unlimited amount of time, SA will compute the global minimum.

3.3.3 Genetic algorithm (GA)

Methods based on genetic algorithms also seek to avoid local minima, and do so by maintaining a pool of parameters that can lead to different local minima [Hol92]. New pools of parameters are computed by combining and modifying previously successful candidates.

3.3.4 Covariance Matrix Adapation (CMA)

A solution-pool based method [HO96] similar to GA, which generates new solutions from ditributions defined by a covariance matrix that is adapted at each iteration.

After a comparison in terms of convergence and time of convergence, a combination of genetic and greedy algorithms (GA+G) has been chosen as offering the best compromise between score optimization and runtime performance. While there is no estimation of how close they come to the real optima, the simulated annealing method should provide a good indication as it can theoretically find a global optimum given an infinite computation time. More results on this comparison can be found in Appendix A Section 3; pseudocode for the above methods can be found in Appendix A, Section 2.

Other global optimization techniques, such as Particle Swarm Optimizations (PSO) [PKB07] or the adjoint method [MTPS04], can be applied to optimize Equation (3.5). Multiple methods can be combined or applied sequentially.

4 Results

The primary benefit of this framework is its generality: it can automatically find the best parameters for any simulation algorithm, based on any metric, for any reference data. In this section, we highlight some advantages and benefits of our parameter-optimization and the framework. First, we present different types of ground-truth data, which are a basis for model comparison, along with examples of model comparisons for each type. Second, we present scores obtained by our implementations of the Boids-like, Social-force and RVO2 models, in the form of a benchmark chart and its analysis.

4.1 Data Categories

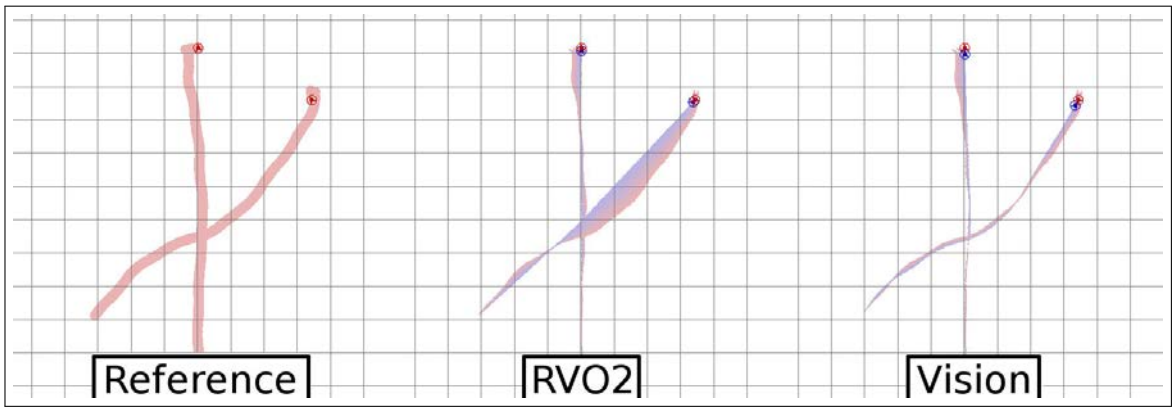
4.1.1 Microscopic data

Microscopic data, 2-5 agents This data category regroups various cases of 2-5 pedestrians crossing ways. The following are two visual examples showing model comparisons based on this data using the Difference metric:

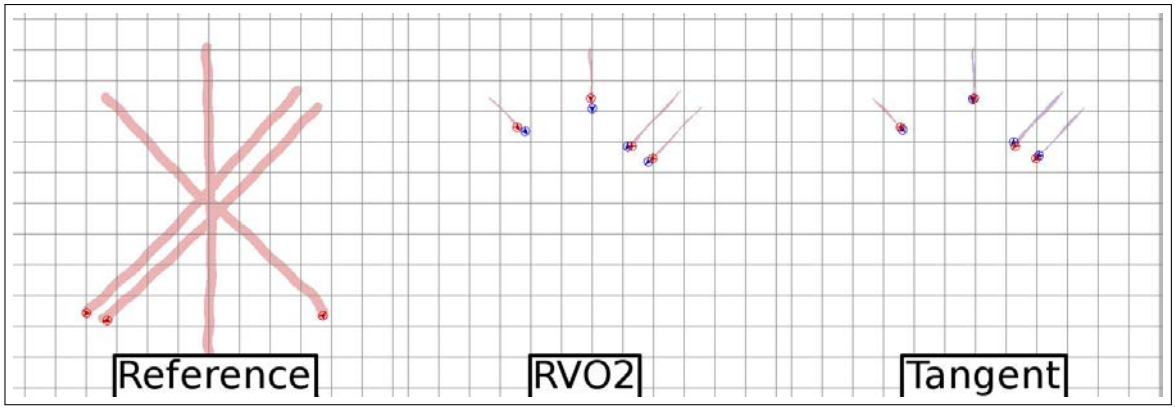
- a simple crossing scenario between two pedestrians. With default parameters, none of the algorithms correctly replicate the avoidance strategy of the pedestrians; after calibration, only the Vision-based algorithm can reproduce it (Figure 3.5(a)). This is consistent with its goal, which is to reproduce human-like reactions to impending obstacles.
- a scenario where trajectories are correct but the agents are ill-synchronized with the real pedestrians, i.e. their speeds along the trajectories are incorrect. After calibration, the Tangent simulation model [POO+09] gives a better speed profile than the other algorithm, because its agents accelerate and decelerate when needed and are better synchronized with pedestrians (Figure 3.5(b)).

Microscopic data, 6-24 agents This data category is similar to the previous, except that more pedestrians are present (6-24) and they are organized into circles, with their goal being to get to the antipodal positions. Here are two examples of comparisons:

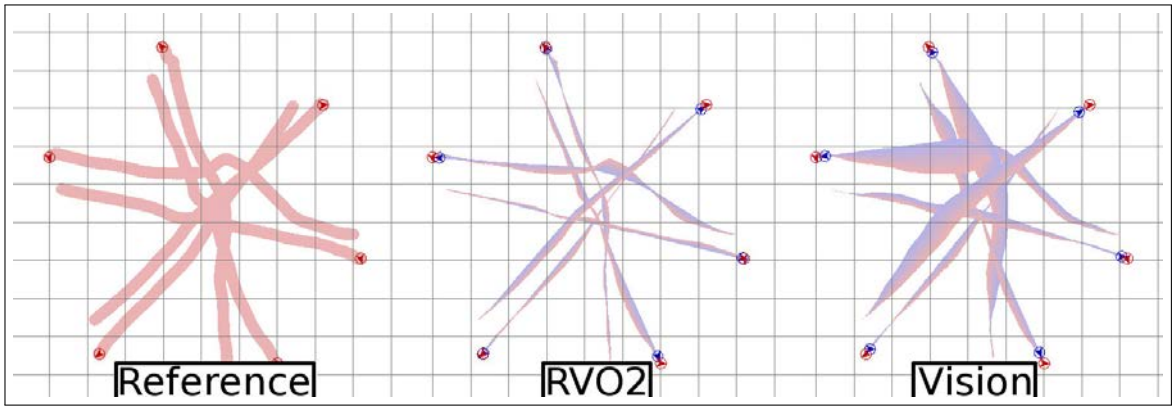
- six pedestrians case: before calibration all trajectories are far from the real ones; after calibration, RVO2 trajectories are near-identical to real-world data (Figure 3.5(c)).
- twenty-four pedestrians case: after calibration, Social-force agents agglomerate in the center without anticipation, while RVO2 agents anticipate future collisions and are spread in a pattern more similar to that of the real pedestrians (Figure 3.6).



(a) Pair-wise crossing.



(b) Four-agent crossing.



(c) Six agents in a circle going for antipodal positions.

Figure 3.5 – Examples of calibration results of algorithms with the Difference metric in different scenarios, which enables their fair comparison. Colored area represents error between real (red) and simulated (blue) trajectories identifiable through the color gradient. Left: reference data trajectories (ending positions in bright). **(a)** Middle: RVO2, incorrect trajectories. Right: Vision-based model, correct trajectories. **(b)** Middle: the RVO2 model's agents are ill-synchronized with real pedestrians. Right: Tangent correctly synchronizes agents with pedestrians. **(c)** Middle: RVO2; copes well with these more complex interactions. Right: Vision-based model; incorrect paths.

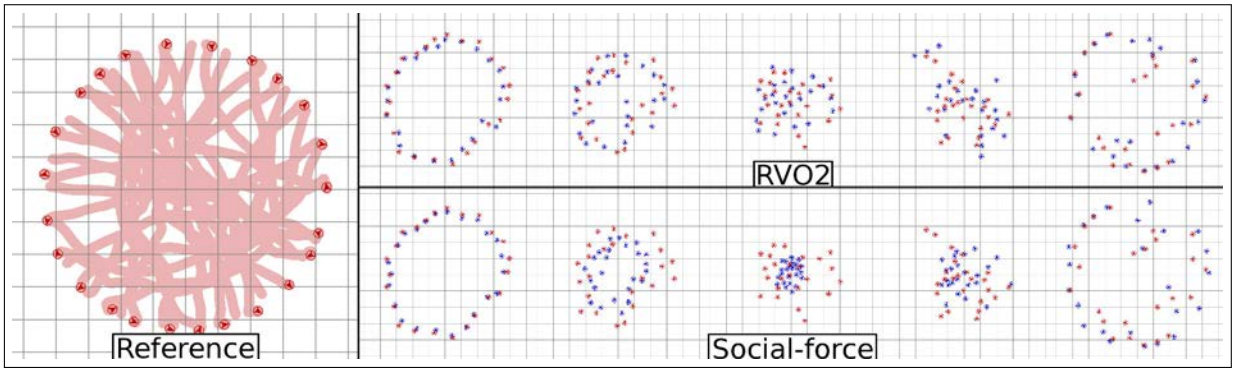


Figure 3.6 – Example of a result of the calibration of two models with the Difference metric in the case of 24 pedestrians in a circle with the goal to cross said circle. Left: reference data trajectories (ending positions in bright). Top row: five consecutive positions of real (red) and simulated (blue) agents in the case of the RVO2 model. Button row: same for the Social-force model. With the lack of anticipation, the Social-force model's agents tend to agglomerate in the center before solving their interactions. On the contrary, the RVO2 model's agents anticipate interactions and spread in a way more similar to the original data.

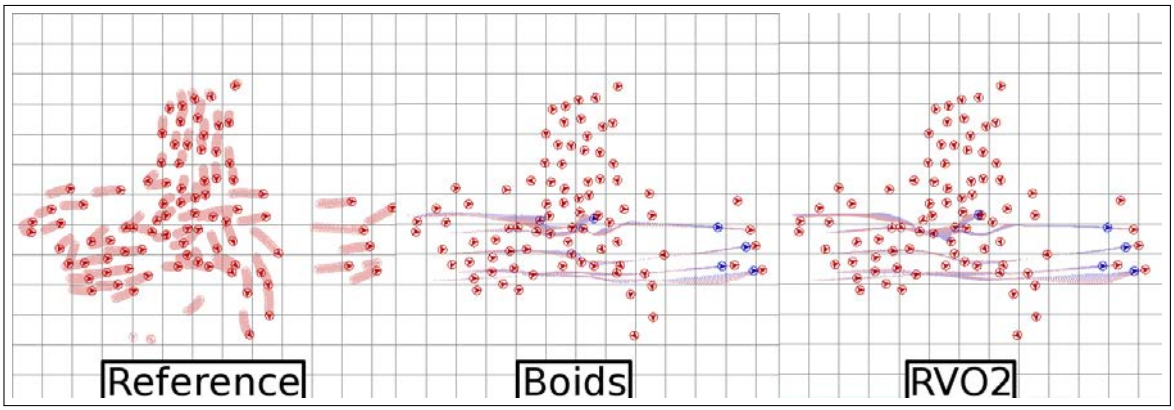
Microscopic data, ∼150 agents This data [PCBS11] contains ∼150 people in two groups crossing ways. Due to the nature of this data (pedestrians constantly appearing and disappearing from the cameras' field of view), a time-window is extracted and only a subset of all simulated pedestrians are evaluated (their positions and velocities are then known during the entirety of the time-window; the more numerous the controlled pedestrians are, the shorter the time-window is). Here are two examples:

- 111 agents total, 5 controlled, traveling from left to right: RVO2 leads to slightly higher errors than the Boids-like model (Figure 3.7(a)).
- 152 agents total, 5 controlled, traveling from left to right: RVO2 leads to lower errors than the Boids-like model (Figure 3.7(b)).

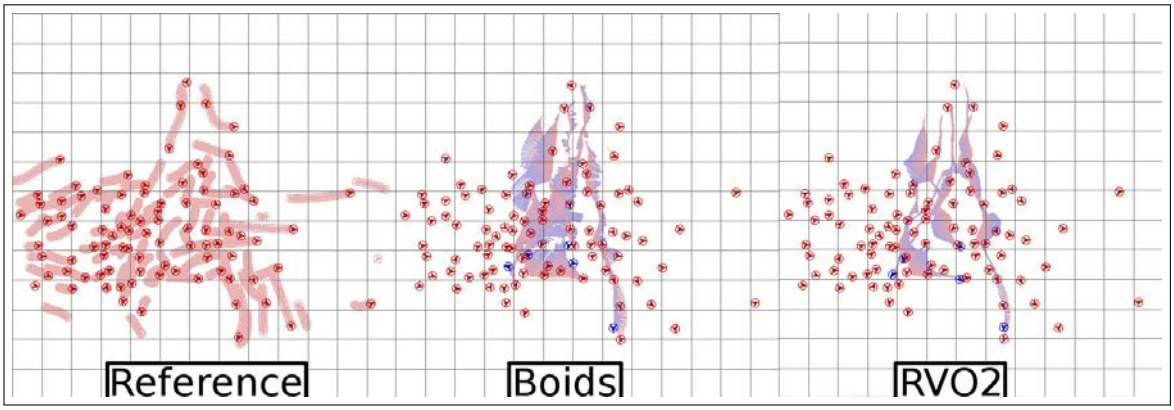
4.1.2 Macroscopic data

In many scenarios, the goal is not to match a specific motion, but to produce an overall characteristic flow. Our framework can be used in these scenarios. For example, there are well-documented cultural differences between the flow rates of Germans and Indians [CSC09]. These cultural flow differences can be described by two different fundamental diagrams.

We can match these fundamental diagrams in different situations with different simulation methods. To this end, we consider the fundamental diagrams of German and Indian people as described in [CSC09] and, in the case of 30-pedestrian crowds, constrain the Boids-like, Social-Force and RVO2 models to follow them. To constrain these models to the fundamental diagrams, we set them to 6 consecutive density-velocity points on each diagram (at 0.75, 1.0, 1.25, 1.50, 1.75 and 2.0 pedestrians per square meter).



(a) Five controlled agents (blue) amidst a flow of 111 agents.



(b) Five controlled agents (blue) amidst a flow of 152 agents.

Figure 3.7 – Example of a result of the calibration of two models with the Difference metric in the case of ∼150 agents; 5 of them are controlled by simulation algorithms. Colored area represents error between real (red) and simulated (blue) trajectories identifiable through the color gradient. Left: reference data; only recent trajectories are shown for clarity (ending positions in bright). Middle: Boids-like. Right: RVO2. **(a)** RVO2 leads to a slightly increased error compared to the Boids-like model. **(b)** RVO2 leads to a decreased error compared to Boids-like.

The diagrams obtained after merging the data for these 6 points are represented on Figure 3.8, along with the original data found in [CSC09].

These diagrams are very useful in setting up evacuation scenarios and adapting them to different cultures. Additionally, the framework can also help decide which model is best suited to a task. For example, here the Boids-like algorithm gets easily stuck in the corridor, and the resulting diagrams are far from the original data. However, the Social-force and RVO2 algorithms fit the data well; RVO2 ultimately matches the fundamental diagrams better at higher densities. This is largely due to the Social-force agents displaying instabilities near walls at high densities (as seen in Figure 3.9).

4.1.3 Sketch-like data

Our framework also has a broader application as a metric-driven animation tool for artists animating crowd scenes. If an animator provides a rough idea of a motion (in the form of a metric), our framework can be used to indicate which is the best algorithm to generate the animation, and can also provide the best parameters for the task. This spares the animator the tedious task of setting each agent's trajectory individually or building new mechanics into the model he is using. Additonally, this process is independent of the simulation algorithm. We provide three examples below. The first example is a group of pedestrians that are made to walk close to each other, then separate, and finally regroup. In order to simulate such behavior, we define three zones based on waypoints, where a distance metric is applied to determine pedestrian distance from one another. This metric is then used to maintain a low inter-agent distance in the first zone, a higher distance in the second zone, and a lower distance in the last zone. Figure 3.11 shows the resulting animations at various stages for the RVO2 algorithm.

The second example involves the vorticity metric used to create vortex-like patterns. Figure 3.10 shows results obtained with Boids and RVO2. Here, the Boids-like model lacks anticipation and fails to completely recreate the wanted behavior; RVO2 is more successful, thanks to anticipation.

Figure 3.12 shows another example with three corridors. Two groups enter through two corridors and exit as one group through a third corridor. They are made to be sparse when entering and dense when exiting.

4.2 Benchmarks

Figure 3.13 summarizes scores obtained by the Boids-like, Social-force and RVO2 algorithms for the various data categories in a benchmark fashion. These three algorithms were chosen for this benchmarking analysis as they are broadly applicable, make few assumptions about the scenarios being used in, and are representative of the types of simulation strategies commonly found in games and VR.

Examining the results, some trends can be seen to emerge across various dataset and metrics. For example, across many datasets and metrics the RVO2 algorithm tends to lead to better scores than our simple Boids-style simulation and Social-force models. This is likely due to RVO2 being the only one of the three methods to incorporate predictive collision avoidance. In fact, as the number of agents (and complexity of the scenario) increased, the advantage of RVO2 decreased significantly. This trend can be seen most

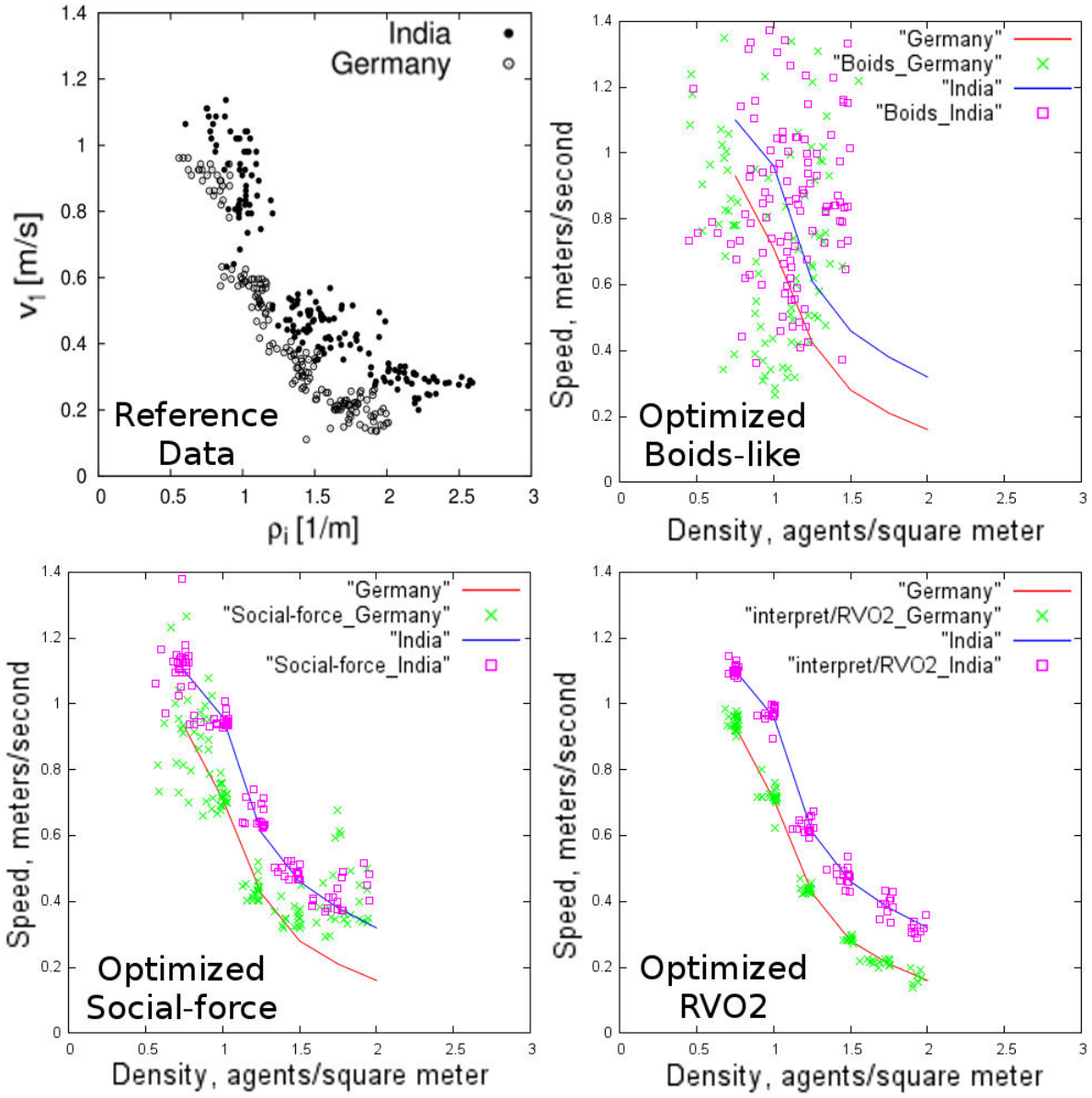


Figure 3.8 – Cultural variation in fundamental diagrams [Chattaraj et al. 2009]. Top left: original data; top right: Boids-like model fit to the fundamental diagrams; bottom left: Social-force; bottom right: RVO2. Boids poorly fits the data while Social-force and RVO2 models offer better matches. Social-force agents travel too fast at high densities compared to RVO2.

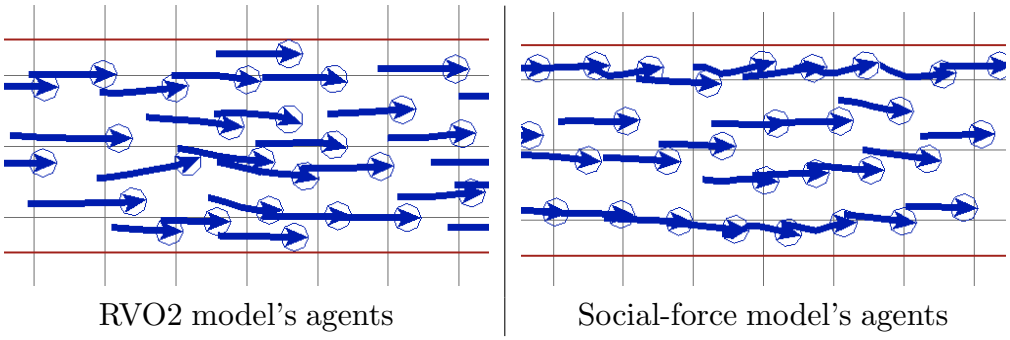


Figure 3.9 – High density (2 agents per square meter) illustration ("tails" indicating recent movement) after calibrating to a fundamental diagram. RVO2 agents on the left go straight. Social-force agents on right become unstable and bounce off of walls causing them to accelerate (explains the excess speed in the fundamental diagram in Figure 3.8).

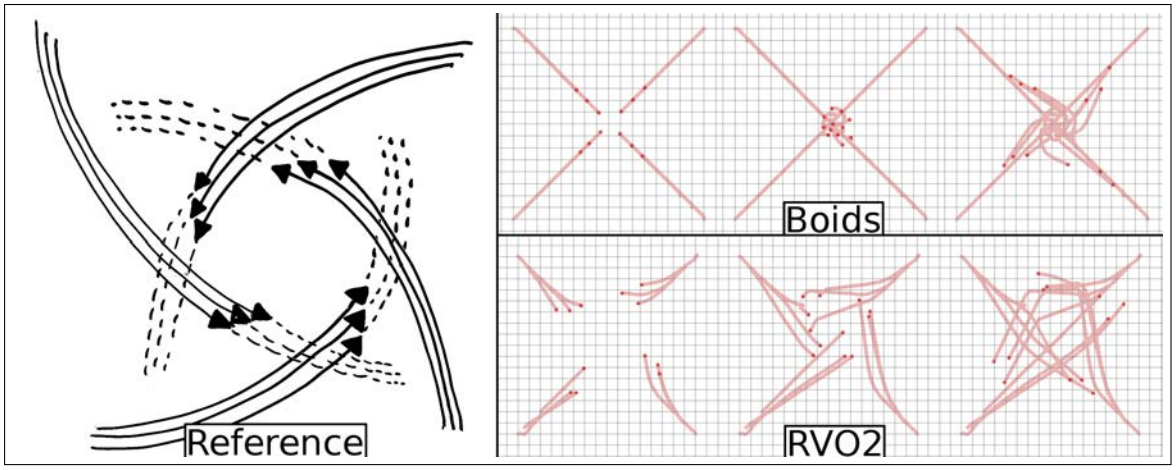


Figure 3.10 – Sketch-based simulation. Left: desired trajectories, four groups forming a vortex-like pattern. Top row: optimized solution for the Boids-like model. Bottom row: same for RVO2. Boids only offers radius and speed parameters, and doesn't include anticipation, it is unable to maintain significant distances between groups of agents without breaking them apart. RVO2 anticipates over a parameterized time horizon, allowing it to keep groups distant while maintaining members of a same group close.

Figure 3.11 – Sketch-based simulation. RVO2 agents are made to travel grouped, then disperse and then regroup again. Four consecutive states are shown.

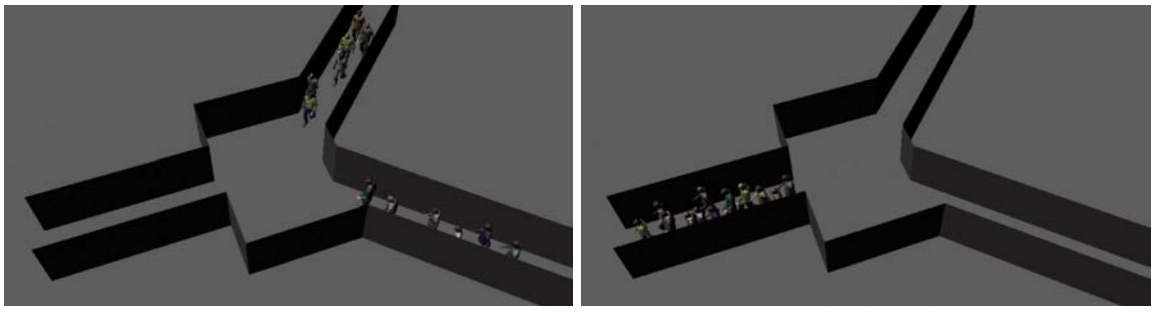


Figure 3.12 – Sketch-based simulation where two sparse groups enter through two corridors and merge into a dense group when exiting through a third corridor. Two consecutive states are shown.

clearly in the difference metric. The convergence of the performance is expected, in part, because there is little room to anticipate trajectories in dense scenarios with 100s of individuals in close quarters. RVO2 and the Social-force model score similarly in the Fundamental diagram metric for similar reasons.

Applying our framework to sketch-like data can also reveal interesting aspects of the simulation techniques. For example, the smooth group behaviors produced by the Boids-like simulation work well in capturing the desired behavior in separate & regroup benchmark. In contrast, the vortex scenario shows how RVO2's anticipation can be used to create novel behaviors difficult for the other methods such as keeping high distances between groups of agents while keeping agents belonging to a same group close to each other.

5 Analysis and Conclusions

In this work, we have addressed the problem of comparing various crowd simulation algorithms. We have formulated the estimation problem in a generalized way as an optimization problem. We have optimized the set of simulation parameters to provide the closest match between the simulation results and target reference data provided by users. We have implemented several comparison metrics to evaluate various aspects of simulation results. The resulting framework can be widely applied and supports different optimization metrics to match a user's target application.

Our approach has several important properties. First, our technique is very generally applicable. We have shown our ability to use the framework with a variety of simulation models (force-based, rule-based, velocity-based, etc.); a variety of types of metrics (microscopic, macroscopic); a variety of reference data (real data, example trajectories, macroscopic measures, sketch-based, etc.). Second, we have explored the problem in its full dimensionality; each agent given individualized parameter values while maintaining reasonable computing requirements. Finally, we have demonstrated the applications of our framework in various contexts.

Our results demonstrate the importance of parameter estimation: the same model can show very different behaviors depending on the parameters of the simulation. It is therefore crucial that researchers account for the effects of parameters when evaluating and comparing simulation models. Our framework has sought to address this question in a generalized way, and we hope this contribution opens various perspectives for future work. We would first like to add more models and more metrics to the framework (and try combining our approach with [SKFR09, KWS+11]). This would facilitate the use of the framework by the research community at large and facilitate efforts for more exhaustive comparisons for different simulation methods. We have also demonstrated the applicability of this framework to the creation of fully animated simulations from high-level specifications of desired behavior. Such cases were previously handled by means of scripts or tedious waypoint sequences that had to be defined by animators. We have shown that our parameter-estimation approach can alleviate some of this tedious work to assist in crowd animation. Further exploration of this area, perhaps with human subject studies, is a very promising research direction.

Limitations Along with the above contributions, our method has some limitations still

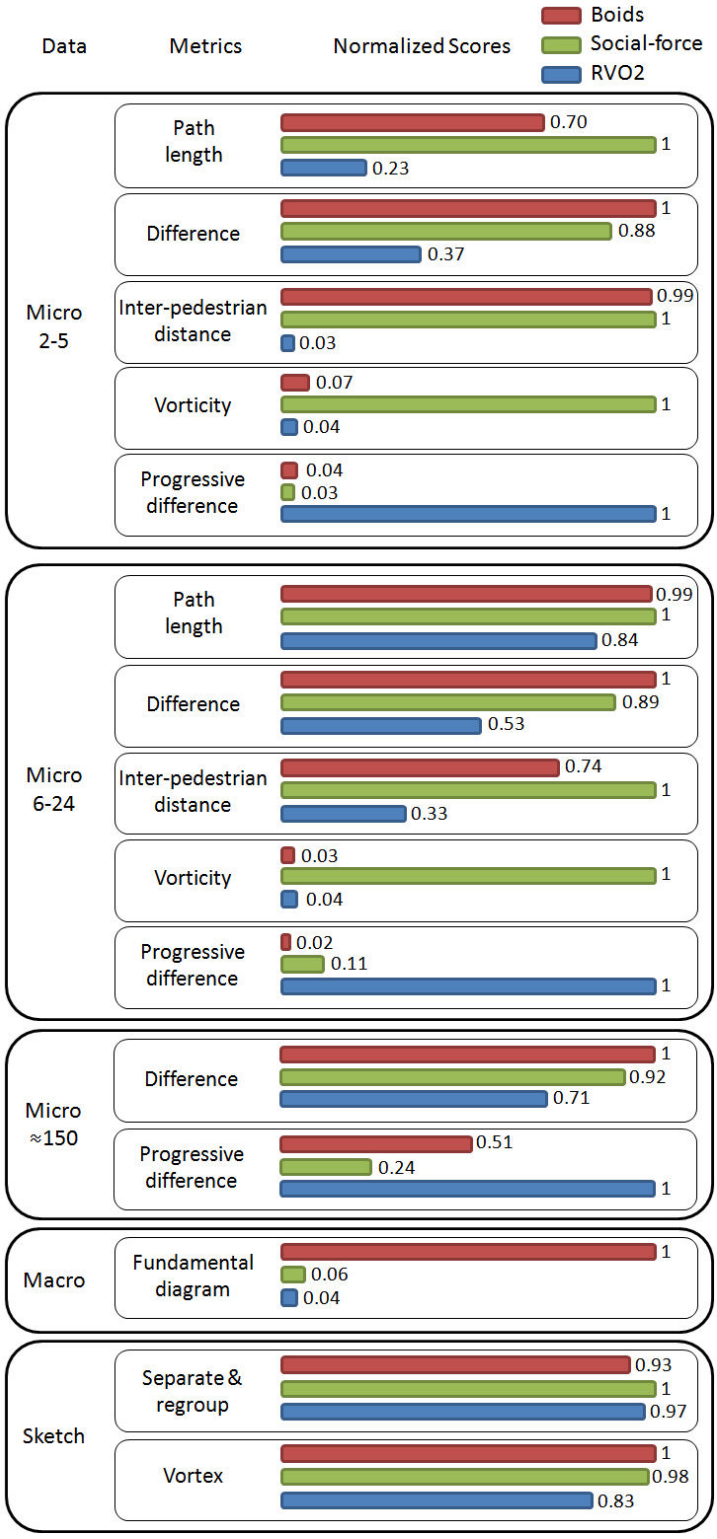


Figure 3.13 – Benchmarks. Comparative scores (lower is better) of the Boids-like (red), Social-force (green) and RVO2 (blue) models in five data categories: microscopic data with 2-5 agents, microscopic data with 6-24 agents, microscopic data with ∼150 agents, macroscopic, and sketch-based.

to be addressed. Most importantly, because a simulation's parameter-space is so high-dimensional, large scenarios with hundreds of agents are still time-consuming. It is not easy to estimate the complexity of our framework because of all the components involved. One metric evaluation step is usually equivalent to running a whole simulation based on the reference data and then comparing the results with the data. Our framework is thus very dependent on the complexities of the metrics and simulation algorithms as well as reference data. For instance, a longer time-window in the reference data or the lack of an accelerating structure (e.g. kd-tree) in the simulation algorithm's implementation can greatly impact the time an evaluation call takes. As for optimization algorithms, only the complexity (in terms of evaluation calls) of an iteration can be theoretically estimated: $O(nm)$ for the greedy algorithm and simulated annealing (n number of agents, m number of parameters per agent); constant (solution pool size) for the genetic algorithm. Some indicative times can be found in Figure A.3 from Appendix A. It is possible to address the question of performance and scalability by using parallel versions of optimization algorithms and simulators (this is less of a problem when evaluating multiple simulation algorithms on multiple scenarios which is easily parallelizable - e.g. one thread per scenario).

From here, in terms of future work, it could be interesting to study how parameters can be generalized beyond a given scenario or metric: can a certain set of parameters work well across various scenarios? Can a "style" be exported and kept for much larger crowds? In situations where there is not a single metric or a single piece of reference data to fit, it would be desirable to find parameter sets that are "good enough" in some high-level sense, or a set of scenarios that are semantically related and provide a good coverage of the problem domain. Such a goal will likely involve user studies and be greatly affected by perceptual factors. Finally, a key point in future studies will have to focus on the metrics. This would include determining which metrics (or combinations of metrics) are best adapted to various scenarios. But also defining metrics that are immune to the variability of agent behavior to similar conditions and that are able to capture high-level aspects that remain consistent across different data.

The first aspect (essentially learning parameters) is explored in Chapter 1, where a large quantity of insect data is used to train simulation algorithms. In that chapter, we also use the current work as a selector, in order to find which collision-avoidance algorithm is most suitable to reproduce observed behaviors (data). Additionally, an application to this work which had not been envisioned at first is explored in Chapter 2,. There, the process of evaluation and parameter estimation is used in an online tracking context, selecting the most appropriate simulator at every moment of the tracking in order to best predict the next state of the crowd, thus improving the real-time tracking of pedestrians. In these two applications, our framework is used most notably as a selector, allowing to use the correct simulator for a given behavior (insect swarming in the first application) and for a given moment (current/instantaneous pedestrian movement in the second application).

Finally, during this work, while using the developed framework, it became clear that despite recent progress on crowd modeling, more work needs to be done in order to further improve simulation results, thus motivating the development of a new simulation algorithm. Further, assuming such an algorithm is developed, we now have a tool that we can use to evaluate it. This is the other main project presented in this document,

which can be found in the next chapter (Chapter 4).

WarpDriver: Context-Aware Probabilistic Motion Prediction for Crowd Simulation

4

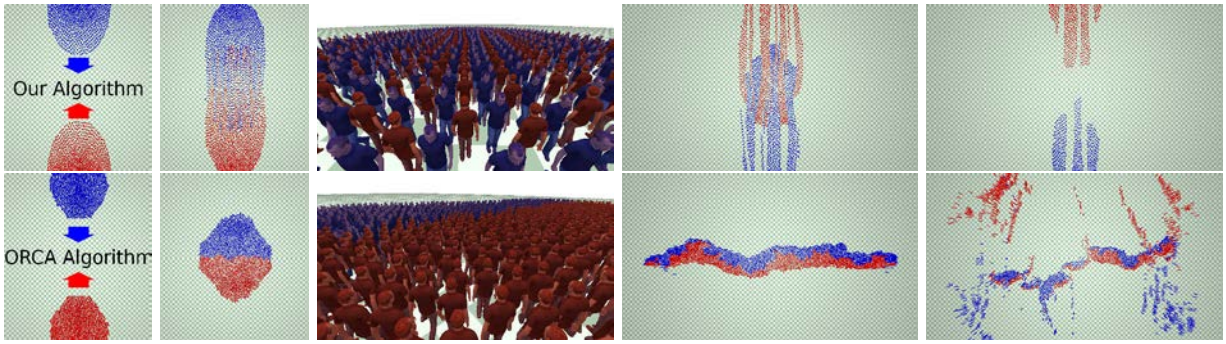


Figure 4.1 – Two 1027-agent groups exchange positions. Each row is a sequence of temporally consecutive stills of the simulations, with stills directly below/above each other at the same timesteps. The third image of each row is a third-person view from a red agent's perspective. Top: our algorithm, groups merge with the formation of lanes. Bottom: ORCA algorithm, groups collide and spread before agents are able to resolve the situation. Overall, our algorithm resolves the situation more quickly and fluidly than ORCA.

Contents

1 Introduction

Much attention has recently been devoted to crowd simulation due to its applications in pedestrian dynamics, virtual reality and digital entertainment. As noted in the *Background* chapter (Chapter 2) and confirmed by the evaluation framework we presented in the previous chapter (Chapter 3), the most recent, velocity-based, microscopic, crowd simulation algorithms [PPD07, vdBLM08] introduced significant progress both in terms of realism and quality. These algorithms and their follow-on works are all based on the same principle: a linear trajectory prediction for future motion using each agent's position and assumed constant velocity. Motion prediction, which humans perform for anticipated reactions [OMCP12], results in smoother trajectories and improves the visual quality of the velocity-based simulations in many ways compared to first-order position-based techniques [HM95].

Nevertheless, even when using state-of-the-art simulation algorithms, visual artifacts or other non-satisfactory results can still often be observed: (1) aggregations of agents and exaggerated avoidance behaviors when dense crowds are involved, (2) last-minute collision-avoidance manoeuvres in seemingly simple/moderately dense situations or (3) agents reacting for no obvious reason to other agents, again even in seemingly simple situations. The causes for these artifacts lie in the basic assumptions of available crowd-simulation methods, including both first-order and velocity-based algorithms. Firstly, first-order motion prediction algorithms cannot anticipate future dense-spots, while velocity-based algorithms look for collision-free velocities in velocity space; the velocity space can quickly get saturated when crowd density is high. In addition, further-in-time interactions are treated in the same way as imminent interactions. Secondly, velocity-based motion prediction techniques linearly extrapolate agents' future motions, so agents moving along non-linear trajectories cannot be detected by other agents except at the last moment when a collision is about to happen (note that agents can interact only at the last moment in first-order algorithms). Finally, agents' instantaneous velocities are used when extrapolating their future motions: agents exhibiting jerky motions for instance, create "false" constraints over other agents (they also unnecessarily affect other agents when passing too closely to each other in first-order algorithms).

In order to address these simulation artifacts, we introduce a new algorithm that has the following characteristics: First, agent interactions are modeled as space-time collision probabilities, that agents steer away from: agents can then fluidly navigate along the minima of the collision probability function even in highly-constrained situations. Second, to construct these collision probabilities, we propose an efficient, easily extensible collision probability sampling algorithm. This algorithm starts with a collision probability field that is intrinsic to all agents, and warps this *Intrinsic Field* using *Warp Operators* for each information source to be considered. Third, we propose a variety of *Warp Operators*, allowing us to anticipate agents' future motions in a non-linear fashion, by taking into account: agent/obstacle positions and velocities, sensing uncertainties, environment layouts, interaction outcomes with obstacles, and past trajectories among other cues.

In summary, in this chapter we propose a novel, easily extensible collision-avoidance

algorithm for microscopic crowd simulation. Our algorithm addresses the visually noticeable artifacts as mentioned above.

- Its probabilistic nature minimizes aggregations of agents and better supports avoidance maneuvers in dense crowds (first artifact).
- By using appropriate *Warp Operators*, agents extrapolate each-others' near-term trajectories adaptively in a non-linear manner, avoiding last-minute collision-avoidance maneuvering (second artifact).
- By using appropriate *Warp Operators*, this method can use agents' motion history to correctly anticipate future events, thereby avoiding agents reacting to each other with no apparent reason (third artifact).

Due to its heavy reliance on *Warp Operators*, we name this algorithm "WarpDriver".

The chapter is organized as follows. In Section 2, we give a summary of related work already given in Chapter 2, describing how the evolution of agent-based algorithms leads to WarpDriver. In Section 3, we give an overview of our algorithm. We present the notations and problem setting in Section 4, while Section 5 describes our sampling method (*Intrinsic Field* and *Warp Operators*), and Section 6 details how steering decisions are taken. In Section 7, we apply our crowd-simulation algorithm to a wide range of challenging scenarios, testing how it solves the aforementioned artifacts. We demonstrate the benefits of our method in highly constrained environments and situations where other algorithms need per-case handling and scripting. Finally, we discuss the limitations of our approach in Section 8, and conclude in Section 9.

2 Summary of Related Work

As related work on the topic of crowd simulation algorithms has already been presented in Chapter 2, this section will simply serve to summarize it.

Following the many applications of crowd simulation in pedestrian dynamics, virtual reality and cinematic entertainment, many algorithms spanning several categories and each with its own characteristics have been devised. Macroscopic crowd simulation algorithms [NGCL09, TCP06] animate crowds at the global level, aiming to capture statistical quantities such as flows or densities, while tile- and patch-based algorithms facilitate the control of these global features. In contrast, microscopic algorithms model interactions between individual pedestrians, with the emergence of movement patterns at the crowd level. For instance, algorithms based on cellular automata discretize space into grids where pedestrians are moved based on transition probabilities [KS08, Sch01]. Other, agent-based algorithms model pedestrians as agents, with various levels of complexity. Finally, example-based algorithms maintain databases of crowd motions which can be reused depending on the context [LCL07, JCP+10].

Among these works, agent-based algorithms remain very popular, due to their ease of implementation and their flexibility through various extensions and scripting. To reproduce local interactions between people, these algorithms have always focused on the most readily availabe and easily useable information: agents' positions (first-order algorithms). This has been the case starting with Reynolds' seminal work with the Boids

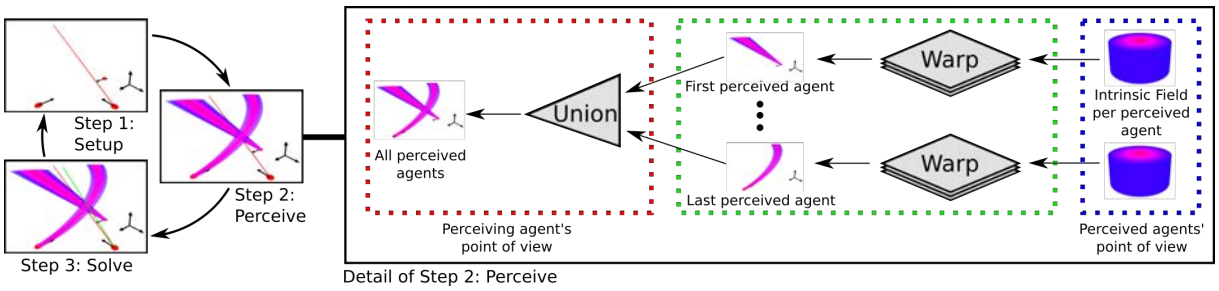


Figure 4.2 – Overview of the Algorithmic Framework of WarpDriver.

algorithm [Rey87], later the Social-Forces algorithm by [HM95, HFV00] and many of their derivatives ever since.

However, anticipation of each other's trajectories is key to people's interactions, and efficient, collision-free navigation [OMCP12, KSG14]. In light of this observation, major advances recently came from second-order, velocity-based algorithms. [Rey99] introduced the point of closest approach between agents, where, if the distance between the concerned agents was to be low enough at this point (reflecting a collision), they would steer away from it. Later, in an algorithm derived from Social-Forces, [KHBO09] used this point of closest approach as a source of repulsive forces; and [PESVG09] used the distance of the closest approach to refrain from choosing velocities which might lead to collisions. In parallel, other algorithms [Feu00, PPD07] work in space-time (2-dimensional space plus one more dimension of time) to, again, select permitted, collision-free velocities. This method of choosing permissible velocities was further accelerated by algorithms that reasoned in 2-dimensional velocity-space such as [vdBLM08, GCK+09, GCLM12, POO+09]. Finally, other algorithms used instantaneous velocities in other ways, such as affordance fields [KSHF09] and velocity-derived values processed from the synthetic visual flows of agents [OPOD10].

As a common assumption, these algorithms all linearly extrapolate agents' future motions from their positions and velocities, making it possible to anticipate collisions up to a certain time horizon and improve simulation results [OMCP12, GvdBL+12, WGO+14]. However, this linear extrapolation remains simplistic, and in many more challenging situations, does not yield truly satisfactory results. Consequently, [KGL+14] introduced a probabilistic component to the algorithm presented by [vdBLM08], while [GNL13] added look-ahead to adaptively increase the time horizon in an efficient way for large groups. Finally, [vdBSGM11] incorporated agents' acceleration constraints into this same algorithm.

This underlying assumption of linear motion prediction, however, does not hold in many cases, and we suggest that constraining a crowd simulator to only information on positions and instantaneous velocities is often insufficient. By addressing these issues, we introduce an approach that enables agents to efficiently take into account arbitrary sources of information in a stochastic framework when anticipating each other's future motions.

3 Overview

Our algorithm builds on an agent-based modeling framework and the resulting simulation captures complex interactions among agents. In this section, we provide a high-level overview (Figure 4.2) of our approach, i.e. how we model interactions between agents and steer them.

Before describing "WarpDriver", we first need to define what we consider an agent in our formulation. An agent is any entity that the algorithm would steer or any other entity that could affect another agent's steering decisions. Agents can be, for instance, pedestrians, cars or walls, and they can further have various *properties*: size, shape, position, velocity, followed path, etc.

In our formulation, interactions between agents are resolved in space-time. To simulate these interactions, we identify the *perceiving* agent (the agent we are currently steering) and the *perceived* agents (the agents that are to be avoided). Interactions among agents are modeled in three main steps:

Step 1, Setup: The *perceiving* agent starts by defining its space-time *projected trajectory*: the trajectory it would follow if no collisions were to happen (*Step 1* on Figure 4.2 and red dotted line on Figure 4.3; detailed in Section 4).

Step 2, Perceive: This agent then constructs its perception of other *perceived* agents' future motions in the form of space-time collision probabilities (*Step 2* on Figure 4.2 and color gradient on Figure 4.3; detailed in Section 5).

Step 3, Solve: Finally, the agent intersects its *projected trajectory* with these collision probabilities (thus evaluating the chances of collision along the *projected trajectory*) and modifies its *projected trajectory* by performing one step of gradient descent to lower its collision probabilities along this trajectory (*Step 3* on Figure 4.2 and green dotted line on Figure 4.3; detailed in Section 6).

The most important aspect of our approach is then how the *perceiving* agent derives collision probabilities from the *perceived* agents. It is through this process that we can model any non-linear behavior of both *perceiving* and *perceived* agents.

Our goal for the collision probability formulation process (Step 2) is to be able to handle each *property* separately. Thus, we define the *Intrinsic Field* as the lowest common denominator among all agents: the fact that they occupy a volume in space-time; this is a collision probability field. We then model any additional *property* as a *Warp Operator* which further warps the *Intrinsic Field*.

In order to define a clean system pipeline for implementation, we further associate every agent with its own *agent-centric* space-time. Step 2 is then described by the following three sub-steps:

- Every *perceived* agent is modeled as an *Intrinsic Field* in its *agent-centric* space-time (blue rectangle in Figure 4.2).
- *Warp Operators* progressively warp every *perceived* agent's *Intrinsic Field* from its *agent-centric* space-time into the *perceiving* agent's *agent-centric* space-time (green rectangle in Figure 4.2).

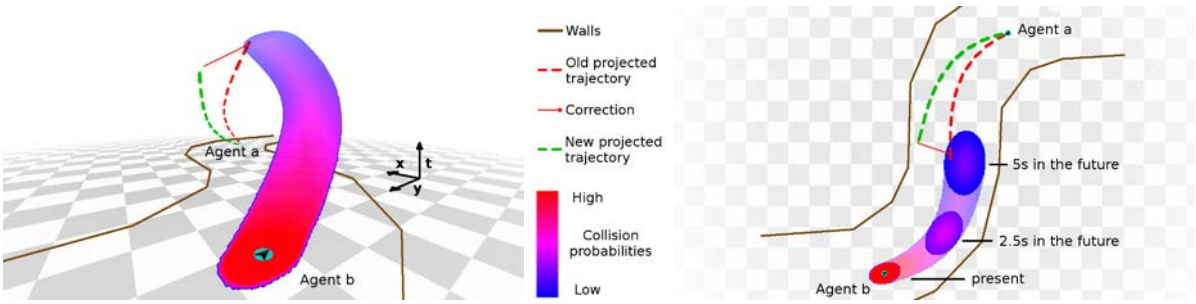


Figure 4.3 – Illustration of collision avoidance between two agents a and b on a curved path. On the left is the full 3D representation (space and time) of collision probabilities; on the right is a simplified 2D view. The color gradient represents a's probability of collision with b, as perceived by a. The red dotted line represents a's initial *projected trajectory*. The green dotted line represents a's final, corrected, *projected trajectory* (exaggerated). The red line in between both dotted ones represents the correction agent a will perform (exaggerated for illustration). Note that thanks to *Warp Operators*, the *projected trajectory* is a curve instead of a straight line.

- These warped collision probability fields (in the *perceiving* agent's *agent-centric* space-time) are then combined into a single collision probability field (red rectangle in Figure 4.2).

Note that by confining agents' *properties* to Step 2, the *perceiving* agent's *projected trajectory* can be simply defined as a line in its *agent-centric* space-time, which simplifies further computations (more detail in Sections 4, 6).

4 Notations and Setup

In this section, we describe the notations used throughout the chapter and detail how a *perceiving* agent constructs its *projected trajectory*, i.e. its current trajectory in space-time assuming no collisions take place (Step 1 of our approach, see Figure 4.2):

- $\cdot, \times, \circ, \star$ and $*$ respectively denote the dot product, cross product, function composition, component-wise multiplication and convolution.
- $\overrightarrow{\nabla}$ is the nabla operator. For a continuous field f, $\overrightarrow{\nabla} \cdot f$ is the gradient of f.
- $\cup$ is the union operator and $\bigcup$ is the union operator over a set.
- $\mathcal{A}$ is the set of all agents, $a, b \in \mathcal{A}$ are two such agents; note that a usually denotes the *perceiving* agent while b usually denotes the *perceived* agent.
- $\mathcal{S}$ is a 3D space-time with basis $\{\mathbf{x}, \mathbf{y}, \mathbf{t}\}$, where $\mathbf{x}$ and $\mathbf{y}$ form the space of 2D positions and $\mathbf{t}$ is the time. A point in such a space-time is noted $\mathbf{s} = (x, y, t) \in \mathcal{S}$. Note the difference between bold-face vectors (e.g. $\mathbf{x}$) and normal-font scalar quantities (e.g. x).

- $\mathcal{S}_{a,k}$ is the *agent-centric* space-time $\mathcal{S}$ centered on an agent a at timestep k such that, in this space-time, agent a is at position $\mathbf{o} = (0,\ 0,\ 0) \in \mathcal{S}_{a,k}$ and faces along the local $\mathbf{x}$ axis, positive values along the local $\mathbf{t}$ axis represent the future.

- $\mathbf{r}_{a,k}$ is agent a's *projected trajectory* in $\mathcal{S}_{a,k}$.

- $\forall \mathbf{s} \in \mathcal{S}_{a,k},\ p_{a \to b,k}(\mathbf{s})$ is what agent a perceives to be its collision probability with agent b.

- I, the *Intrinsic Field*, gives the probability of colliding with any agent b in space-time $\mathcal{S}_{b,k}$. $\overrightarrow{\nabla} \cdot I$ is the gradient of I.

- W denotes a *Warp Operator* that warps I for every *property* of an agent. $\mathbf{W} = W_n \circ ... \circ W_1$ further denotes the composition of operators $\{W_1, ..., W_n\}$.

- W^{-1} is used to apply the inverse of a *Warp Operator* W to probabilities and probability gradients. Assuming a collection of operators $\{W_1, ..., W_n\}$ where $\mathbf{W}(\mathcal{S}_{a,k}) = \mathcal{S}_{b,k}$, then $\mathbf{W}^{-1} = W_1^{-1} \circ ... \circ W_n^{-1}$ and $\forall \mathbf{s} \in \mathcal{S}_{a,k}$:

$$(\mathbf{W}^{-1} \circ I \circ \mathbf{W})(\mathbf{s}) = p_{a \to b,k}(\mathbf{s}), \tag{4.1}$$

$$(\mathbf{W}^{-1} \circ (\overrightarrow{\nabla} \cdot I) \circ \mathbf{W})(\mathbf{s}) = \overrightarrow{\nabla} \cdot p_{a \to b,k}(\mathbf{s}). \tag{4.2}$$

With these notations, in Step 1 of our approach, the *perceiving* agent a constructs its *projected trajectory* $\mathbf{r}_{a,k}$ in its *agent-centric* space-time $\mathcal{S}_{a,k}$. We further assume that the *perceiving* agent a is a point in its *agent-centric* space-time, $\mathcal{S}_{a,k}$ is then its configuration-space. As mentioned in Section 3, since the processing of agents' *properties* is confined to Step 2, the *perceiving* agent's *projected trajectory* can be defined as a line.

Specifically, assuming agent a has an instantaneous speed $v_{a,k}$ at timestep k, its *projected trajectory* is expressed as $\mathbf{r}_{a,k} = line(\mathbf{o},\ v_{a,k}\mathbf{x} + \mathbf{t})$. Further, at any time $t \in \mathbb{R}$ in the future, the *perceiving* agent a projects to be at point $\mathbf{r}_{a,k}(t) = \mathbf{o} + t(v_{a,k}\mathbf{x} + \mathbf{t})$ in space-time $\mathcal{S}_{a,k}$.

5 Perception: collision probability Fields

We here describe how the *perceiving* agent constructs collision probabilities from the *perceived* agents. As mentioned in Section 3, this is a three-step process where:

- the *Intrinsic Field* I is defined for each *perceived* agent b in its *agent-centric* space-time $\mathcal{S}_{b,k}$,

- *Warp Operators* warp I from each $\mathcal{S}_{b,k}$ into the *perceiving* agent a's *agent-centric* space-time $\mathcal{S}_{a,k}$, thus modeling agents' *properties*,

- the resulting collision probability fields are combined.

We detail each of these three steps in the following sub-sections.

5.1 The Intrinsic Field

As defined in Section 3, the *Intrinsic Field* is the lowest common denominator between agents, independently of their *properties*. It is also a continuous collision probability field: for each point $\mathbf{s}$ in a *perceived* agent b's *agent-centric* space-time $\mathcal{S}_{b,k}$, it gives the probability of colliding with b at that point $I(\mathbf{s}) \in [0,\ 1]$.

Since the *perceiving* agent a is a point in its *agent-centric* configuration-space $\mathcal{S}_{a,k}$, any *perceived* agent b should therefore be perceived as a configuration-space obstacle (the Minkowski sum of agents a and b). As we want the *Intrinsic Field* to be independent of agents' *properties* (including size and shape) we define the Minkowski sum of agents a and b as a disk with a normalized radius of 1, this is the step function g:

$$\forall \mathbf{s} = (x, y, t) \in \mathcal{S}_{b,k},\ g(\mathbf{s}) = \begin{cases} 1, & \text{if } \sqrt{x^2 + y^2} \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

We further model the perception error in the form of a Gaussian function: $\forall \mathbf{s} = (x, y, t) \in \mathcal{S}_{b,k},\ f(\mathbf{s}) = exp(-(\frac{x^2+y^2}{2\sigma^2}))$.

Consequently, we define the *Intrinsic Field* as the convolution of functions f and g:

$$\forall \mathbf{s} \in \mathcal{S}_{b,k},\ I(\mathbf{s}) = (f * g)(\mathbf{s}). \tag{4.3}$$

It is computed up to a normalized time of 1 second in the future. An illustration of the *Intrinsic Field* can be found on Figure 4.2 (cylinder on the right side of the figure).

5.2 Warp Operators

Warp Operators model each agent *property* that we want to include in the algorithm. As mentioned in Section 3, these could be: shape, size, position, velocity, followed path, etc. Mechanically, *Warp Operators* warp the *Intrinsic Field* defined for each *perceived* agent b in its *agent-centric* space-time $\mathcal{S}_{b,k}$ into the *perceiving* agent a's *agent-centric* space-time $\mathcal{S}_{a,k}$.

In this sub-section, we describe *Warp Operators* modeling agent-related and context-related *properties*. Note that their formal expressions are given in Appendix B.

5.2.1 Agent-Related Operators

The following *Warp Operators* model *properties* which only depend on agents:

Position and Orientation The *Warp Operator* W_{local} models the agents' position and orientation *properties*. It is a simple change of referential between $\mathcal{S}_{a,k}$ and $\mathcal{S}_{b,k}$.

Time Horizon To avoid collisions in a time horizon $\mathcal{T}$ (beyond the normalized 1 second in the *Intrinsic Field*), we define a time horizon operator W_{th}.

Time Uncertainty The W_{tu} operators models the increased uncertainty on the states of other agents the further we look in time.

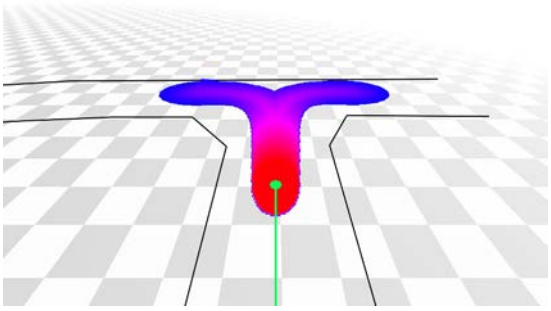
Radius The W_r operator changes the radius of the agents by dilating space along the **x** and **y** axes.

Velocity The W_v operator models the agent's instantaneous velocity as a displacement along the **x** axis.

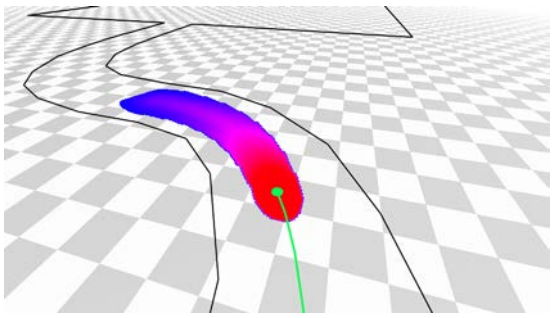
Velocity Uncertainty Depending on the speed of an agent, that agent could be more or less likely to make certain adaptations to its trajectory. For instance, the faster an agent travels, the more likely it is to accelerate/decelerate rather that turn. This is modeled by the W_{vu} operator.

5.2.2 Context-Related Operators

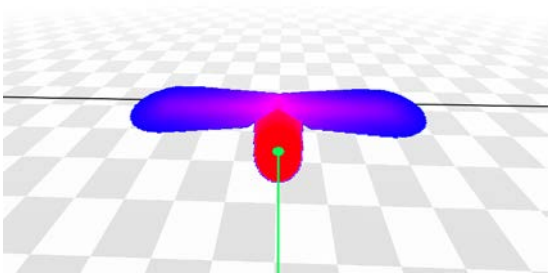
The following operators provide information based on the Environment Layout (operator W_{el}), Interactions with Obstacles (operator W_{io}) and Observed Behaviors of agents (operator W_{ob}). These operators, where applicable, **replace** the *Local Space operator* W_{local}. We call W_{ref} the resulting operator: $W_{ref} = \{W_{local}\ or\ W_{el}\ or\ W_{io}\ or\ W_{ob}\ \}$.



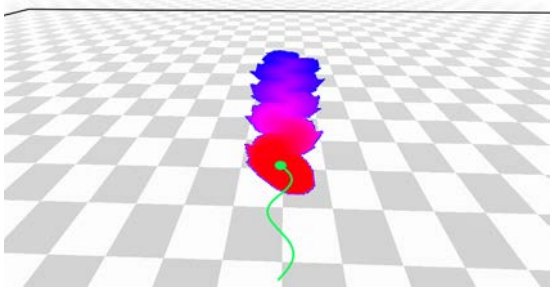
(a) W_{el}: T-junction, the agent could turn left or right.



(b) W_{el}: Predicted motion of an agent on a curved path.



(c) W_{io}: The agent can not go further than the wall, either go left or right.



(d) W_{ob}: Predicted motion based on observed past motion.

Figure 4.4 – Cases using context-related *Warp Operators* (Section 5.2.2). Each case represents one context-related *Warp Operator* combined with all agent-related ones (Section 5.2.1). Same simplified 2D representation as in Figure 4.3(right).

Environment Layout When navigating in an environment, based on its layout, we can predict what trajectories other pedestrians are likely to follow. In a series of hallways,

for instance, when not threatened by collisions with other pedestrians, one would stay roughly in the middle of the hallway and take smooth turning trajectories at intersections (an agent could turn either left or right in Figure 4.4(a)). When navigating on curved paths, one would, again, have a tendency to stay roughly in the middle, resulting in a curved trajectory (Figure 4.4(b)). The operator W_{el} models this knowledge by warping space to "align" it with these probable trajectories.

Interactions With Obstacles Where the environment layout operator focuses on other agents' probable trajectories assuming they will continue travelling, this operator W_{io} takes care of possible interactions between agents and obstacles. These interactions are essentially much more drastic changes to an agent's locomotion than paths, such as full stops. These can occur if, for instance, an agent comes up to a wall (Figure 4.4(c)) (to interact with an ATM, look out the window, check a map...). This can also happen with an agent coming into contact with a small/temporary/unexpected obstacle which could force it to stop and then "hug" the obstacle to get around it.

To achieve this, we construct a graph around each obstacle (an obstacle being modeled as a series of connected line segments). When an agent's *projected trajectory* intersects with an obstacle, we extend the graph to that agent and "align" space-time wih this graph.

Observed Behaviors With the last operator W_{ob}, we aim to improve the prediction of agents' future motions by looking at their past ones. In the worst case, we might not find any useful information, which won't impact the prediction. However, we might also find some behaviors similar to what the agent is currently doing (e.g. turning in a particular way) or, in the best case, we might find patterns (e.g. agents going in near-circles, zig-zags...) that we can extend to the currently-observed situation (Figure 4.4(d) shows anticipation on a zig-zagging agent).

In order to take this information into account for an agent a at timestp k, we keep a history of this agent's positions during h previous timesteps. These past positions form a graph which we repeat on the current position of the agent and then "align" space-time with it.

5.2.3 Composition of Warp Operators

As defined in Section 3, we can compose all these operators $\{W_{ref}, W_{th}, W_{tu}, W_r, W_v, W_{vu}\}$:

$$\mathbf{W} = W_{ref} \circ W_{th} \circ W_{tu} \circ W_r \circ W_v \circ W_{vu},$$
$$\mathbf{W}^{-1} = W_{vu}^{-1} \circ W_v^{-1} \circ W_r^{-1} \circ W_{tu}^{-1} \circ W_{th}^{-1} \circ W_{ref}^{-1}.$$

For any point $\mathbf{s}$ in *perceiving* agent a's *agent-centric* space-time $\mathcal{S}_{a,k}$:

$$p_{a\to b,k}(\mathbf{s}) = (\mathbf{W}^{-1} \circ I \circ \mathbf{W})(\mathbf{s}),$$
$$\overrightarrow{\nabla} \cdot p_{a\to b,k}(\mathbf{s}) = (\mathbf{W}^{-1} \circ (\overrightarrow{\nabla} \cdot I) \circ \mathbf{W})(\mathbf{s}).$$

5.3 Combining collision probability Fields

Before the collision avoidance problem can be solved, one last mechanic still needs to be defined which is how pair-wise interactions can be combined (Step 3 on Figure 4.2). Let a be the *perceiving* agent, and $b, c \in \mathcal{A}$, $b \neq a$, $c \neq a$ be a pair of *perceived* agents. At timestep k, we have access to the following collision probabilities: $p_{a\to b,k}$ and $p_{a\to c,k}$. We can then define the probability agent a has of colliding with either b or c:

$$p_{a\to\{b,c\},k} = p_{a\to b,k} + p_{a\to c,k} - p_{a\to b,k}p_{a\to c,k}.$$

And we can similarly define its gradient:

$$\begin{aligned}\overrightarrow{\nabla} \cdot p_{a\to\{b,c\},k} = \overrightarrow{\nabla} \cdot p_{a\to b,k} + \overrightarrow{\nabla} \cdot p_{a\to c,k} \\ - p_{a\to b,k}\overrightarrow{\nabla} \cdot p_{a\to c,k} \\ - p_{a\to c,k}\overrightarrow{\nabla} \cdot p_{a\to b,k}\end{aligned}$$

Finally, considering the whole set of agents $\mathcal{A}$, the probability agent a has of colliding with any other agent $b \in \mathcal{A} \setminus a$ is obtained in the same manner and noted $p_{a\to\mathcal{A}\setminus a,k}$, with the corresponding gradient $\overrightarrow{\nabla} \cdot p_{a\to\mathcal{A}\setminus a,k}$.

6 Solving the Collision-Avoidance Problem

This section details the third and final step in our approach: how the *perceiving* agent modifies its *projected trajectory* to reduce the collision probabilities along it.

To solve the collision-avoidance problem, the *perceiving* agent samples collision probabilities and their gradients $p_{a,k}$ along its *projected trajectory* $\mathbf{r}_{a,k}$ (this is the cost function and its gradient), and performs one step of gradient descent to modify its *projected trajectory*. First, we compute the overall probability an agent a has of colliding with other agents $p_{a,k}$, its gradient $\overrightarrow{\nabla} \cdot p_{a,k}$ and application point $\mathbf{s}_{a,k}$ (red continuous line in Figure 4.3), when traveling along its *projected trajectory* $\mathbf{r}_{a,k}$ (red dotted curve in Figure 4.3). We compute these quantities for a time horizon $\mathcal{T}^*$ until a collision with a wall is detected: $\mathcal{T}^* \leq \mathcal{T}$. With the following normalization factor $N_{a,k}$, and $t \in [0, \mathcal{T}^*]$:

$$N_{a,k} = \int_t p_{a\to\mathcal{A}\setminus a,k}(\mathbf{r}_{a,k}(t)), \tag{4.4}$$

We compute $p_{a,k}$, $\overrightarrow{\nabla} \cdot p_{a,k}$ and $\mathbf{s}_{a,k}$:

$$p_{a,k} = \frac{1}{N_{a,k}} \int_t p_{a\to\mathcal{A}\setminus a,k}(\mathbf{r}_{a,k}(t))^2, \tag{4.5}$$

$$\overrightarrow{\nabla} \cdot p_{a,k} = \frac{1}{N_{a,k}} \int_t p_{a\to\mathcal{A}\setminus a,k}(\mathbf{r}_{a,k}(t)) \, (\overrightarrow{\nabla} \cdot p_{a\to\mathcal{A}\setminus a,k})(\mathbf{r}_{a,k}(t)), \tag{4.6}$$

$$\mathbf{s}_{a,k} = \frac{1}{N_{a,k}} \int_t p_{a\to\mathcal{A}\setminus a,k}(\mathbf{r}_{a,k}(t)) \, \mathbf{r}_{a,k}(t). \tag{4.7}$$

From these quantities, given a user-set parameter α, we compute the new projected trajectory that agent a should follow in $\mathcal{S}_{a,k}$ to lower its collision probability (green dotted curve in Figure 4.3) :

$$\mathbf{r}^*_{a,k} = line(\mathbf{o},\ \mathbf{s}_{a,k} - \alpha p_{a,k} \overrightarrow{\nabla} \cdot p_{a,k}). \tag{4.8}$$

7 Results

In this section, we show the benefits of our WarpDriver algorithm with various types of results, including large, dense test cases, scenarios with non-linear routes, history-based anticipation cases and a highly-constrained situation. Finally, we present the results of the benchmarks on previously studied data and the performance of our algorithm.

7.1 Large and Dense Cases

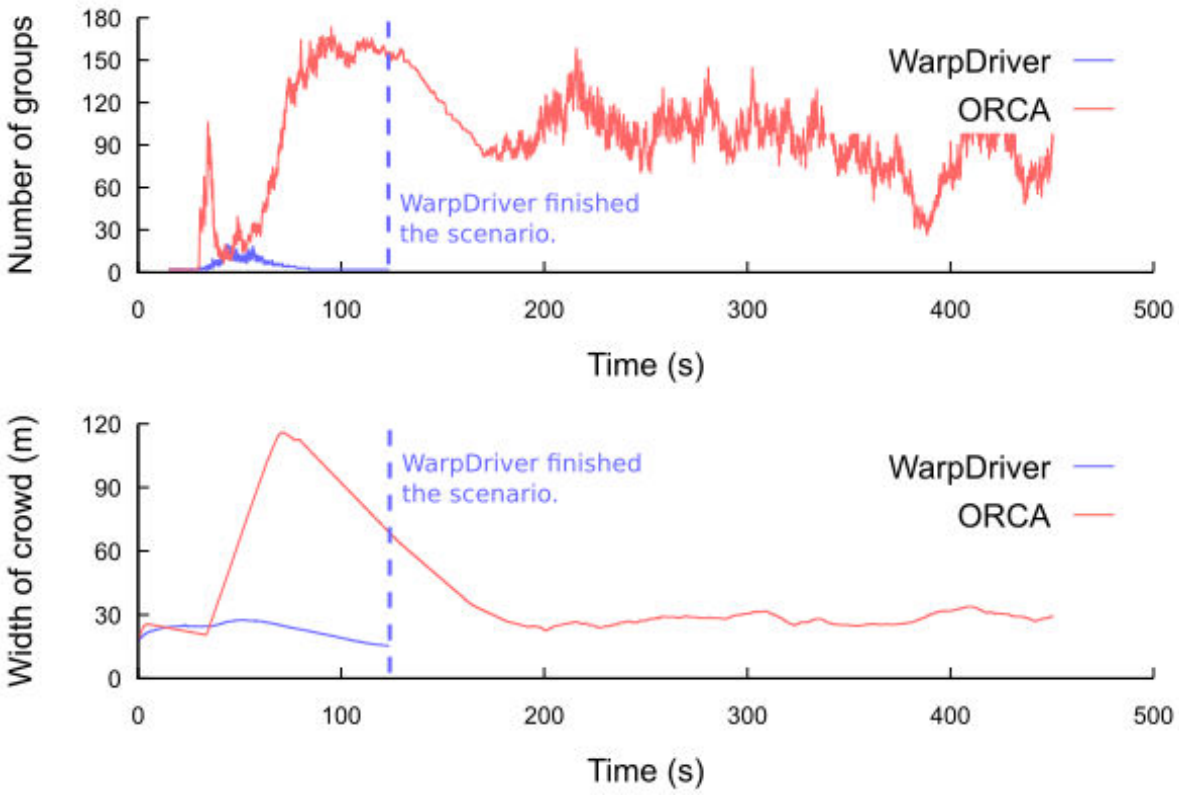
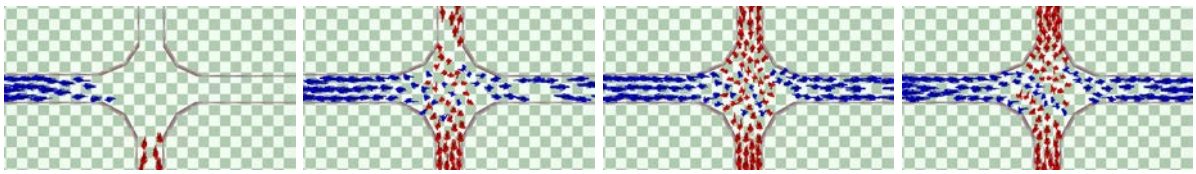
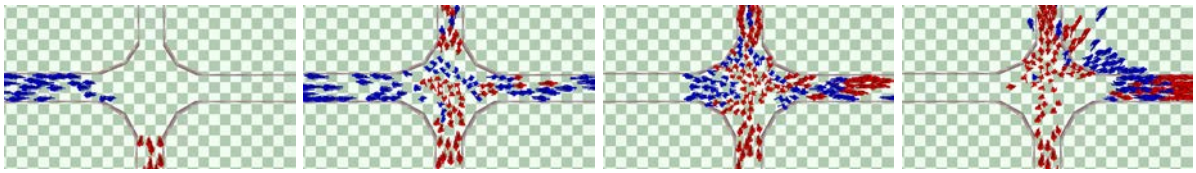


Figure 4.5 – Dual Big Groups example. Top: number of emerging sub-groups. Low number for WarpDriver and high number for ORCA. Bottom: spread of agents. WarpDriver agents stay compact, ORCA agents spread widely. Note that WarpDriver achieves simulation goals four times faster than ORCA.

We start with simulation tests involving a large number of agents and high densities (agents are within contact distance of each other), testing our algorithm's ability to navigate agents while subject to many, simultaneous interactions.



(a) WarpDriver.



(b) ORCA.

Figure 4.6 – Crossing example: two flows of agents in corridors cross at a right angle (red ones going to the top and blue ones going to the right) simulated by WarpDriver (top) and ORCA (bottom).

7.1.1 Test case 1: Big Groups

This first test case involves two 1027-agent groups exchanging positions as seen on Figure 4.1. In this kind of example, we expect agents to be able to traverse through the opposing group and reach their destinations. This expected behavior implies a certain level of organization of the agents; thus we measure how many sub-groups emerge (using the method from [ZTW12]) and how widely agents might spread (Figure 4.5, top and bottom respectively).

7.1.2 Test case 2: Crossing

The second test case involves two corridors intersecting at a right angle, each with a uni-directional flow of agents (Figure 4.6). This kind of situation is well studied and 45° lines should form between agents of each flow at the intersection [CARH13], facilitating their movement. We measure this by detecting sub-groups with the previously mentioned method and perform linear regression on the agents, results are reported on Figure 4.7(middle, bottom). Furthermore, as the situation is very constrained (agents at contact distance from each other with the presence of walls), we also measure how many agents are jammed (travel at less than 0.1m/s) during the simulations, as shown in Figure 4.7(top).

7.1.3 Analysis

In the Big Groups example (Figure 4.1), agents simulated with our algorithm are able to do two things. First, front-line agents are able to find points of entry in the opposing group (which correspond to the minima of the collision probability function) and consequently enter through them. Second, non-front-line agents are able to anticipate the front-liners' continuing motion and align themselves behind them. In the resulting motion, agents re-organize themselves into lanes and are able to fluidly reach their destinations. This re-organization can be observed through the low number of emerging

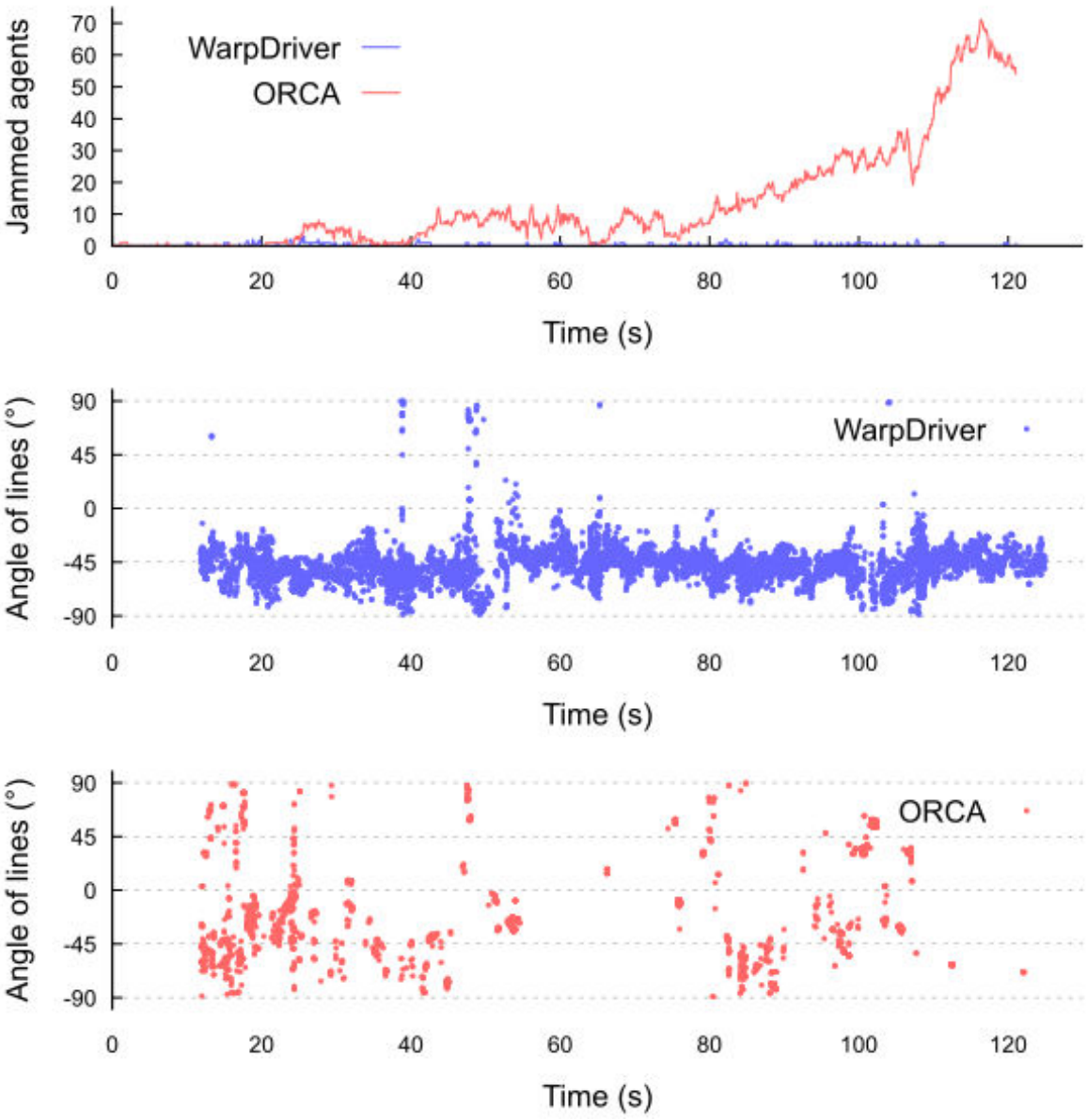


Figure 4.7 – Crossing example. Top: number of jammed agents (speed lower than 0.1 m/s). WarpDriver has almost none while ORCA has increasingly high numbers. Middle: angles of agent lines formed at intersection for WarpDriver, lines form with angles around -45°. Bottom: angles of agent lines formed at intersection for ORCA, very few lines form (absence and/or disorganization of agents).

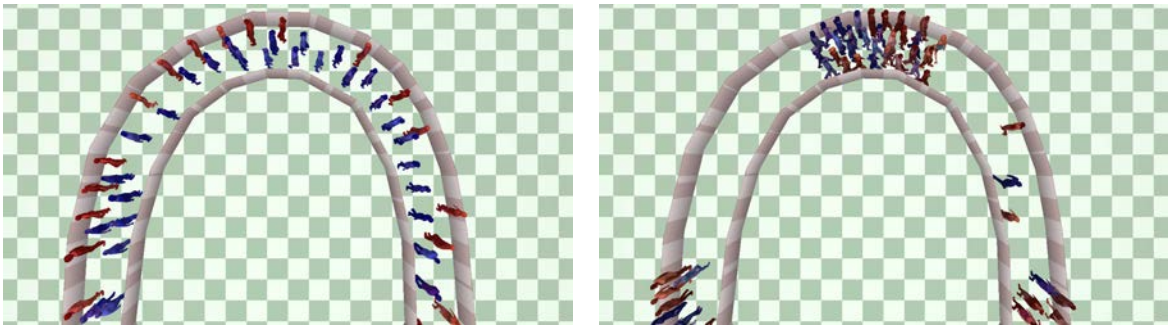
sub-groups (Figure 4.5(top)) which correspond to the formed lanes, and through the relatively low spread of the agents (Figure 4.5(bottom)). In the ORCA simulation, however, when agents of both groups meet, their solution spaces are quickly saturated, thus forcing them to start spreading on the sides in order to free up the velocity space and be able to continue their motion. As a result, groups collide and spread until agents are able to pass through to their goals and that takes much more time. This disorganization can be observed through the high number of emerging sub-groups (Figure 4.5(top)) cor-

responding to agents searching for less saturated solution space, thereby spreading over larger distances (Figure 4.5(bottom)). Also note that our algorithm solves the situation much more quickly than ORCA (much shorter blue graphs than red ones in Figure 4.5).

In the Crossing situation (Figure 4.6), as expected agents simulated with our algorithm are able to cross without congestion (Figure 4.7, top: no jammed agents) forming the expected 45° crossing patterns (Figure 4.7, middle). ORCA agents on the other hand, quickly get into a congestion (Figure 4.7, top: increasing numbers of jammed agents) and no patterns can be found, explaining the lack of points on the bottom graph in Figure 4.7.

Overall, our algorithm is able to better find (and take advantage) of small spaces between agents (local minima in the collision probability fields) thus producing more visually pleasing results than ORCA, which often has more binary reactions, leading to stopping agents in congested scenes.

7.2 Non-Linear Scenarios



(a) WarpDriver. (b) ORCA.

Figure 4.8 – Curved Flows example with agents on a curved path (blue ones turn clockwise, red ones counter-clockwise) simulated by WarpDriver (left) and ORCA (right).

With the following test cases, we investigate how our algorithm copes with situations where agents' future motions are non-linear. To this end, we make agents interact with each other and with obstacles, while traveling along curved paths.

7.2.1 Test case 3: Curved Flows

In this situation (Figure 4.8), we set two opposing flows of agents (moderate density, about a meter between agents) in a curved corridor. Here, with the moderate density, we expect agents to fluidly navigate to the other end of the corridor. To measure the impact the curved corridor has on the agents, we reproduced the experiment in all aspects (same number and density of agents, same corridor length and width) except for one: we made the corridor straight. We then measured the average speed of the agents first in the straight version and then in the curved version, as seen in Figure 4.9.

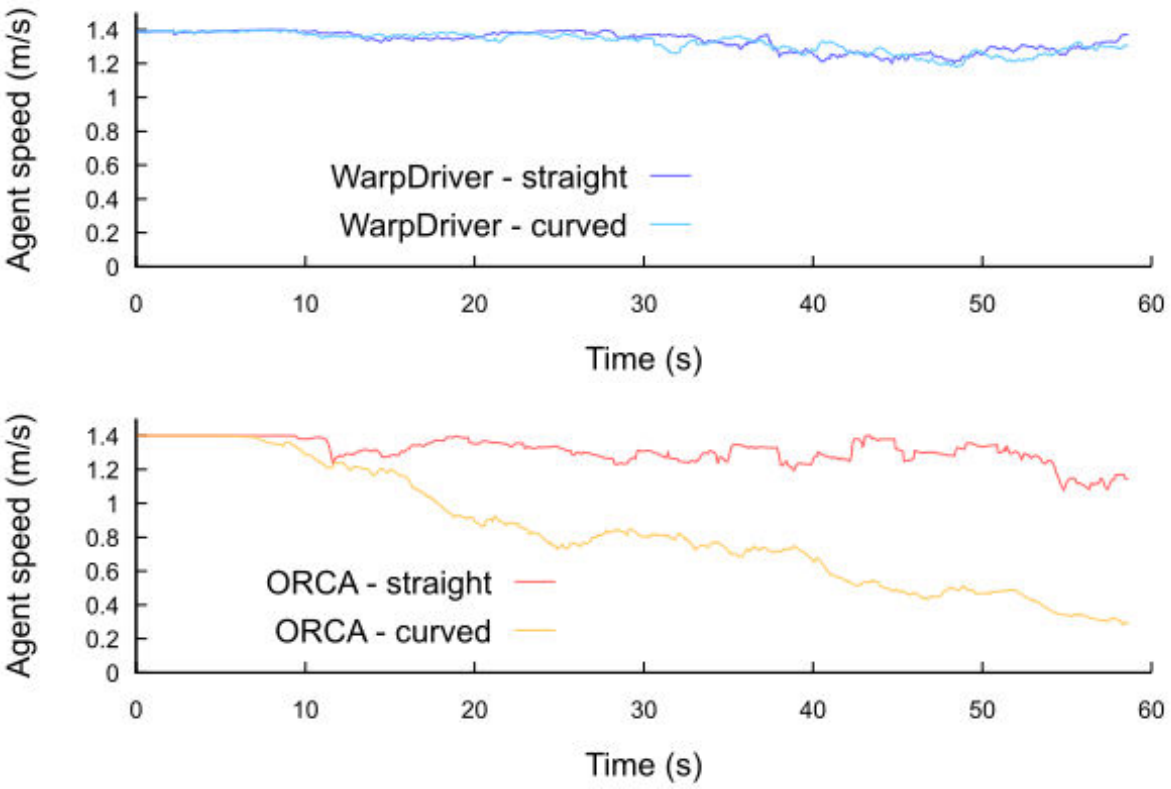
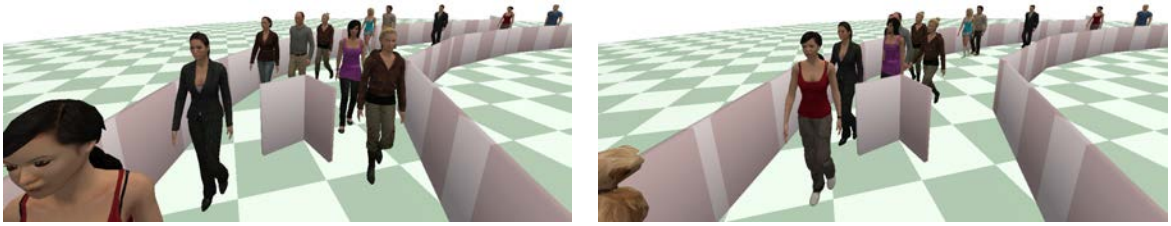


Figure 4.9 – Curved Flows example. Top: average speed of agents in curved vs. straight corridors for WarpDriver, the speed remains similar. Top: average speed of agents in curved vs. straight corridors for ORCA, an important loss of speed occurs in the curved path.



(a) WarpDriver. (b) ORCA.

Figure 4.10 – Curved Obstacle example with a small obstacle on a curved path simulated by WarpDriver (left) and ORCA (right).

7.2.2 Test case 4: Curved Obstacle

This situation is a simplification of the previous test case: one uni-directional flow of agents is made to travel the same curved corridor with one small obstacle in the middle as shown on Figure 4.10. In this simple test case, we expect the agents to easily bypass the obstacle on the side that is most direct, i.e. if an agent is on the outer (resp. inner side) side of the corridor it should bypass the obstacle on the outer side (resp. inner side). We thus looked at the paths agents followed (Figure 4.11.

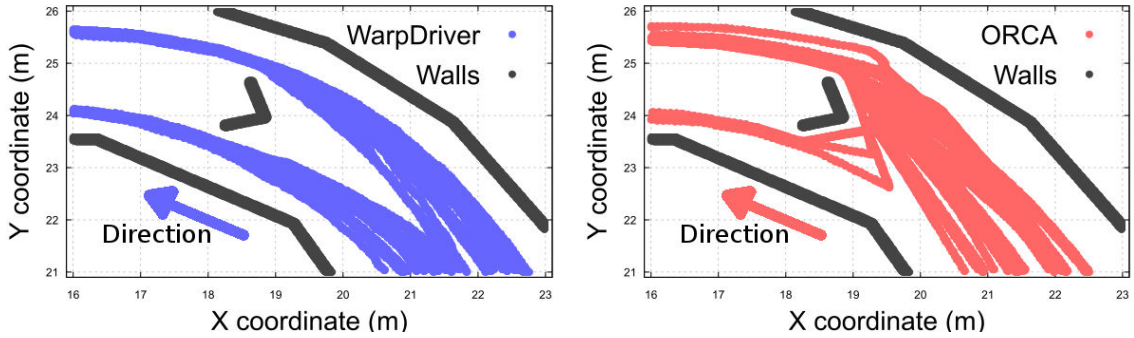


Figure 4.11 – Curved Obstacle example. Top: agent traces obtained with WarpDriver, agents bypass the obstacle on both sides. Bottom: agent traces obtained with ORCA, agents bypass the obstacle on the right (unless too crowded, then on the left).

7.2.3 Analysis

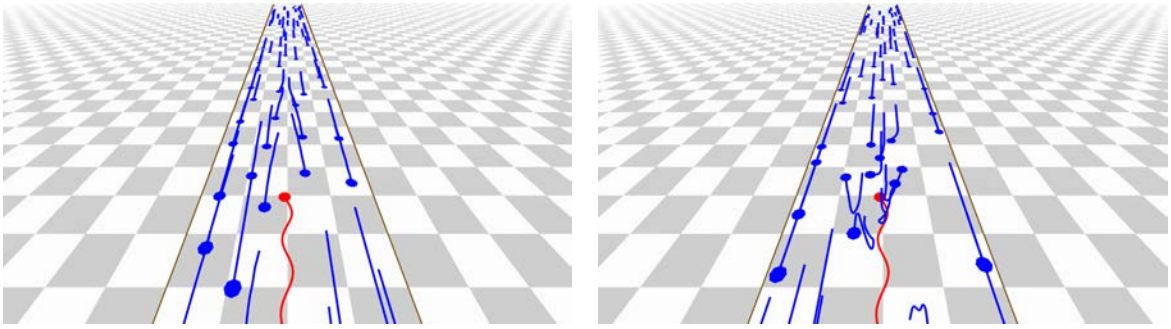
In the Curved Flows example (Figure 4.8), agents simulated with our algorithm are able to avoid each other correctly, eventually leading to the emergence of two opposing lanes (red agents travel on the outer rim while blue agents are on the inner rim). In Figure 4.9, we can see that using our algorithm, agents travel at the same overall speed in both the straight and curved versions. With the ORCA algorithm, on the other hand, agents quickly get stuck in a congestion (Figure 4.8). We can observe this in Figure 4.9, with an important loss of agents' speed on the curved version of the corridor as compared to the straight one.

The Curved Obstacle situation shows the phenomenon more clearly. With our algorithm, agents anticipate the obstacle about 3m in advance (see Figure 4.11, left) and choose the most direct (expected) side. ORCA agents on the other hand can be seen to all prefer the outer side (with respect to the curve) of the obstacle and some agents backtrack (see Figure 4.11, right) and use the inner side when a bottleneck situation forms.

The difference between algorithms in both cases is that agents simulated with WarpDriver anticipate their own (and others') future trajectories as curved along the corridor, thus perceiving interactions where they most probably will occur (thus they see the opposite-flow agents and obstacle from the two previous examples well in advance). ORCA agents in these cases exhibit artifacts linked to their linear extrapolation of trajectories based on instantaneous velocities: they can only perceive interactions that will occur roughly on a line tangent to the corridor curve at their position (thus they do not see the opposite-flow agents nor the obstacle from the two previous examples until the very last moment).

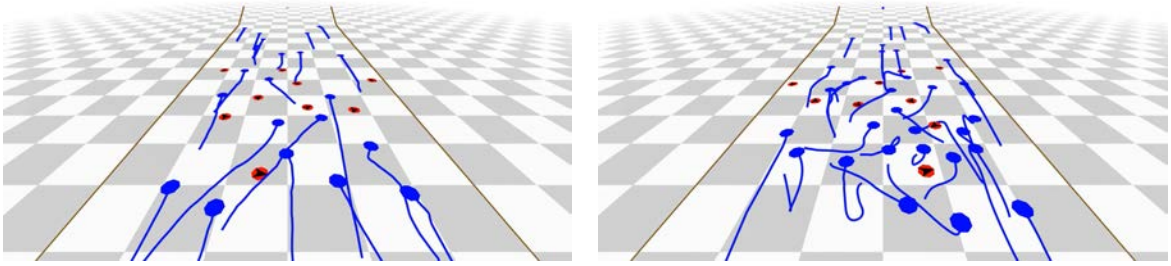
7.3 History-based Anticipation

As instantaneous velocities can vary very rapidly and not be representative of agents' overall motions, we next test situations where agents or obstacles behave according to pattern-like movements. We test two easily-recognizable behaviors: zig-zagging and revolving motions.



(a) WarpDriver. (b) ORCA.

Figure 4.12 – Zig-Zag example, agents (blue) avoid a zig-zagging agent (red) simulated by WarpDriver (left) and ORCA (right).



(a) WarpDriver. (b) ORCA.

Figure 4.13 – Danger Corridor example, agents (blue) avoid turning obstacles (red) simulated by WarpDriver (left) and ORCA (right).

7.3.1 Test case 5: Zig-Zags

In this scenario (Figure 4.12), we set up a uni-directional flow of moderately-spaced agents (in blue) traveling along a straight corridor and further add an agent (in red) which travels counter-flow with a zig-zagging trajectory. Figure 4.12 shows the setup where the red agent has a narrow zig-zagging motion (another version with wider zig-zags can be found in the companion video). In this example, we expect the blue agents to recognize and anticipate the red one's motion pattern and easily avoid it. We measured how easily blue agents are able to avoid the red one by recording the angle between the agents' orientation and their goal direction (their deviation from their goal) on Figure 4.14(top: narrow zig-zags situation, middle: wide zig-zags situation). We also report what proportion of the simulated frames contain backtracking agents (180° deviations) on Figure 4.15.

7.3.2 Test case 6: Danger Corridor

This scenario (Figure 4.13) is largely similar to the previous one in that a uni-directional flow of agents (in blue) travel down a corridor, except that we set nine slowly revolving pillars (in red) in the middle of the path. We then expect agents to be able to recognize

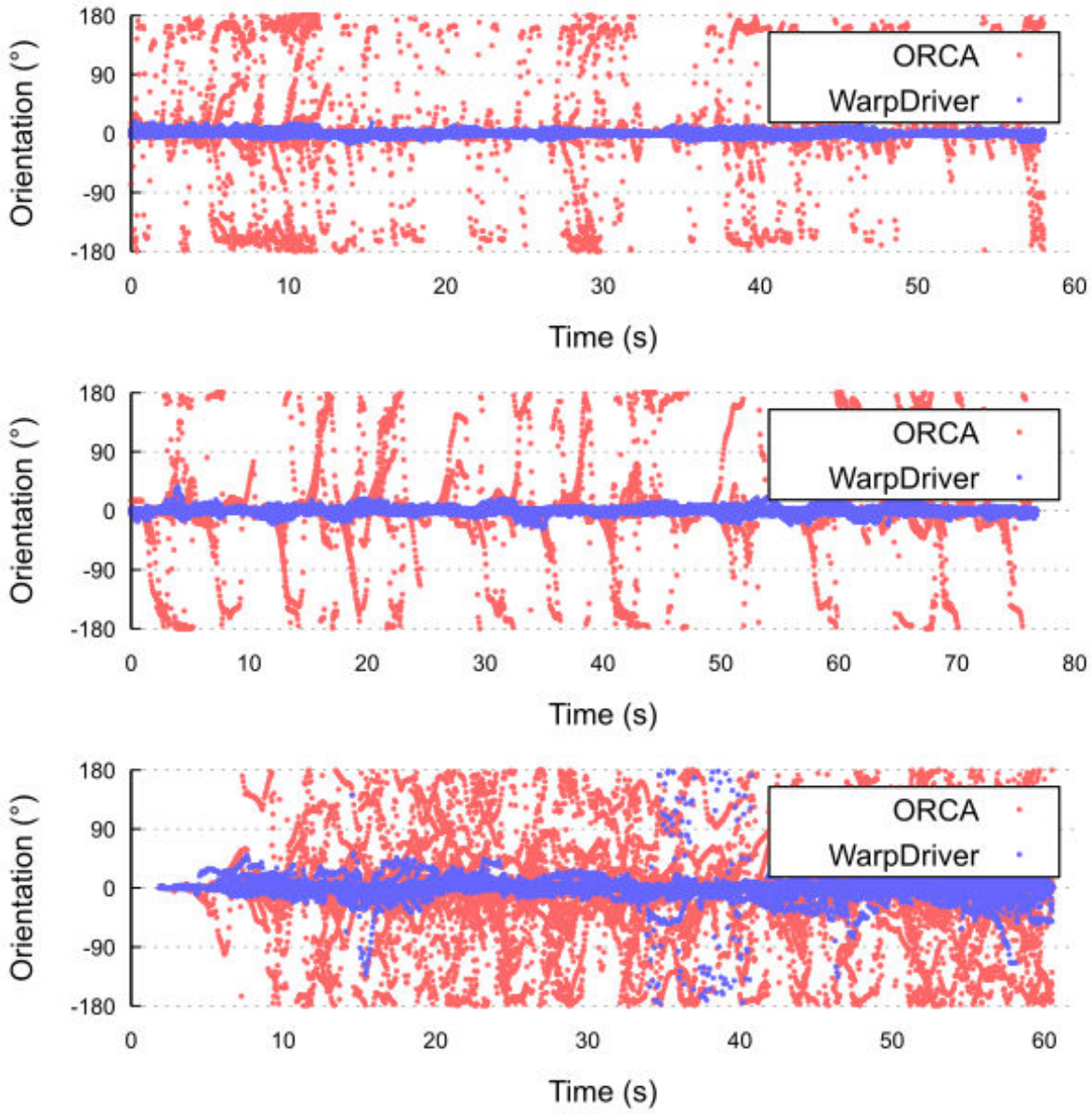


Figure 4.14 – Top: Zig-Zag example, deviation angles of agents from their goals for narrow zig-zagging motions. Middle: Zig-Zag example, deviation angles of agents from their goals for wide zig-zagging motions. Bottom: Danger Corridor example, deviation angles of agents from their goals. Overall: WarpDriver agents deviate very little from their intended trajectories while ORCA agents deviate much more.

how these pillars move and easily work out a path through them. Again, we measure the agents' deviation form their goals which we report on Figure 4.14(bottom) and the proportion of frames containing backtracking agents on Figure 4.15.

7.3.3 Analysis

In the Zig-Zag example (Figure 4.12), agents (in blue) simulated with WarpDriver are able to anticipate the zig-zagging agent (in red) in advance and minimally adapt their

Figure 4.15 – Narrow/Wide Zig-Zags and Danger Corridor examples, proportion of frames containing backtracking agents per example. WarpDriver has no frames with backtracking agents (except a few in the last example) while the majority of frames simulated with ORCA contain backtracking agents.

trajectories to avoid it. This is confirmed by Figure 4.14(top and bottom, corresponding to narrow and wide zig-zags respectively) which show that the heading direction of the agents is very close to 0° (heading in their preferred direction). ORCA agents, on the other hand, have more trouble anticipating the jerky motion of the zig-zagging agent and noticeably over-react as a result. This is confirmed by Figure 4.14(top and bottom) where agents often deviate by ±180° (backtracking from their goal).

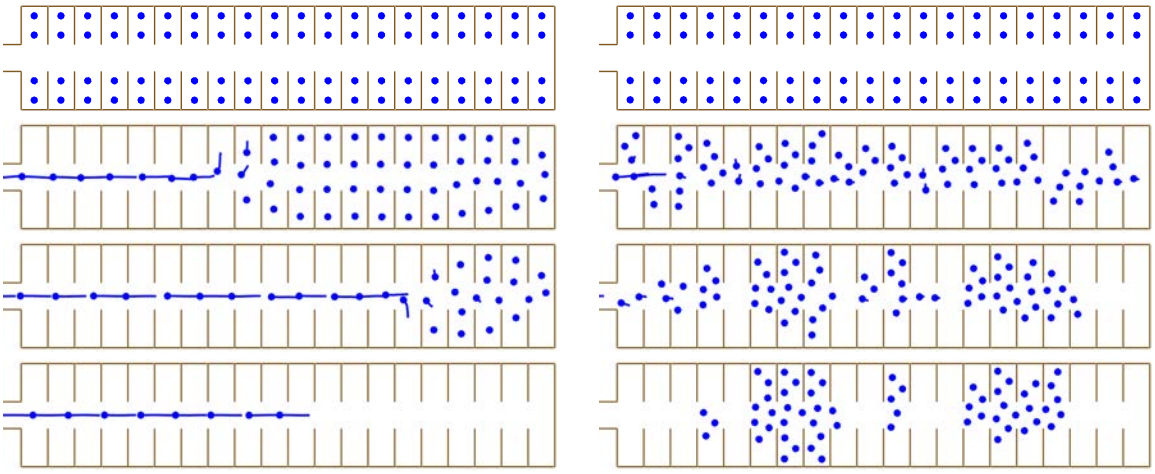
The Danger Corridor example (Figure 4.13) yields results largely similar to the Zig-Zags one (but more pronounced). WarpDriver agents are able to fluidly avoid the revolving obstacles while ORCA agents have more trouble doing so. Again, Figure 4.14(bottom) shows the agents' deviations from their goals (low for WarpDriver and high for ORCA).

Additionally, as shown in Figure 4.15, when using WarpDriver the resulting simulations contain no backtracking agents for Zig-Zags and only 11% of frames contain backtracking agents in the Danger Corridor. When using ORCA, on the other hand, the Narrow Zig-Zags, Wide Zig-Zags and Danger Corridor are simulated with respectively 68%, 54% and 90% of frames containing backtracking agents.

Overall, the differences can be explained by the fact that in these cases, the instantaneous velocities of the zig-zagging agent and revolving obstacles are constantly changing and their trajectories are not straight. Thus, ORCA first linearly extrapolates (incorrect) future motions and then faces these extrapolations constantly changing. ORCA agents thus avoid many, ever-changing and possibly non-existent future interactions. These artifacts are addressed by WarpDriver: first, it detects patterns in the past motions and learns from them to anticipate future motions; second, when anticipating future motions it does so non-linearly. As a result, WarpDriver agents are able to correctly anticipate

and avoid colliding contacts with other agents.

7.4 Highly-constrained Case



(a) WarpDriver. (b) ORCA.

Figure 4.16 – Plane exit situation simulated by our method (left) and ORCA (right).

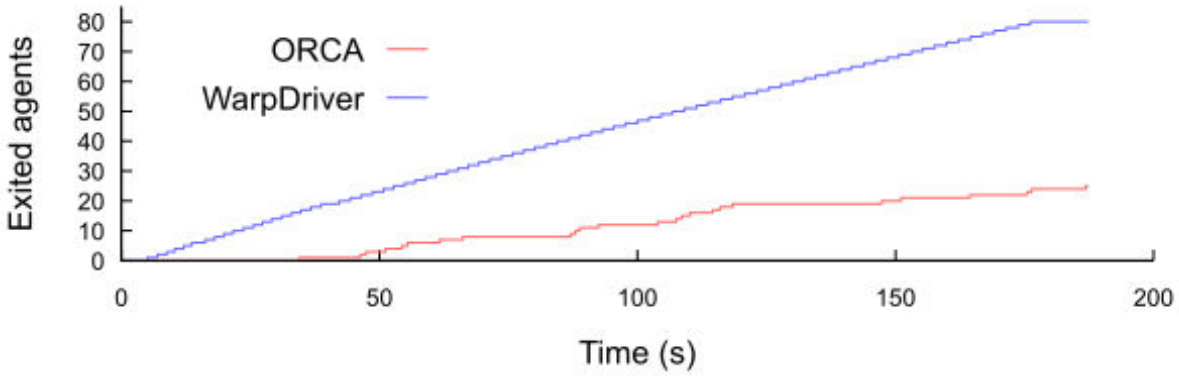


Figure 4.17 – Plane example: number of agents evacuated during the simulation. WarpDriver agents evacuate regularly while ORCA agents get congested and have trouble evacuating.

In the last scenario, we test how our algorithm copes with highly-constrained scenarios in confined spaces, where agents are within contact distance and many encounter path intersections.

7.4.1 Test case 7: Plane

This example features a plane egress scenario with 80 agents. Figure 4.16 features two sequences of stills, one for each algorithm (left: WarpDriver, right: ORCA) captured at

the same moments, showing the successive states of the simulations. Here, we expect agents to orderly exit the plane starting with the ones close to the exit and with more far-away agents exiting last. To see how agents are able to cope with this situation, we recorded how many agents have successfully exited the plane during the simulation on Figure 4.17.

7.4.2 Analysis

On the left sequence of stills from Figure 4.16, with our algorithm (left column), agents in the back allow agents up front to exit first. This behavior leads to an orderly exiting process, where all agents are progressively evacuated as evidenced by the regular blue plot from Figure 4.17. On the contrary, with the ORCA algorithm (right column), all agents try to exit at the same time which, with the very constrained space (little room for maneuvers) leads to dead locks. This general difficulty of the agents to exit is confirmed by the stagnating number of evacuated agents depicted by the red graph from Figure 4.17.

The behavior obtained with our algorithm results from a combination of factors. First, agents are able to predict which way the others will go: into the alley and then towards the exit (note that these paths are non-linear since they contain a right turn). Second, agents in the front rows are closer to the exit than the others and they are thus perceived as obstacles blocking the exit from the other agents (and conversely front agents perceive other agents as being "behind"), thus creating a hierarchy. Finally, the agents easily navigate between the chairs by following the local minima of the collision probabilities defined by these obstacles. On the contrary, ORCA agents navigate with their solution-spaces saturated (both by obstacles and other agents) and without any sense of emerging hierarchy.

7.5 Benchmarks

Finally, we compared our algorithm with the Boids [Rey87], Social-Forces [HM95] and ORCA [vdBLM08] on previously-studied test cases using the method from [WGO+14]. In these tests, the difference between our algorithm and the others is not always as pronounced as in the previously shown scenarios. This is due to the nature of the available ground truth data which only captures simple interactions: (1) simple crossing situations between 2-5 agents, and (2) 6-24 agents exchanging positions on a circle. Nonetheless, as Figure 4.18 shows, on these test cases, our method (in blue) gives comparable results to ORCA (red), occasionally outperforming it (and almost always outperforming the other algorithms).

The benefits of our algorithm shown here over previous methods come at a higher computational cost, however. We measured this cost by simulating increasing numbers of agents in the worst possible configuration: circles of agents trying to get to the antipodal positions (an example with 200 agents can be found in the companion video). The reason why this configuration is the most computationally intensive for our algorithm is that our current implementation has no limit on neighbors. Additionally in this configuration, agents all initially converge towards the center of the circle. Consequently, all agents interact with all other agents, thus reaching our algorithm's maximum complexity of

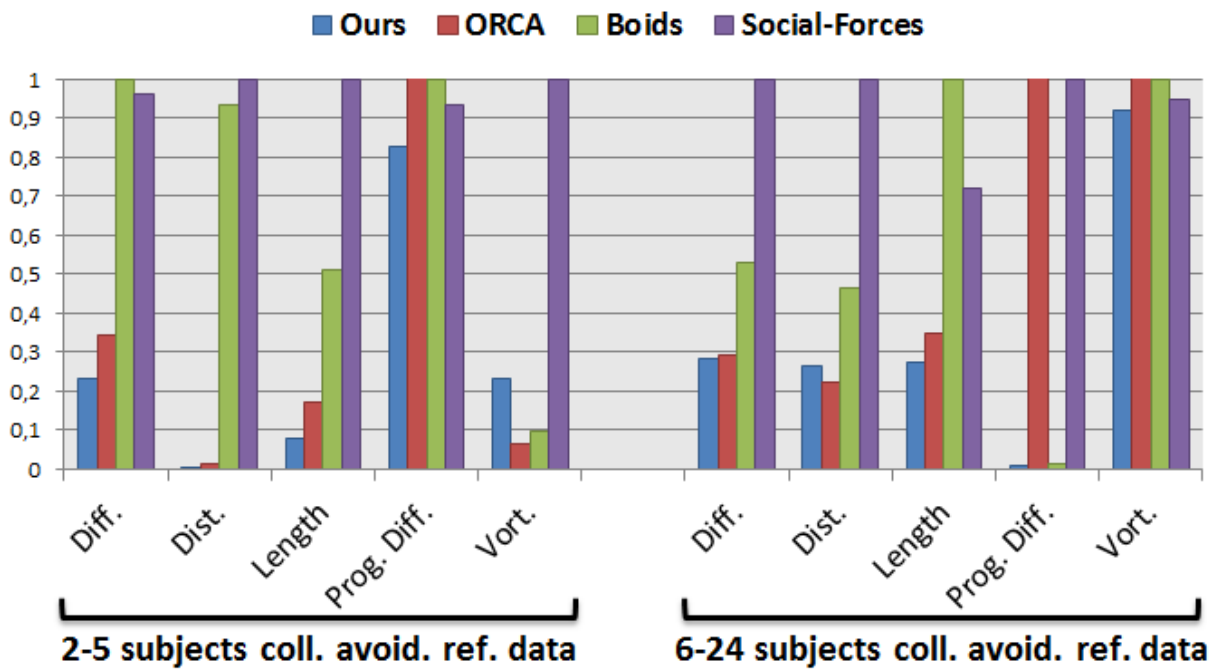


Figure 4.18 – Benchmarks results using the method from [WGO+14], lower is better.

$O(n^2)$. We report the performance of our algorithm compared to ORCA (super-linear complexity thanks to neighbor culling and a space-optimizing structure) in frames per second in Table 4.1.

Many options can be explored to improve the performance of our algorithm as compared to its current un-optimized implementation. Some simple options include better selection strategies when sampling along the *projected trajectory* (currently fixed step, fixed number of samples), choosing/limiting neighbors, and space-optimization structures (e.g. kd-tree). And more optimization strategies can be borrowed from the ray-tracing literature, such as parallel sampling, level-of-detail on *perceived* agents' collision-probability fields, caching, etc.

Table 4.1 – FPS per number of agents.

	10	100	200	300	400	500
WarpDriver	1555	46	16	9	6	5
ORCA	156272	10289	4468	3107	2260	1920

8 Discussion and Limitations

We present a novel probabilistic motion prediction algorithm for crowd simulation that takes into account of the contextual interaction between the agent and its surrounding, including other agents, the environment layout, motion anticipation, etc.

We assume that the environments can be annotated with probable routes to be followed by agents. This step does not present any difficulty – it just needs to be done once for each new environment. This environment annotation could be easily automated in

various ways. First, the geometry of probable routes could be extracted automatically based-on smoothed Voronoï diagrams or any technique to compute static obstacles' medial axes, or even learned from real data (e.g. camera feeds). More interestingly, our representation could be extended with route selection probabilities.

Although in this chapter we have limited the application of motion prediction to mostly collision avoidance for crowd simulation, motion prediction is generally at the core of numerous types of interactions among humans and it represents the most basic software module of all crowd simulators. Thus, our crowd simulator can and should be extended to handle other types of interactions, including following, fleeing, intercepting, grouping behaviors, etc.

A possible limitation concerns our probabilistic modeling of risk of collision. The current implementation does not make any distinction among various origins of the collision risks. As a result, for example, equivalent collision probabilities between a neighboring agent moving in the same direction and one moving in the opposite direction are processed the same way. They, however, do not result in the same energy of collision, which should be integrated into the notion of risk of collision. Theoretically, our method can handle any kind of moving obstacles. Extending the notion of risk of collision would allow us to mix into our simulations other types of moving obstacles (e.g. cars) with their corresponding dangerousness.

Addressing each of these issues can lead to promising directions for future work. While we have presented noticeable improvements in terms of motion quality for agents in this work, investigating each of these aspects would likely result in new-generation crowd simulators that may match real observations even more accurately in the near term.

9 Conclusion

A new motion prediction algorithm for crowd simulation has been presented in this chapter. The main novelties of this approach are two-fold.

The first novelty is the non-deterministic and probabilistic nature of our motion predictions. Agents do not perceive future collisions in a binary manner like in most of the existing methods; instead, they perceive a probability of future collision. This approach has several advantages. First, this characteristics results in smoother motion because the probability field is continuous. Agents adapt their motion to lower the probability of collision by following the gradient of the probability field. Second, some agent's oscillations between two binary future collision states that could be observed in previous techniques are avoided. Third, our anticipation considers several possible hypotheses, the notion of routes can be used when future position probabilities are propagated in time. Fourth, the non-determinism allows us to simulate uncertainty due to sensing or variety in locomotion trajectories. As we increase the uncertainty of agents' future positions the further they are in time, we change the relative importance of agents that may collide sooner as opposed to those that may collide later. This aspect makes a large difference as compared to most of the previous algorithms, which give the same importance to all neighboring agents.

The second innovation is related the contextually-aware technique, which not only

depends on agents' states, but also on external and contextual cues. This makes a major difference with all previous methods that assume agents keep moving with the same current velocity vector. One can easily conceive that agents' current velocity vectors are, most of the time, not representative of the intention of future motion, especially in crowds, where we are constantly adapting our locomotion trajectory to the presence of others. For this reason, our motion anticipation is based on factors such as the environment's layout, interactions with other agents, the recent history of agents' movements, etc. and can be extended to include social protocols for instance. In addition, the mechanism to update agents' velocities in order to avoid highest risks of collision is also revisited accordingly. Whereas previous techniques searched for admissible velocities that presented no risks of future collision in the near term, our technique aims to lower the risk of future collisions.

Through a set of challenging evaluation scenarios, as well as quantitative evaluations, we have demonstrated that we considerably improve the quality of visual simulation of crowds and alleviate visual artifacts commonly known in the current state-of-the-art collision-avoidance algorithms. There are several avenues for future research, as listed in the previous section. We plan to address a number of them first. We would like to adapt our simulator to consider other types of local interactions than collision avoidance. More importantly, our system also opens new possibilities to be explored. One promising research direction is to learn future position likelihood based on real observations. This would allow us to automatically adapt our simulator to a specific situation. In a given place, the probability of future positions depends on the nature of people who frequent this specific place, and on the exact activities they perform there. Without the need to explicitly specify this knowledge, we could easily learn the resulting probability fields. It enables crowd simulations to reach a new level of predictive capability that is not possible with existing solutions.

Applications to Evaluation and Parameter Estimation

5

Contents

In Chapter 3, we introduced a framework for the evaluation of crowd simulation algorithms and the estimation of their parameters. In that context, we applied it to the evaluation of algorithms, and also showed how it could help users in achieving certain simulation goals. In this chapter, we show how this work can be used to approach the question of the validity domain of each simulator: determining for each application which simulation algorithm is the most appropriate. Thus, we use our framework as a selector, first on a per-simulation basis, and second on a per-moment basis.

The first application deals with insect simulation. There, our framework allows to select the simulation algorithm which best describes a given behavior: that of the swarming insects in our ground-truth data. The selected simulator (also correctly configured parameter-wise) is used for the local interactions of our simulated insects.

The second application deals with pedestrian tracking. There, our framework allows to select the simulation algorithm which best describes a crowd at a given moment: the instantaneous navigation decisions of pedestrians in data acquired in real time. The selected simulator (also correctly configured parameter-wise) is used to predict the next state of the tracked crowd.

1 Application to Insect Simulation

Figure 5.1 – Simulation of butterflies moving on a prairie.

1.1 Introduction

This chapter describes a project on which I collaborated with Weizi Li, a PhD student at the Gamma Group from the University of Carolina at Chapel Hill, NC, USA. It proposes an application of the evaluation and parameter estimation framework presented in Chapter 3.

The work presented in this chapter is geared towards insect simulation, which in graphics, with increasingly realistic animation of various life forms, can enhance the visual realism of many graphical applications from virtual environments, to computer games, and cinematography. The variety of insect behaviors, arising from the heterogeneity in sensing and cognitive abilities, survival strategies, interpretations of environmental factors etc. [CDF+03] makes the simulation of insects a challenging computational problem. A complete model able to capture the different and complex phenomena is conceptually difficult and can introduce an near unmanageable amount of parameter tuning and tweaking. To overcome this difficulty, we propose a general, biologically-inspired framework to analyze, evaluate, and simulate insect swarms. We also evaluate simulation results both qualitatively and quantitatively.

Swarming insects often exhibit complex global patterns, yet frequently combines local randomness and chaotic individual movements. The motion of insect swarms can be generally decomposed into different spatial scales [Oku86]. At the microscopic scale, insects have local interactions among them (e.g. collision avoidance). At some intermediate mesoscopic scale, insects move in some preferred direction and form trajectories of various shapes. At the macroscopic level, more coherently, global structures emerge from collections of individual movements. Various reasons have been proposed to what drives the overall direction of an insect swarm, including food seeking and hunting, predator avoidance and evasion, courtship and mating, and seasonal migration, etc.

Given the complex causalities leading to insect swarming, we propose a multi-scale approach that is biologically inspired [Sum10] and experimentally validated. Our objective is to recreate realistic visual simulation of insect swarms by learning from real insect

motion data. We analyze the insect motion data at micro-, meso- and macroscopic scales to configure and estimate parameters of various components residing in our simulation system. More specifically the captured data enables us to (1) automatically select and configure the most suitable local steering algorithm, (2) generate statistically accurate waypoints at the mesoscopic level using segmented piecewise-linear trajectories, and (3) parameterize a global guiding field that captures swarming spatial patterns.

Since insect motion data is not always readily available and sometimes animators prefer to have direct control over the resulting simulation, our system can further serve as an assistive animation tool that empowers artists and amateurs. Our system can interpret sketch-like input and generate similar patterns; or the user can provide "guiding simulation results" with specifications on motion models and related parameters, our simulation can then *re-target* the motion onto different swarming behaviors.

In summary, the main contributions of this work are the following:

- A general, biologically-inspired system is developed to analyze, evaluate and simulate insect swarms (Section 1.3).
- Comparative evaluations of various local steering algorithms are performed to select and configure the best model for insect local collision avoidance (Section 1.3.2).
- A statistical learning and evaluation approach is developed for modeling individual behaviors within the swarm and preserving spatial structures of the swarm (Section 1.3.2 and 1.3.2).
- The resulting framework can serve as an assistive animation tool for users to either turn sketched patterns into animations or utilize existing simulations for retargeting purposes (Section 1.4).

1.1.1 Working Arrangements

While Weizi focused on the mesoscopic and macroscopic parts of the system, assembling the complete system together and generating the results, my primary focus was at the microscopic level: the adaptation of the evaluation and parameter estimation framework from Chapter 3 (mainly retaining an advisory role for the rest). As such, the "technical section" of this chapter also focuses on this same part, only briefly describing the rest of the approach.

The rest of the chapter is then organized as follows. In Section 1.2, we review existing work in the field of pedestrian tracking. The approach is described in Section 1.3, and results are presented in Section 1.4. Finally we conclude in Section 1.5.

1.2 Related Work

Insect swarms can be viewed as multi-agent systems, which have been a flourishing research topic for nearly three decades. Many aspects including local collision avoidance, global path planning, behavior modeling, group motion and user interaction have had striking results. A large portion of simulation aspects applicable to insects have been addressed under the focus of crowd simulation. In this section, we will first discuss

techniques that were proposed to generate agents' local and global behaviors from the point of view of graphics applications, then proceed to approaches developed from the perspective of swarm studies in biology, physics and mathematics.

1.2.1 Graphics Point of View

From the graphics point of view, a swarm of insects can be approached with crowd simulation techniques. At the microscopic level, for local collision avoidance, many algorithms can be used. As already listed in Chapter 2, Section 2, these algorithms can range from first-order ones such as Boids [Rey87] and Social Forces [HM95], to second-order ones including velocity obstacles [vdBLM08], future collision-place avoidance [KHBO09], synthetic vision [OPOD10], etc. The difference with human pedestrian applications however, is that insect swarms present more readily visible global behaviors. While these local collision-avoidance algorithms can lead to emergent patterns at the macroscopic level, they alone may not account for all observed macroscopic swarm behaviors.

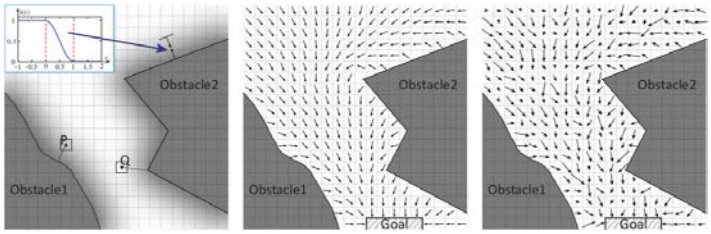
These macroscopic aspects could then be taken care of by using macroscopic algorithms (see Chapter 2, Section 3.4) which model crowds using scalable quantities such as density [Hug03, TCP06, NGCL09]. Additonally, these techniques can be made more flexible by using potential fields [TCP06] and navigation fields [PvdBC+11] to alter agent flows. Furthermore, techniques have been devised to edit group motions, which can generate believable agent formations. Approaches include the work by Kwon and colleagues [KLLT08] who model group motions as graphs whose vertices represent agents' positions at given frames and which can be deformed, split and merged as necessary. Takahashi and colleagues [TYK+09] model formations using connectivity graphs from which they can extract Laplacian matrices; by interpolating between matrices (and thus underlying graphs) corresponding to different foramtions, they are thus able to transition from formation to formation.

Following from these approaches to macroscopic crowd simulation, Wang and colleagues [WJDZ14] proposed an insect swarm simulator where agents are advected along two fields. A three-dimensional velocity noise field, procedurally generated using the Perlin noise algorithm and further modified to be divergence-free and bound, models the noisy nature of insects' individual trajectories. A three-dimensional velocity, minimum-cost field represents the path (also allowing the swarm to avoid environment obstacles, splitting itself if necessary) that agents should take in case of a migratory behavior. The combination of these fields yields coherent swarms which can move around in an environment avoiding large obstacles.

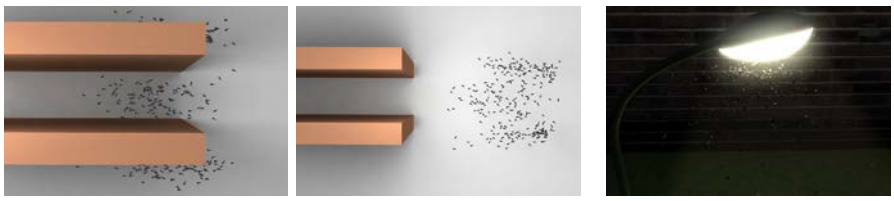
1.2.2 Biology Point of View

In addition to the previous multi-agent analogy in graphics, a swarm of insects can be more generally viewed as a self-organized particle system. As such, swarms are thus one of many possible applications to research lead in the fields of mathematics and physics, in addition to obvious interest in the field of biology. From these perspectives and similar to what can be found in graphics, both individual-based and continuum approaches exist.

Individual-based approaches include self-propelled particles (SPP) [VCBJ+95] and their variants [CkJ+02, DCBC06], force-based techniques [FGLO99] and Brownian particles [SESG08]. Again, parallels can be made between these techniques and those found



(a)



(b) (c)

Figure 5.2 – Insect simulator [WJDZ14]. (a) Environment with obstacles. The signed distance to obstacles (left) is used to compute the minimum-cost field (center), further combined with the noise field (right). (b) A swarm which was previously split to avoid obstacles (left) merges back into one (right). (c) A synthesized swarm of moths near a lamp.

in graphics. Self-propelled particles for instance, are defined to have a velocity which has a fixed norm and a direction that is the mean direction of the velocities of neighbor particles to which some noise is added, reminiscent of the "alignment" rule from the Boids algorithm ([Rey87], Chapter 2, Section 2.1). Even with simple rules, these individual-based approaches have been experimentally confirmed to model specific aspects of insect behavior. For instance, Buhl et al. show that self-propelled aprticles can replicate transitions from individual to collective behaviors in locusts [BSC+06].

Much work has also been carried out from a continuum point of view. As continuum techniques found in graphics derive from this work, much of what can be said for them also holds true here, with the exception that many more aspects have been studied in biology/physics/mathematics. For instance, Mogilner and Edelstein-Keshet [MEK99] have studied how modifying advection-diffusion equations can result in swarms with sharp edges and a constant internal density. Similarly, Topaz and colleagues [TBL06] investigate the formation of population "clumps", sharpness of their edges, evolution of the internal density under increasing population, etc.

1.2.3 Discussion

In addition to these efforts, other works have then focused on many different aspects of insect simulation. For instance, while many of the studied approaches are deterministic, studies also suggest a need for randomness. Yates and colleagues [YEE+09] show that in groups of locusts with low alignment, more random individual behaviors allow the group "to find (and remain in) a highly aligned state more easily". Additionally, Brecht et al. [BKBS13] show how randomly selecting only a subset of neighbors in individual-based algorithms can be sufficient, as opposed to every agent interacting with every other agent.

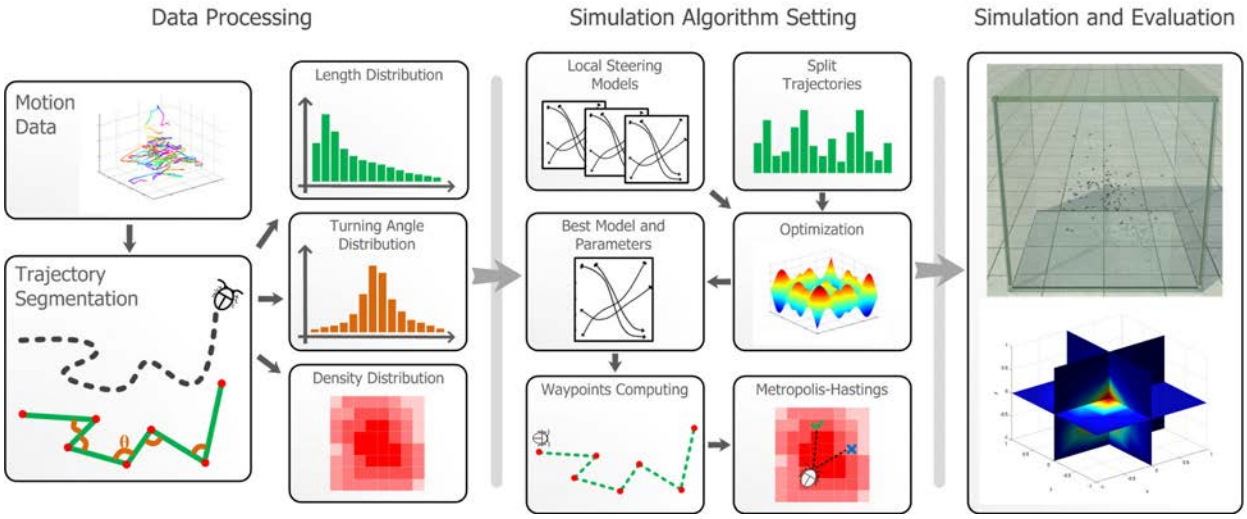


Figure 5.3 – The schematic view of our simulation pipeline.

Other investigated aspects include mating strategies [BMD+13] or phase transitions and kinematic fluctuations [VCBJ+95, ACC+13] for example.

On a general note, while many aspects of animal aggregation have been studied, these works very often focus on very specific aspects, and the effectiveness of applying these approaches on graphical applications is unclear and comparative evaluations of these approaches seem scarce. In addition, noted by Butail et al. [BMD+13], few models are validated against reference data giving that animals motion tracking task is non-trivial. In comparison to previous work, our system takes a data-driven approach (given the recent availability of insect data), independent of insect species, automatically conducting model evaluation and configuration, and generates realistic insect swarming behaviors on multiple scales.

1.3 Approach Overview

In this section, we give an overview of our framework. As part of the off-line process, our pipeline processes the insect motion data into a collection of linear trajectory segments and a density field. We then use this information to (1) perform automatic selection of local steering algorithms, (2) construct probability distribution functions used in the generation of intermediate waypoints for insect travel, and (3) capture the spatial structure of swarms. We describe how we combine these components into the final simulation system. The process is illustrated in Figure 5.3.

1.3.1 Pre-Processing Stage: Data

Insects' trajectories have been observed to be zig-zags [Oku86]. Inspired by this observation, we divide the trajectories into successive linear segments using the piecewise linear regression [NWK90] with a goodness of fit $R^2 = 0.9$. These segments will serve to set up the mesoscopic goal-selecting algorithm.

Formally, the data [KO13, PKO14] $\mathbf{z}$ can be defined as containing the agents' position

information at any timestep k. Thus, assuming m timesteps are recorded in the data:

$$\mathbf{z} = \bigcup_{k=1}^{m} \mathbf{z}_k. \tag{5.1}$$

For a given trajectory segment s of the ith agent, the data $\mathbf{z}_{s_i}$ is:

$$\mathbf{z}_{s_i} = \bigcup_{k \in s_i} \mathbf{z}_k, \tag{5.2}$$

where $s_i = [s_i^1, ..., s_i^n]$ groups all timesteps in the linear section, starting with $s_i^1 \geq 1$ and ending with $s_i^n \leq m$.

Piecewise-linear segments and turning angles (between consecutive segments) are automatically extracted to establish corresponding segment length and turning angle distributions. For the experiments, we used approximately one million video frames, containing more than 500 insects recorded at 100 FPS. Given about 100,000 extracted segments, the average segment length is 14.57 mm (median = 7.22, standard deviation = 19.01) and the average turning angles are around 2.34 degrees (median = 2.73, standard deviation = 40.02). We also discretized the simulation space into a number of cells and calculated the visiting frequency of all insects within each cell to construct a static density distribution.

While this work is inspired by zig-zag traveling fashion of common insects, as a merit of the piecewise linear regression, our approach does not require “zig-zag" trajectories. The data processing steps and simulation methods are general, independent of insect species, and applicable to smooth insect trajectories as well.

1.3.2 Runtime Stage: Three Levels of Simulation

Our system is composed of three levels:

- microscopic level: a local collision avoidance algorithm is selected and configured to compute each insect's successive positions,
- mesoscopic level: waypoints guiding the collision avoidance algorithm are constructed by sampling from the learned segment length and turning angle distributions,
- macroscopic level: the Metropolis-Hastings criterion is used to accept or reject (and thus re-sample) new mesoscopic waypoints.

We develop each of these levels in the next paragraphs, focusing on the application of the framework from Chapter 3 (microscopic level), and only briefly describing the rest, which can be found in [LWPCL15].

Microscopic Level At the lowest level of our simulation pipeline, lies the collision avoidance algorithm which ensures no insects would collide into each other. A collision avoidance algorithm can be defined as a function f which, given the current timestep k, the current positions of all agents $\mathbf{x}_k$, the agents' current velocities $\mathbf{v}_k$ as well as their goals $\mathbf{g}$, computes the agents' positions and velocities at the next timestep:

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ \mathbf{v}_{k+1} \end{bmatrix} = f\left(\mathbf{x}_k, \mathbf{v}_k, \mathbf{g}\right). \tag{5.3}$$

At this level, we use the framework described in Chapter 3 to automatically configure candidate local steering algorithms. This configuration aims to find the best performance for each steering algorithm to best approximate the actual trajectories conatined in the ground-truth data. After finding the optimal parameters for each steering algorithm, we can perform comparative evaluations and determine which model is the most suitable to simulate insect behaviors at the microscopic level. The major differences with Chapter 3 is that (1) the framework has been adapted to work with 3D data and collision-avoidance algorithms, and (2) the comparisons are performed on (a large collection of) linear segments of insects's trajectories rather than the complete trajectories. We here recall the framework's principle in the current context and present some related results.

A collision-avoidance algorithm (as defined in Equation 5.3) usually has several tunable parameters (e.g. agents' size, agents' preferred velocity...) which can influence the simulation results. With all agents' parameters $\mathbf{p}$, such an algorithm f (Equation 5.3) can be extended to the parameterized version as:

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ \mathbf{v}_{k+1} \end{bmatrix} = f\left(\mathbf{x}_k, \mathbf{v}_k, \mathbf{g}, \mathbf{p}\right). \tag{5.4}$$

Considering a parameterized collision avoidance algorithm, a simulation on a linear section $\mathbf{z}_s$ of the data at the mesoscopic level can be defined as follows:

$$f\left(\mathbf{z}_s, \mathbf{p}\right) = \bigcup_{k \in s} f\left(\mathbf{x}_k, \mathbf{v}_k, \mathbf{g}, \mathbf{p}\right), \tag{5.5}$$

initialized with $\mathbf{x}_{s_{start}} = \mathbf{z}_{s_{start}}$, $\mathbf{v}_{s_{start}} = speed\left(\mathbf{z}_{s_{start}}\right)$ and $\mathbf{g} = \mathbf{z}_{s_{end}}$.

Now, given a distance metric $dist\left(\right)$ between the data $\mathbf{z}_s$ and the simulation $f\left(\mathbf{z}_s, \mathbf{p}\right)$, we seek the parameter set $\mathbf{p}^{\mathbf{opt}}$ that minimizes this distance:

$$\mathbf{p}^{opt,f} = \underset{\mathbf{p}}{\operatorname{argmin}}\, dist\left(f\left(\mathbf{z}_s, \mathbf{p}\right), \mathbf{z}_s\right). \tag{5.6}$$

Once this parameter set is found, we can rank collision avoidance algorithms. For two such algorithms f_1 and f_2, we can say that f_1 is better than f_2 if and only if:

$$dist(f_1(\mathbf{z}_s, \mathbf{p}^{\mathbf{opt,f_1}}), \mathbf{z}_s) < dist(f_2(\mathbf{z}_s, \mathbf{p}^{\mathbf{opt,f_2}}), \mathbf{z}_s). \tag{5.7}$$

To perform what is described in Equations 5.6, 5.7, we rely on the framework's combinatorial algorithms including *Greedy*, *Genetic*, *Simulated Annealing*, and hybrid versions of these algorithms to handle high-dimensional tasks such as the multi-agent simulation.

Further, to evaluate insect movements and derive optimal parameters of a model that can provide the best approximation of a simulation to the reference data, we use the microscopic metric *Progressive Difference*. This metric measures the difference between the simulated and the reference trajectory between the starting and ending points of a segment while resetting the simulation to configurations of the reference data at each time step. The reason for us to pick this metric is that the data contains many frames representing long and split trajectories (not losing track of insects during the motion capture is very challenging). Thus, to learn the collision-avoidance behavior of the insects, it makes sense to focus on decisions being made at any given moment with limited influence from the past states of the swarm.

The specific steering algorithms we are evaluating are the Boids model [Rey87], Social Force model [HM95] and ORCA model [vdBSGM11]. In particular, for the Boids model, we only examine its collision-avoidance mechanism without other flocking rules. Since the distribution of evaluation scores appears to be non-Gaussian, we choose non-parametric within-subject tests to analyze differences between these models. The evaluation results shown in Figure 5.4 depict the median of scores across different steering algorithms before and after various optimization techniques have been applied. As all metrics are measuring the distance between simulated and reference trajectories, smaller scores indicate better algorithms. Note that each time, these scores are normalized by the worst algorithm's score (i.e. the optimization techniques did not degrade Boids and Social Force scores, as one could at first think by looking at the figure).

Using default parameters, a Friedman test revealed a significant effect of different steering algorithms on their scores ($p < 0.01$). A post-hoc test of pairwise comparisons using Wilcoxon signed rank tests with Bonferroni correction showed significant differences between all paired groups ($p < 0.01$). As results, Boids performed better than Social Force and the latter performed better than ORCA.

However, after applying the optimization techniques, across all evaluations, the Friedman test revealed a significance effect on different models and the post-hoc tests showed no significant difference between Boids and Social Force ($p > 0.01$), but significant difference between Boids and ORCA, and Social Force and ORCA ($p < 0.01$). These results indicate that, after optimization, the Boids and Social Force algorithms performed relatively similarly, while ORCA outperforms both.

Further, we measure the relationship between the number of timesteps that a segment contains and its corresponding evaluation score, by performing correlation and causality tests. To be more specific, pearman's rank test shows that these two variables are very weakly correlated (Spearman's $\rho \approx 0.27$, $p < 0.01$) and the constructed linear models show no significant effect of one variable on predicting the other (Intercept $p > 0.01$ and Slope $p > 0.01$). This indicates that the two variables are in general independent of each other. Thus we can use a specific steering model consistently rather than a mixture of steering algorithms switched at certain timesteps. The latter option may provide marginal improvement but at the cost of efficiency. Based on these results, we use ORCA for the local collision avoidance task with its statistically optimal parameters.

Mesoscopic Level Once we have chosen the best local steering algorithm, in order to actually generate trajectories for each insect agent, we need a goal destination. We define this destination as a series of waypoints. To achieve that, the segment length and turning

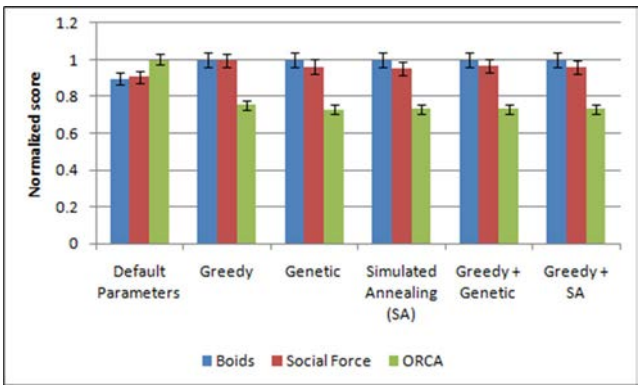


Figure 5.4 – Evaluation results of three steering algorithms under the Progressive Difference metric with various optimization techniques.

angle distributions constructed at the data pre-processing phase are sampled, and used to calculate a new waypoint every time an agent reaches its current one. Assuming at timestep k, an agent i, its position $x_{i,k}$, its previous (κ frames ago) goal $g_{i,k-\kappa}$, its current goal $g_{i,k}$, and new (to be computed goal) $g_{i,k+1}$. If that agent reaches its current goal at frame k, i.e. $\parallel g_{i,k} - x_{i,k} \parallel \leq \epsilon$, we have:

$$g_{i,k+1} = g_{i,k} + l_{i,k} T_{i,k} \frac{g_{i,k} - g_{i,k-\kappa}}{\parallel g_{i,k} - g_{i,k-\kappa} \parallel}, \tag{5.8}$$

where $l_{i,k}$ is the sampled length, and $T_{i,k}$ is a rotation matrix from the sampled turning angles. This statistical approach aims to replicate the noisy and chaotic aspects of insect local behaviors.

Macroscopic Level The sole computation of waypoints at the mesoscopic level would result in the agents' dissemination over the whole simulation space. To deal with this, we need a method at the macroscopic level to preserve the insects' spatial pattern and density distribution which are essential aspects of insect collective behaviors [CDF+03]. Thus, we use the Metropolis-Hastings criterion in order to either reject or accept newly computed waypoints, based on values stored in a three-dimensional grid, which can be either learned from the ground-truth data or input manually.

1.4 Results

In this section, we show our simulation results illustrated in several examples and provide both qualitative and quantitative evaluations.

1.4.1 Simulation

Our first example (Figure 5.5) shows the simulation of 100 insects moving in a confined space. This setting aims at mimicking the capturing environment of the collected data [PKO14]. In this example, we compare simulations of random sampling approach, meso-scale sampling approach, our combined meso- and macro-scales sampling approach, with the reference data. Using random sampling method, agents' trajectories tend to be

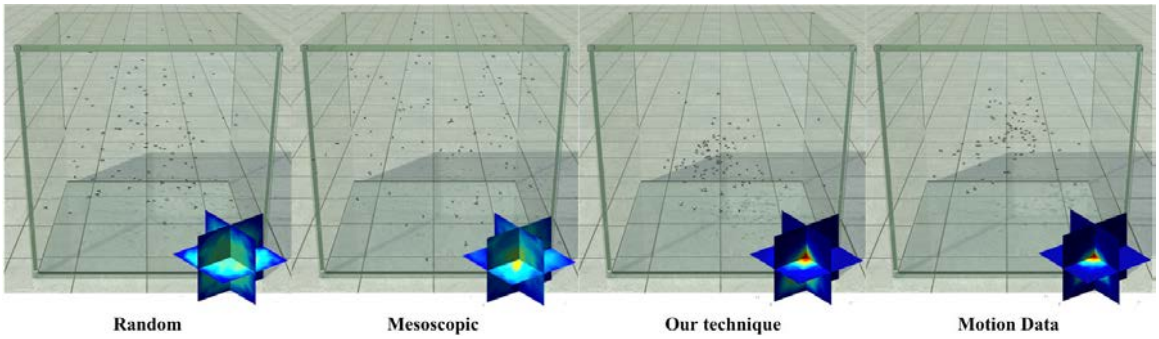


Figure 5.5 – Comparison of simulations using different sampling techniques with the reference data. Accumulated density distributions during the simulations are also shown.

Figure 5.6 – (Left) Simulation of 200 midges swarming around a lantern. (Right) Simulation of 50 butterflies moving on a prairie.

very smooth; after applying meso-scale sampling method, agents start to exhibit swerving and zig-zaging behaviors. However, since there exists no macroscopic-level guidance for governing the spatial pattern, agents would eventually spread out the simulation space. By adding the macroscopic level method, our technique finally closely approximates the reference data, preserving the swarming spatial pattern. The accumulative density information during entire simulations is also shown in Figure 5.5.

Our framework can also be adopted as an assistive animation tool to interpret sketch-like input and generate corresponding simulations. By combining features learned from the captured data (e.g. segment length and turning angle distributions) and user specifications including sketched trajectory and other settings (e.g. density distribution, steering models and parameters). Our framework is capable of creating insect simulations of the same or variable sizes – even with different species of insects.

To better demonstrate, we embedded the simulation in virtual environments[1]. The first example (Figure 5.6, left) shows an interpretation of a circle sketch (Figure 5.7, left): we obtain sampled trajectories – within the density field derived from the Gaussian distribution along the sketch – for multiple agents. In this particular scene, we simulate 200 insects orbiting around a lantern. Similarly, taking the sketch from the right side of Figure 5.7 as input, we generated a simulation with 50 butterflies moving on a prairie (Figure 5.6, right). Our technique can also be easily integrated with existing global path planning algorithms. An example shown in Figure 5.8 demonstrates our framework incorporated with a navigation algorithm similar to the one presented in [PvdBC+11].

[1]The virtual environments' creation and design are inspired by Andrew Price.

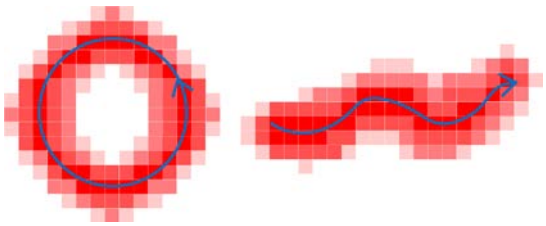


Figure 5.7 – Example sketches (3D) with density fields derived from the Gaussian distribution.

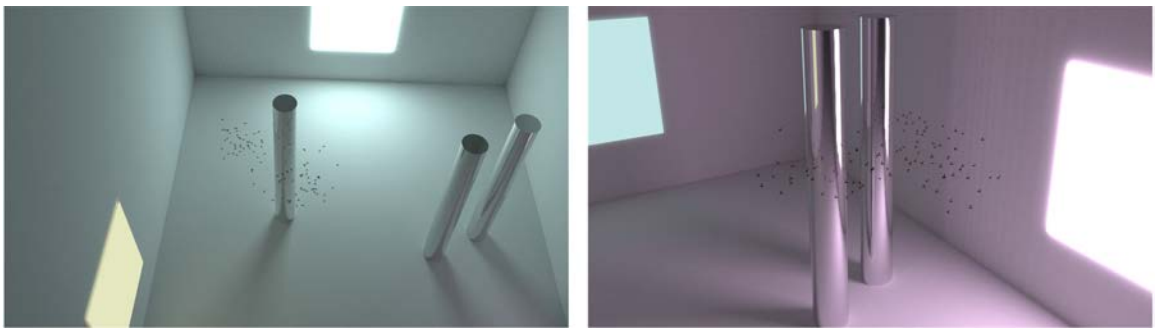


Figure 5.8 – A swarm of insects is traveling to successive light sources while avoiding obstacles.

1.4.2 Evaluation

To evaluate our technique, we conform the size of our simulation domain to the same scale as the data capture environment, and adopt both qualitative and quantitative measurements.

To start with, one of the most prominent features of swarming phenomena is the spatial pattern and corresponding density information. Comparison results can be seen in Figure 5.5. Between the random waypoint sampling and our mesoscopic waypoint sampling approaches, the latter tends to have small clusters (reflected on the small intensity sub-figures in the bottom-right corners) due to the fact that most processed segments have relatively small lengths compared to the capturing environment. Thus by sampling the segment length distribution, agents travel frequently in short distances. After combining with the macroscopic level sampling, the density distribution of our simulation closely approximates the data and demonstrates the spherical pattern as is the case in the data. In this case, the density decreases when the distance to the swarm's centroid increases resulting in an aggregation gclose to the centroid. This finding is consistent with previous literature in biology and also dispels the assumption that insects can be simulated in a purely random fashion which would result in roughly uniform distributions within any predefined simulation space.

While visual similarity is an intuitive way to evaluate the results, for large dynamic systems, we also need quantitative measurements. In particular, as we are using the Metropolis-Hastings criterion to govern the mesoscopic-level waypoint generation based on discrete density distributions constructed from the data, different discretizing scales of the simulation space could alter simulation qualities. In Figure 5.9, we show by altering cell numbers, the change of average distance of all agents to the swarm centroid during a period of time. Notably, adding mesoscopic level computation increases the

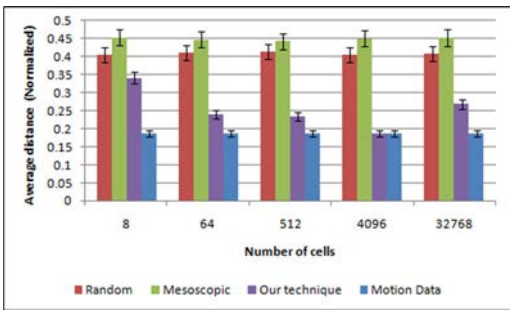


Figure 5.9 – Agents' average distances of different sampling techniques on various discretizing scales comparing to the motion data.

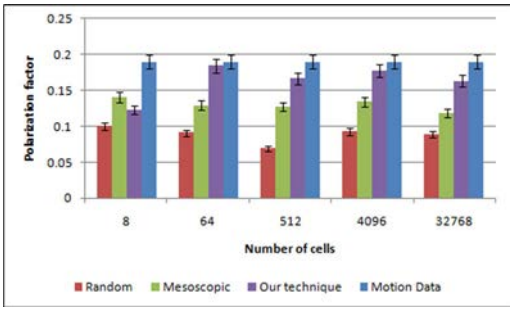


Figure 5.10 – Polarization scores of different sampling techniques on various discretizing scales comparing to the motion data.

average distance and the reason is probably that by traveling short distances and with random assigned initial locations, agents tend to stay close to their initial positions while under random assigned waypoints agents travel across the whole space more frequently resulting in shorter distances to the centroid during many timesteps.

Another notable feature of biological systems is the order in collective behaviors. While swarms of flies are rarely similar to other animal species (e.g. fish) that would polarize their state and make the group behave as a whole, they have been reported do show some features as collective behaviors [ACC+13]. To measure such an order, we use the polarization factor which is a common metric in studying the collective animal behavior [VCBJ+95]. The polarization Φ of a swarm is calculated as follows:

$$\Phi = \| \frac{1}{N} \sum_{i=1}^{N} \frac{\overrightarrow{v_i}}{\|v_i\|} \| \tag{5.9}$$

where $\overrightarrow{v_i}$ indicates the velocity of ith agent and N is the total number of agents. When Φ is zero, it means all individual velocities are canceling each other in terms of the direction. When Φ is close to one, it means velocities of all agents have nearly parallel directions. The values of polarization Φ at different discretizing scales under various sampling methods are shown in Figure 5.10. In general, by combining more waypoints sampling techniques we get Φ closer to the reference data.

It is worth noting that for both the average distance and polarization factor analyses, when the number of cells gets either too small or too large, the Metropolis-Hasting criterion appears to be less effective. This is due to the scale of the discretization which affects the visiting frequency of each cell and further influences the spread of the underlying density distribution. At the extremes, either the generated waypoints would be far

away from each other (effectively skipping cells), or many successive waypoints would be in the same cells (and not necessarily moving to neighbors). Either way, this would result in some cells being under-visited.

In terms of performance, the pre-processing time and memory complexities are $O(n)$ where n is the number of frames and the sampling methods take $O(1)$ time. On a typical laptop (Intel i5-3230@2.60GHz, 6GB RAM, Windows 8.1 64 bits), the simulation of 100 agents, using ORCA, runs at 4,900 FPS.

1.5 Conclusion

In this chapter, we show how our work on evaluation and parameter estimation can be applied in the context of insect simulation. This resulted in a multi-scale, data-driven system for the visual simulation of insect swarms. We simulate each insect as an individual agent; at each stage of the simulation, we select the appropriate techniques or parameters based on the analysis of observed/captured insect motion data. Like many agent-based methods, our method consists of a local avoidance module and a waypoint synthesis module to capture local interaction among insects and noisy individual trajectories that result in the coherent macroscopic motion patterns of a swarm.

We show how to automatically tune each module of our method based on observed insect motion data. In general, it is very difficult for an animator to control a swarm motion, as it is nearly impossible to specify motion trajectories for each individual insect, simulation is the most efficient way to create insect swarms. However, the parameter space is large and a desired result can be achieved only after a very time-consuming, manual trial-and-error process. Our system avoids this tedious tuning task and reproduces the visual animation of insect swarming motion with ease. Our method is also user-friendly, as it enables direct manipulation of large-scale insect swarming motion by allowing the animator to sketch out a desirable path and incorporate this user input into density gradient maps without changing other modules. In addition, as our approach runs at very high interactive rates, animators can apply iterative refinement on simulations until the visual quality of the animation results is satisfactory.

We tested, evaluated, and demonstrated our approach on several scenarios both qualitatively through visual inspection and quantitatively using scoring metrics. In particular, our quantitative comparisons illustrate the ability of our simulator to reproduce statistically significant measurements.

The main future direction for this work is the exploration of data from other insect species. Our current implementation assumes that local interactions among insects are largely driven by local collision avoidance and that each behaves according to the same statistical laws for simplicity and efficiency in simulation, as commonly done by many multi-agent simulations. In reality, there may be multiple types of insects interacting with each other and their swarming behaviors can vary. Clustering trajectories before processing them may introduce a new ability to consider mixed data from multiple species or more complex insect behaviors. Or, one can imagine different factors to consider and evaluate them for different types of insects (e.g. crawling vs. flying).

While we consider 3D motion data as a more general case for testing our algorithm, our method is perfectly able to handle 2D insect motion data. Adapting our data pro-

cessing pipeline to such a situation is straightforward. However, due to the lack of 2D insect motion data, we have not tested our method on insects moving on a 2D plane (crawling). In addition, we consider a static density field to guide the global, aggregate insect motion using the Metropolis-Hastings criterion. It would be interesting to consider dynamic fields and to automatically adapt these fields to significant changes in the surrounding environment. Finally, the promising results of this work suggest the possibility of developing more novel tools for animators to guide simulated swarm motion for artistic control.

2 Application to Pedestrian Tracking

Figure 5.11 – Our mixture motion model can accurately compute the trajectories in real time. We highlight different motion models (Boids, Social-Forces, or reciprocal velocity obstacles) used for the same pedestrian (marked in red) over different frames. We believe that it is not possible to model the trajectory of all pedestrians based on a single, uniform model. Instead, we adaptively choose the best-fit model for every pedestrian in the scene that can be adapted to the environment or the crowd conditions.

2.1 Introduction

This chapter describes a project on which I collaborated with Aniket Bera, a PhD student at the Gamma Group from the University of Carolina at Chapel Hill, NC, USA. It proposes another application to the evaluation and parameter estimation framework presented in Chapter 3.

The work presented in this chapter is geared towards pedestrian and crowd traffic management, which compared with car or vehicular traffic, does not yet benefit from a large set of technologies to perform traffic forecast, modeling, surveillance, disaster prevention, etc. Typically, a pedestrian traffic management system is made of three components: a tracker to estimate the current traffic or movement conditions, a simulator to predict short-term traffic evolutions, and active systems to adapt to the traffic forecast (e.g. traffic signs, automatic gateways, etc.). Some examples of recent computer assisted pedestrian traffic systems are described in [SF15, HHD14]. One of the major problems is the tracking of pedestrian and crowd motions and trajectories in indoor or outdoor scenes. This is a key problem for estimating traffic conditions, modeling the pedestrian flow as well as the design of architectural and urban structures (sidewalks, crossings...) [SBR14, ZMMS15]. Despite many recent advances in image processing, traffic management and computer vision, it is still difficult to accurately track pedestrians in real-world scenarios, especially as the crowd density increases [ABV14, BM09, SBR14, ZMMS15]. There are many challenges that arise, including intra-pedestrian occlusions (pedestrians blocking one another) that vary between frames, changes in lighting and pedestrian appearance (e.g. shadows or partial visibility), and the difficulty of modeling the pedestrian behavior and intent of each pedestrian. In this context, our objective is to improve the accuracy of tracking algorithms that can be widely used for traffic management.

In the context of pedestrian and crowd traffic management, online probabilistic trackers (Section 2.2.1) better answer the needs of crowd management, since they perform real-time tracking that captures complex pedestrian dynamics in heterogeneous crowds that arise frequently in traffic management systems. The accuracy of this category of trackers is improved by using realistic crowd motion models for computing motion priors (Section 2.2.2). Generally, a single, homogeneous motion model is used for the computation of this motion prior. As already mentioned in Chapters 2 and 3, every motion model relies upon one or more assumptions and has a limited validity range. It is far easier to describe a crowd's behavior at all times and under all circumstances with several, well tuned algorithms than with a single, generally-configured one. The configuration of these simulators is done through their underlying parameters, which may correspond to the size, speed, anticipation period, or local navigation constraints of each pedestrian. As the behavior of each pedestrian responds to changes in a dynamic environment (due to other pedestrians or obstacles such as vehicles), these model parameters should be recomputed or updated to improve the resulting motion model's accuracy. Overall, we need efficient techniques that can take into account heterogeneous behaviors based on constantly changing models and underlying parameters. We refer to this set of constantly tuned motion models as Mixture of Motion Models.

2.1.1 Working Arrangements

While Aniket focused on the tracker itself, interfacing it with the Mixture of Motion Models, and the generation of results with the complete system, I focused on the Mixture of Motion Models itself. As such, the "technical section" of this chapter also focuses on this same part, only briefly describing the rest of the approach.

The rest of the chapter is then organized as follows. In Section 2.2, we review existing work in the field of pedestrian tracking. The approach is described in Section 2.3, and results are presented in Section 2.4. Finally we conclude in Section 2.5.

2.2 Related Work

The problem of tracking objects and pedestrians has been studied in computer vision, image processing and broader traffic management, resulting in a variety of approaches. The following paragraphs aim to give a general idea of recent advances in multi-object/pedestrian tracking and to introduce approaches on which our work relies. For a more complete introduction to object detection (and tracking), we direct the reader to [YJS06] for a survey and to [EG09, WLY13] for benchmarks and comparisons.

2.2.1 General Object Tracking

Before even a single object can be tracked, one needs to be able to detect it in images/video sequences. A variety of approaches exist to solve this task, including point detectors which look for objects' points which have special properties, background subtraction where a representation of the scene is learned and objects are thus any deviations that appear, image segmentation where related regions are detected, and supervised

learning where a representation of the object is learned and subsequently detected in new frames.

These object detection techniques can then be leveraged to track objects as follows.

Appearance-Based Tracking Appearance-based algorithms, also known as "tracking-by-detection" or "tracking by repeated recognition", seek to track objects by recognizing these same objects in consecutive frames. In one such recent approach, Grabner and colleagues[GGB06] proposed a semi-supervised learning algorithm which assumes the tracked object has been detected/labeled in the first frame. They use Haar-like features to classify regions of frames as object or non-object (binary classifiers). They then use adaptive boosting (or *AdaBoost*) to combine these weak classifiers (typically classifiers which only need to perform a little better than random, i.e. 50% in the present case of binary classifiers) into a strong classifier being able to reliably detect the tracked object. These classifiers are further continuously updated at each frame: assuming the object has been detected in a frame, a sampling module will generate samples in the neighboring area and a labeler will assign labels to these samples. For instance, a sample on the object would be used as a positive sample, further-away samples with significant overlap with the object would be ignored, and still further samples with little to no overlap with the object would be used as negative samples. Finally, these labeled samples are used to update the classifiers.

In this and related approaches, the most common issue that emerges is the drift which results from sub-optimal update samples, where positive samples may not fully contain the target object for instance (online update noise). Subsequent works have then sought to solve this issue. One such approach is to use Multi-Instance Learning (MIL) where, instead of using labeled samples, the algorithm uses labeled bags of samples [BYB11, SHN12]. Bags are labeled positive if they contain at least one sample which would have been labeled positive, and negative otherwise. This way, the ambiguity is passed from the labeling module to the learning module, which has been shown to improve the tracking accuracy.

With the progress achieved with these approaches (and others further improving robustness to online update noise [HST11, ZZY12]), the tracking of objects has become very robust, even in real time. Despite these advances (applicable to multi-target tracking), pedestrian tracking in crowded environments remains challenging as the image-size of targets (pedestrians) can be very low making it difficult to reliably extract features from targets (and thus distinguish between them), which additionally further exacerbates problems such as object occlusion.

Data-Association In order to cope with these more challenging types of data, where tracking errors are common (e.g. inter-object occlusions often lead to mix-ups between target objects, i.e. ID switches), various data association techniques have been proposed (also known as "detect-then-track"). The purpose of these techniques is to combine target object detection responses into tracks, while correcting possible errors.

One possible approach to this association problem is to use flow networks [ZLN08, BC13]. Li and colleagues [ZLN08] build a flow network where, for each target object detection response i, they create two nodes u_i and v_i and arcs (u_i,v_i), (s,u_i), and (v_i,t), where s and t are the source and sink of the flow network. Additonally, a detected

possible transition between two responses i and j (corresponding to a possible object track) is modeled as a corresponding (v_i,u_j) arc. Then, by assigning cost and flow values to each arc, it is possible to run a min-cost flow algorithm, thereby computing the flows of the network corresponding to the tracked objects' trajectories.

Other approaches to the association problem include using the Hungarian algorithm [PSH+06], as well as linear programming [JFL07, BFF09]. While data-association methods much improve multi-object tracking, they are not suited for real-time tracking applications as they rely on a global optimization process which requires the data to have already been fully acquired.

Particle Filters Particle filters are a technique which is commonly used for tasks involving (noisy) measurements or observations of an otherwise unknown quantity (hidden markov model). Given such a quantity x and the corresponding observation y, at the current timestep k, by using Bayes' rule, one can express the probability of the quantity x_k conditioned by all the previous observations $y_{1:k}$:

$$p(x_k|y_{1:k}) = \frac{p(y_k|x_k)p(x_k|y_{1:k-1})}{p(y_k|y_{1:k-1})}. \tag{5.10}$$

Through marginalization, we have:

$$p(y_k|y_{1:k-1}) = \int p(y_k|x_k)p(x_k|y_{1:k-1})dx_k, \tag{5.11}$$

which is constant, resulting in this denominator being treated as a normalization constant (since $p(x_k|y_{1:k})$ is a probability density function and thus must integrate to 1). Additionally:

$$p(x_k|y_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|y_{1:k-1})dx_{k-1}. \tag{5.12}$$

As a result we can write the optimal Bayesian filter:

$$p(x_k|y_{1:k}) = \alpha p(y_k|x_k) \int p(x_k|x_{k-1})p(x_{k-1}|y_{1:k-1})dx_{k-1}. \tag{5.13}$$

Then, a particle filter is this same equation with a Monte-Carlo approximation of the integral which does not have a closed-form solution. Assuming a set of m particles $\{x^{(i)}\} i \in \{1..m\}$ and corresponding weights $\{\pi^{(i)}\} i \in \{1..m\}$, the particle filter is written:

$$p(x_k|y_{1:k}) \approx \alpha p(y_k|x_k) \sum_{i \in \{1..m\}} \pi_{k-1}^{(i)} p(x_k|x_{k-1}^{(i)}). \tag{5.14}$$

This technique is well suited for (and commonly used in) tracking [KBD04, OTDF+04, BRL+09, HF09, BCM12, LCLM14, BM14], as in this last formulation, the probability $p(x_k|y_{1:k})$ of the object being at x_k given all past observations $y_{1:k}$ is computed with: a normalization constant α, the probability $p(y_k|x_k)$ of observing the object at x_k (confidence of the object detector) and the transition probability $p(x_k|x_{k-1}^{(i)})$ (based on an internal model of the obejct's motion) of particle i (which is a possible state of the object, with probability/weight $\pi_{k-1}^{(i)}$). This technique makes choices on current frames given only the previous ones, thus being applicable to real-time tracking.

2.2.2 Crowd Motion Priors

In addition to the previously discussed multi-object (general) tracking techniques, some approaches have been designed specifically to track crowds by using motion priors in order to have an idea where to look for the pedestrians in the following frames.

For instance, Ali and Shah [AS08] use a cellular automaton, where the image space is discretized into cells. Each individual is represented by a template pixels around a centroid particle. When tracking such an individual, the algorithm computes the probabilities of this centroid particle to transition to neighboring cells. These probabilities are based on the match of the template at the new cell, as well as three floor fields. These floor fields here capture the motion prior; the Static Floor Field represents constant properties of the scene such as preferred paths or exits, the Boundary Flow Field models the presence of obstacles, and the Dynamical Floor Field represents the current state of the crowd's motion. In this manner, the motion prior is tightly integrated in the tracking process. Similarly, Rodriguez and colleagues [RS11] track pedestrians by matching video features to previously learned motion patterns from a database, and use these motion priors in a Kalman Filter-based tracker. Kratz and Nishino [KN12] use motion priors in the form of a set of hidden Markov models trained on regions of the video (cuboids) which contain direction statistics of pedestrians at those locations. Other approaches infer individual pedestrians' motions from priors in theform of motion patterns [YN12, ZGM12].

Other approaches compute these motion priors from crowd simulation algorithms. Antonini and colleagues [AMBT06] estimate for each pedestrian the cost of doing specific navigation choices. They discretize a pedestrian's velocity space, and compute for each cell the cost or"utility" of that cell based on a number of factors, such as other pedestrians' presence in the direction defined by the cell, the pedestrian's destination, the walking directions of other pedestrians, and the cell's implied velocity adaptation. The resulting motion prior is then incorporated into a Bayesian filter. Pellegrini et al. [PESVG09] compute a confidence for patches of pixels based on a normalized cross correlation similarity metric and the predicted position of the pedestrians computed with a simulation algorithm (already mentioned in Chapter 2, Section 2.2.1), and Yamaguchi and colleagues propose a similar approach based on their energy-based simulation algorithm [YBOB11].

Finally, it should be noted that the afore-mentioned particle filters include a transition probability term $p(x_k|x_{k-1}^{(i)})$, meaning that they wourld greatly benefit from more acurate motion priors. Thus, some works [JB12, LCLM14, BM14] have integrated particle filter tracking with a crowd simulation algorithm (introduced in [vdBLM08], see Chapter 2, Section 2.2.2).

2.2.3 Summary

Despite the recent progress on appearance-based algorithms, they still struggle on crowded scenes. Data-association algorithms are able to solve some these issues but they are ill-suited to real-time applications as they often require all of the video data to be available for global optimization. Particle filters on the other hand, make decisions on only past frames of the video data, and have been successfully employed in rela-time applications. Tracking techniques relying on motion priors also allow to improve tracking results by

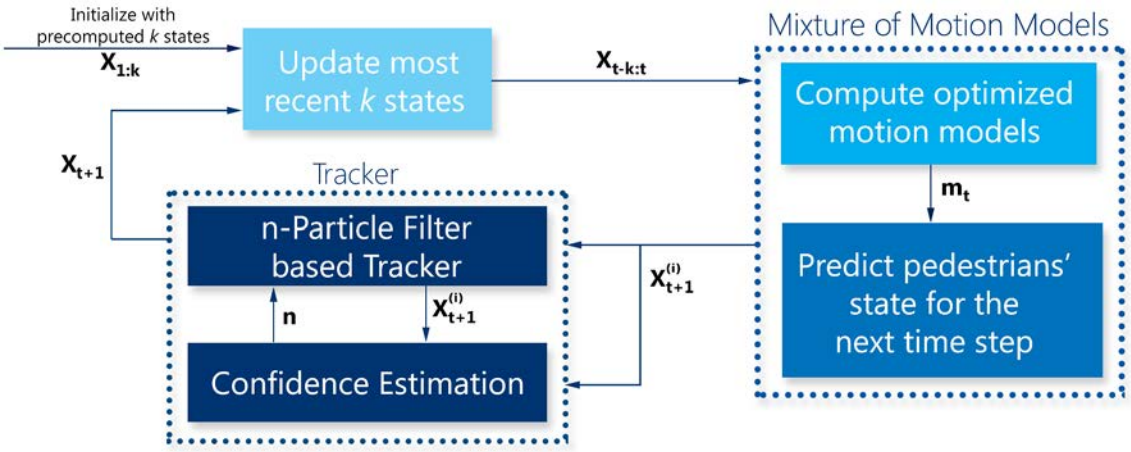


Figure 5.12 – Overview of our real time tracking algorithm. The symbols used in this figure are explained in Section 2.3.1. We use the trajectory computed over prior k frames, expressed as a succession of states, to compute the new motion model; we use our mixture motion model to compute the next states using a particle filter.

approximating the underlying behaviors of the tracked pedestrians. Finally, crowd simulation algorithms have successfully been coupled with tracking algortihms as motion priors, and in particluar, with particle filters. However, as crowd simulation algorithms have their strengths and weaknesses in simulating various types of crowds and in various circumstances, by matching them closer to the observed trajectories, one should be able to further improve tracking results.

2.3 Mixture Motion Model

In this section, we introduce the notion of a parameterized motion model. We then describe the different parameterized motion models that form the basis of the mixture motion model. Finally, we describe the mixture motion model itself.

2.3.1 Overview and Notations

We present a method that uses particle filters to perform real time pedestrian tracking in moderately crowded scenes. Our approach can be viewed as a feedback pipeline (Figure 5.12), where we use (1) the most recent estimated agent states to (2) compute the best-fit motion model for each pedestrian, which is then used as motion prior for (3) a particle-filter based tracker:

Data Representation Our algorithm keeps track of the *state* (i.e. position and velocity) of each pedestrian for the last k timesteps or frames. These are referred to as the k-states of each pedestrian. These k-states are initialized by pre-computing the states from the first k timesteps. The k-states are updated at each timestep by removing the agents' state from the oldest frame and adding the latest tracker-estimated state.

The mixture motion model is a combination of several independent motion models. This mixture motion model is used to compute the best motion model for the agents during each frame. First, based on an optimization algorithm, we "configure" the motion

models to "best" match the recent k-states data and select the best model based on a specific metric. Second, we use the "best configured" motion model to make a prediction on the agents' next state.
The tracker is a particle-filter based tracker that uses the motion prior, obtained from the Mixture of Motion Models, to estimate the agents' next state. This tracker further uses a confidence estimation stage to dynamically compute the number of particles that balance the trade-offs between the computation cost and accuracy.

We use the following notations in the remainder of the text:

- x represents the "real" state (position and velocity) of an arbitrary pedestrian,
- y represents an observation of the "real" state (position and velocity) of an arbitrary pedestrian,
- m represents the "best configured" motion model from the Mixture of Motion Models $\{f1, f2, ...\}$,
- bold fonts are used to represent values for all the pedestrians in the crowd; for example $\mathbf{y}$ represents the states (positions and velocities) of all pedestrians as computed by the tracker,
- subscripts are used to indicate time; for example m_t represents the "best configured" motion model at timestep t, and $\mathbf{x}_{t-k:t}$ represents all states of all agents for all successive timesteps between $t-k$ and t, as computed by the tracker.

The "best configured" motion model can then be used as follows: $x_{t+1} = m_t(x_t)$ or $\mathbf{x}_{t+1} = m_t(\mathbf{x}_t)$ to compute the motion of one arbitrary pedestrian or all pedestrians, respectively.

2.3.2 Particle Filter for Tracking

Though any online tracker that requires a motion prior system can be used, we use particle filters as the underlying tracker algorithm. As already defined in Section 2.2.1, particle filtering is an approach that solves Markovian estimation problems. We recall the expression of a particle filter for an arbitrary pedestrian:

$$p(x_t|y_{t-k:t}) \approx \alpha p(y_t|x_t) \sum_{i=1}^{n} \pi_{t-1}^{(i)} p(x_t|x_{t-1}^{(i)}). \tag{5.15}$$

In our formulation, we compute the dynamic transition $p(x_t|x_{t-1}^{(i)}) = \mathcal{N}(m_t(x_{t-1}^{(i)}), \Gamma)$ thanks to the "best configured" motion model m_t at timestep t ($\mathcal{N}$ here denotes the bidimensional Gaussian probability distribution, and Γ is a diagonal covariance matrix of noise).

The particle filter which we use further adapts the number of particles for each agent in order to make a compromise between accuracy and computation cost, more can be found on this topic in [BM14].

2.3.3 Parameterized Motion Model

A motion model is defined as an algorithm f, which from a collection of agent states $\mathbf{x}_t$, derives new states $\mathbf{x}_{t+1}$ for these agents, representing their motion over a timestep towards the agents' immediate goals $\mathbf{g}$:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{g}). \tag{5.16}$$

Motion algorithms usually have several parameters that can be tuned in order to change the agents' behaviors. We assume that each parameter can have a different value for each pedestrian. By changing the value of these parameters, we get some variation in the resulting trajectory prediction algorithm. We use $\mathbf{p}$ to denote all the parameters of all the pedestrians. Typically, for a crowd of 50 pedestrians, the dimension of $\mathbf{p}$ could be anywhere in the range of 150-300 depending on the motion model. In our formulation, we denote the resulting parameterized motion model as:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{g}, \mathbf{p}). \tag{5.17}$$

2.3.4 Mixture of Motion Models

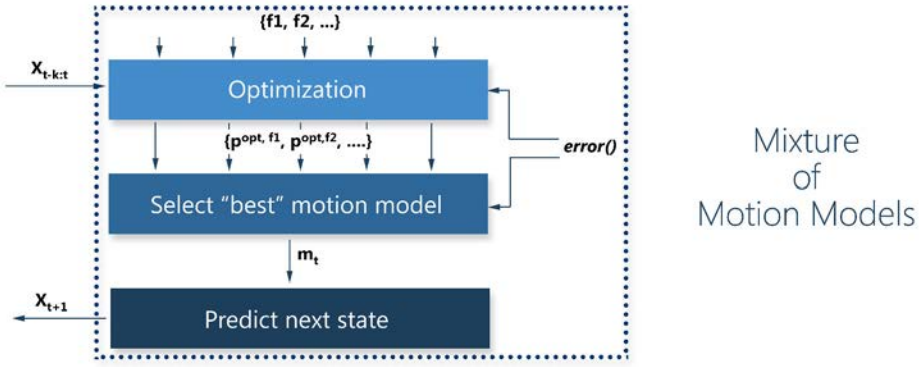


Figure 5.13 – Our parameter optimization algorithm used in Figure 5.12. Based on the error metric, we compute optimal parameters for each motion model. The best motion model (from RVO2, Social Forces or Boids) is used for trajectory extraction and to predict the next state.

We now present the algorithm to compute the mixture motion model (Figure 5.13), which essentially corresponds to computing the "best" motion model at any given timestep. In this case, the "best" motion model is the one that most accurately matches agents' immediately past states, as per a given error metric. This "best" motion model is determined by an optimization framework, which automatically finds the parameters that minimize the error metric. In Chapter 3 we designed an optimization framework for evaluating crowd motion models, which computes optimal parameters of simulation algorithms in offline situations. Here we use an adapted, online version which iteratively computes the best heterogeneous motion every few frames and chooses the most optimized crowd parameters at a given time. The computation cost is considerably lower and hence useable for real-time tracking.

Formalization Formally, at any timestep t, we define the agents' k-states (as computed by the tracker) $\mathbf{x}_{t-k:t}$:

$$\mathbf{x}_{t-k:t} = \bigcup_{i=t-k+1}^{t} \mathbf{x}_i. \tag{5.18}$$

Similarly, a motion model's corresponding computed agents' states $f(\mathbf{x}_{t-k:t}, \mathbf{p})$ can be defined as:

$$f(\mathbf{x}_{t-k:t}, \mathbf{p}) = \bigcup_{i=t-k+1}^{t} f(\mathbf{x'}_i, \mathbf{g}, \mathbf{p}), \text{ with } \mathbf{x'}_{t+1} = f(\mathbf{x'}_t, \mathbf{g}, \mathbf{p}) \tag{5.19}$$

initialized with $\mathbf{x'}_{t-k+1} = \mathbf{x}_{t-k+1}$ and $\mathbf{g} = \mathbf{x}_t$.

At timestep t, considering the agents' k-states $\mathbf{x}_{t-k:t}$, computed states $f(\mathbf{x}_{t-k:t}, \mathbf{p})$, and a user-defined error metric $error()$, our algorithm computes:

$$\mathbf{p}_t^{opt,f} = \underset{\mathbf{p}}{\operatorname{argmin}}\, error(f(\mathbf{x}_{t-k:t}, \mathbf{p}), \mathbf{x}_{t-k:t}), \tag{5.20}$$

where $\mathbf{p}_t^{opt,f}$ is the parameter set which, at timestep t, leads to the closest match between the states computed by the motion algorithm f and the agents' k-states.

For several motion algorithms $\{f1, f2, ...\}$, we can then compute the algorithm which best matches the agents' k-states $\mathbf{x}_{t-k:t}$ at timestep t:

$$m_t = f_t^{opt} = \underset{f}{\operatorname{argmin}}\, error(f(\mathbf{x}_{t-k:t}, \mathbf{p}_t^{opt,f}), \mathbf{x}_{t-k:t}), \tag{5.21}$$

and consequently, the best (as per the error in the $error()$ metric itself) prediction for the agents' next state obtainable from the motion algorithms for timestep $t+1$ is:

$$\mathbf{x}_{t+1} = m_t(\mathbf{x}_t). \tag{5.22}$$

Optimization Algorithm and Error Metric The optimization of 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 a hundred-dimensional optimization problem.

In total, we tested three global optimization approaches which we described in Chapter 3, Section 3.3 and Appendix A, Section 2:*Greedy Algorithm*, *Simulated Annealing*, and *Genetic Algorithm*.

We have tested these algorithms both in terms of how well they minimize the error metric and in terms of how fast they converge. Figure 5.14 shows the scores after optimization with all three methods, for the Boids, Social-forces and RVO2 motion models (separately), as well as the Mixture of Motion Models. As can be observed, the genetic algorithm leads to the lowest scores, followed by the simulated annealing and greedy algorithms. Note that as expected, the Mixture of Motion Models gives the lowest error scores as it consistently selects the best motion model for each situation. Similarly, Figure 5.15 shows the convergence times for each optimization method and motion model. The greedy algorithm is here the fastest, followed by the genetic and simulated annealing

algorithms. Note that the Mixture of Motion Models is only marginally slower than the other methods as the optimizations for the models constituent of the Mixture are performed in parallel. From these comparisons, we chose the genetic algorithm to generate all further results.

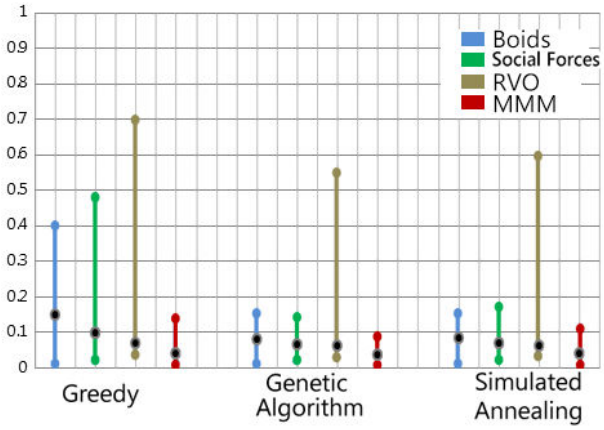


Figure 5.14 – Comparing the score of the different optimization approaches. Each graph is a range of the scores (minimum and maximum) and the black dot is the mean score. We compute the score from the normalized error metric. A lower value indicates better optimization. MMM or the 'Motion-Model Mixture' is our approach.

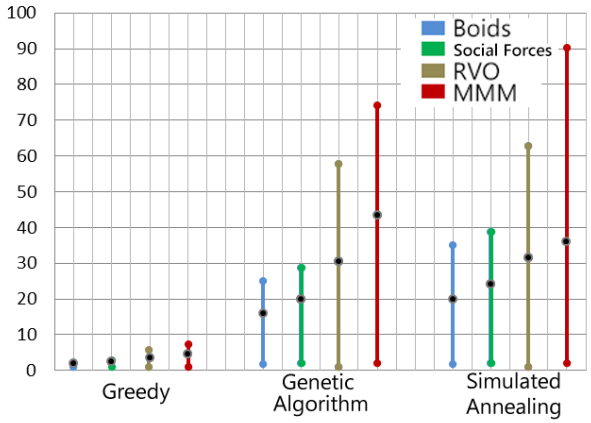


Figure 5.15 – This graph shows the time taken for each computation of every set of optimal parameters corresponding to each motion model. MMM is our approach. Time computed is in milliseconds. Each graph is a range of the scores (minimum and maximum) and the black dot is the mean score. We compute the score from the normalized error metric.

An error metric is also needed to compute the term in Equation (5.20). In our case, we have chosen a metric that simply computes the average 2-norm between the observed agent positions and the tracker-computed positions. Formally, this metric is defined at timestep t as follows:

$$error(f(\mathbf{x}_{t-k:t}, \mathbf{p}), \mathbf{x}_{t-k:t}) = \sum_{i=t-k+1}^{t} \|\mathbf{x'}_i - \mathbf{x}_i\|, \tag{5.23}$$

where the $\mathbf{x'}_i$ are from $f(\mathbf{x}_{t-k:t}, \mathbf{p})$ as per Equation (5.19).

2.4 Implementation and Results

In this section we present our implementation details and highlight the performance on 14 different crowd video datasets.

2.4.1 Motion Models

Our Mixture Motion Model can include any generic motion model that conforms to Equation (5.17). Here we describe the three component motion models that currently make up the Mixture Motion Model in our current implementation (and give their initial parameters in Table 5.1). We selected 3 motion models: the Reciprocal Velocity Obstacle model (RVO2), the Boids model, and the Social-Forces model. As detailed below, they cover complementary ranges of crowd densities. The Social-Forces model simulates local interactions between pedestrians as sets of repulsive forces. This is representative of what happens in dense situations, where people may enter in contact with one another. In lower densities, but still with highly cohesive motions, the Boids model simulates well how each pedestrian aligns his own motion with his or her nearest neighbors. Finally, in less dense scenarios, trajectories are individualized and anticipation plays a great role in interactions: RVO2 is the only one of the three techniques capable of simulating such behaviors.

Model / Parameters	**min**	**max**	**mean**
Boids model			
radius (m)	0.1	1	0.3
comfort speed (m/s)	1	2	1.5
Social-Forces model			
radius (m)	0.1	1	0.3
comfort speed (m/s)	1	2	1.5
RVO2 model			
comfort speed (m/s)	1	2	1.5
neighbor distance (m)	2	20	11
radius (m)	0.2	0.8	0.5
agent time horizon (s)	0.1	5	2
obstacle time horizon (s)	0.1	5	2

Table 5.1 – Initial motion model parameters for optimization.

Reciprocal Velocity Obstacles RVO2 is a local collision-avoidance and navigation algorithm. Given each agent's state at a certain timestep, it computes a collision-free state for the next timestep [vdBLM08]. Each agent is represented as a 2D circle on the plane, and the parameters (used for optimization) for each agent consist of the representative circle's radius, maximum speed, neighbor distance, and time horizon (only future collisions within this time horizon are considered for local interactions).

Let V_{pref} be the preferred velocity for a pedestrian that is based on the immediate goal location. The RVO2 formulation takes into account the position and velocity of

each neighboring pedestrian to compute the new velocity. The velocities of the neighbors are used to formulate the ORCA constraints for local collision avoidance [vdBLM08]. The computation of the new velocity is expressed as an optimization problem for each pedestrian. If an agent's preferred velocity is forbidden by the ORCA constraints, that agent chooses the closest velocity that lies in the feasible region:

$$V_{RVO} = \underset{V \notin ORCA}{\arg\max} \|V - V_{pref}\|. \tag{5.24}$$

More details and mathematical formulations of the ORCA constraints are given in [vdBLM08]. As per Equation (5.17), f returns the states obtained with the admissible velocity that is closest to the preferred velocity. The parameters are radius (size of 2D circle agents), comfort speed (i.e., speed when no interactions occur), neighbor distance (how close should neighbor agents be in order to be considered for collision avoidance), agent time horizon (agent collisions further in time than this horizon will not be taken into account) obstacle time horizon (same but for wall-like obstacles).

The Boids Model Initially developed to simulate the flocking behavior of birds, the use of this model has been extended to pedestrian motion in a crowd. Broadly, three rules are enforced on Boids agents:

- **Separation**: steer to avoid crowding local agents.
- **Alignment**: steer towards the average heading of local agents
- **Cohesion**: steer to move toward the average position (center of mass) of local agents

Thus, as per Equation (5.17), f is a function of agents' positions at some specified future time (current time plus constant). When the predicted distance between the pedestrians gets too low, a separation force is computed and added to the attraction force that is pulling the agents toward their goal. The parameters are radius and comfort speed.

Social-Forces Model The social forces model is defined by the combination of three different forces:

- **Personal Motivation force** (F^M): he incentive to move at a certain preferred velocity in a certain direction.
- **Social-Forces** (F^S): the repulsive forces from other agents and obstacles.
- **Physical Constraints** (F^P): the hard constraints other than the environment and other agents.

The net force $F^C = F^M + F^S + F^P$ then defines an agent's chosen new velocity. For a detailed explanation of the method, refer to [HM95].

As per Equation (5.17), f is a function of the agents' positions from which all computed forces are derived. The parameters are radius and comfort speed.

2.4.2 Evaluation

We use 14 different datasets accross three levels of density (low, medium, high) as summarized in Table 5.2. Some of these are challenging datasets [BM14] which are available publicly, as well as some standard datasets from the pedestrian tracking community [PEVG10].

Dataset	Challenges	Density	Agents
NDLS-1	BV, PO, IC	High	131
IITF-1	BV, PO, IC, CO	High	167
IITF-3	BV, PO, IC, CO	High	189
IITF-5	BV, PO, IC, CO	High	71
NPLC-1	BV, PO, IC	Medium	79
NPLC-3	BV, PO, IC, CO	Medium	144
IITF-2	BV, PO, IC, CO	Medium	68
IITF-4	BV, PO, IC, CO	Medium	116
NDLS-2	BV, PO, IC, CO	Low	72
NPLC-2	BV, PO	Low	56
seq_hotel	IC, PO	Low	390
seq_eth	BV, IC, PO	Low	360
zara01	BV, IC, PO	Low	148
zara02	BV, IC, PO	Low	204

Table 5.2 – Crowd Scenes used as Benchmarks. We highlight many attributes of crowd videos including density and the number of pedestrians tracked. We use the following abbreviations about the underlying scene: Background Variations(BV), Partial Occlusion(PO), Complete Occlusion(CO), and Illumination Changes(IC).

Successful Tracks and ID Switches We highlight the performance of our Mixture of Motion Models algorithm on these different benchmarks (partially illustrated in Figure 5.16), comparing the performance of our algorithm with single, homogeneous motion model methods: constant velocity model (LIN), LTA [PESVG09], Social-Forces [YBOB11], Boids [Rey99], and RVO2 [vdBLM08]. LIN models the velocities of pedestrians as constant, and is the underlying motion model frequently used in the standard particle filter. The other four models compute the pedestrian states based on optimizing functions, which model collision avoidance, destinations of pedestrians, and the desired speed. In our implementation, we replace the state transition process of a standard particle-filtering algorithm with different motion models. We also compare our performance to a baseline mean-shift tracker.

We show the number of successfully tracked pedestrians and the number of ID switches. A track is counted as "successful" when the estimated mean error between the tracking result and the ground-truth value is less than 0.8 meter in groundspace. The average human stride length is about 0.8 meter and we consider the tracking to be incorrect if the mean error is more than this value. As can be seen in Tables 5.3 and 5.4, our method consistently outperforms other approaches with more successful tracks (ST) and fewer ID switches (IS).

	High Density								Medium Density								Low Density			
	NDLS-1		**IITF-1**		**IITF-3**		**IITF-5**		**NPLC-1**		**NPLC-3**		**IITF-2**		**IITF-4**		**NDLS-2**		**NPLC-2**	
	ST	IS	ST	IS	ST	IS	ST	IS	ST	IS	ST	IS	ST	IS	ST	IS	ST	IS	ST	IS
LIN	53	17	63	27	51	35	59	18	67	15	60	29	36	22	52	36	68	23	69	21
Boids	58	15	66	23	56	33	65	14	73	13	65	26	40	19	52	35	70	22	72	19
Social Forces	56	16	66	26	52	33	62	15	74	11	68	23	41	19	59	31	75	18	72	14
LTA	54	17	65	22	51	32	60	17	68	11	62	28	42	18	54	32	69	23	70	20
RVO2	57	14	69	20	53	29	64	13	71	10	64	26	42	18	53	32	72	20	74	16
MeanShift	27	32	31	38	23	52	34	29	39	36	41	31	22	33	39	45	31	28	45	28
MMM	63	12	73	19	57	27	67	10	77	7	71	20	44	16	63	28	79	17	78	14

Table 5.3 – We compare the percentage of successful tracks (ST) and ID switches (IS) of our Mixture Motion Model algorithm (MMM) with homogeneous motion models - LIN, Boids, Social Force, LTA, RVO2, and a baseline mean-shift tracker. Our method provides higher accuracy compared to homogeneous motion models and fewer ID switches. The benefits of our approach are higher, as the crowd density increases. These datasets are publicly available at http://gamma.cs.unc.edu/RCrowdT/.

	seq_hotel		**seq_eth**		**zara01**		**zara02**	
	ST	IS	ST	IS	ST	IS	ST	IS
LIN	182	92	187	58	51	27	49	27
Boids	192	78	202	59	52	27	54	26
Social Forces	221	73	232	48	54	26	55	25
LTA	238	70	249	42	60	24	62	25
RVO2	241	71	258	37	61	22	65	23
MeanShift	98	171	112	139	32	41	33	39
MMM	252	68	267	34	63	20	68	21

Table 5.4 – We compare the percentage of successful tracks (ST) and ID switches (IS) of our Mixture Motion Model algorithm (MMM) with homogeneous motion models - LIN, Boids, Social Forces, LTA, RVO2 and a baseline mean-shift tracker with standard datasets - seq_hotel , seq_eth , zara01 , zara02 [PEVG10]. Again, our method provides higher accuracy compared to homogeneous motion models and fewer ID switches.

CLEAR MOT metrics We also use the **CLEAR MOT** [KR08] evaluation metrics to analyze the performance, specifically, the **MOTP** and **MOTA** metrics. **MOTP** evaluates the alignment of tracks with the ground truth while **MOTA** produces a score based on the amount of false positives, missed detections, and identity switches. These metrics have become standard for evaluation of detection and tracking algorithms in the computer vision community, and we refer the interested reader to [KR08] for a more detailed explanation.

We use these metrics across the density groups and the different motion models. As shown in Table 5.5, the scores obtained with our Mixture of Motion Models are consistantly better (higher).

	LIN			Boids			Social-Forces			RVO2			**MMM**		
	LD	MD	HD	LD	MD	HD	LD	MD	HD	LD	MD	HD	**LD**	**MD**	**HD**
MOTP	64.42%	52.82%	40.31%	67.24%	57.10%	43.14%	70.52%	61.33%	49.88%	72.19%	63.17%	51.31%	**73.98%**	**69.23%**	**54.29%**
MOTA	49.42%	35.3%	31.37%	50.59%	26.42%	30.88%	53.28%	44.19%	33.51%	53.95%	48.81%	35.83%	**54.18%**	**50.16%**	**38.83%**

Table 5.5 – We compare the MOTA and MOTP values across the density groups and the different motion models.

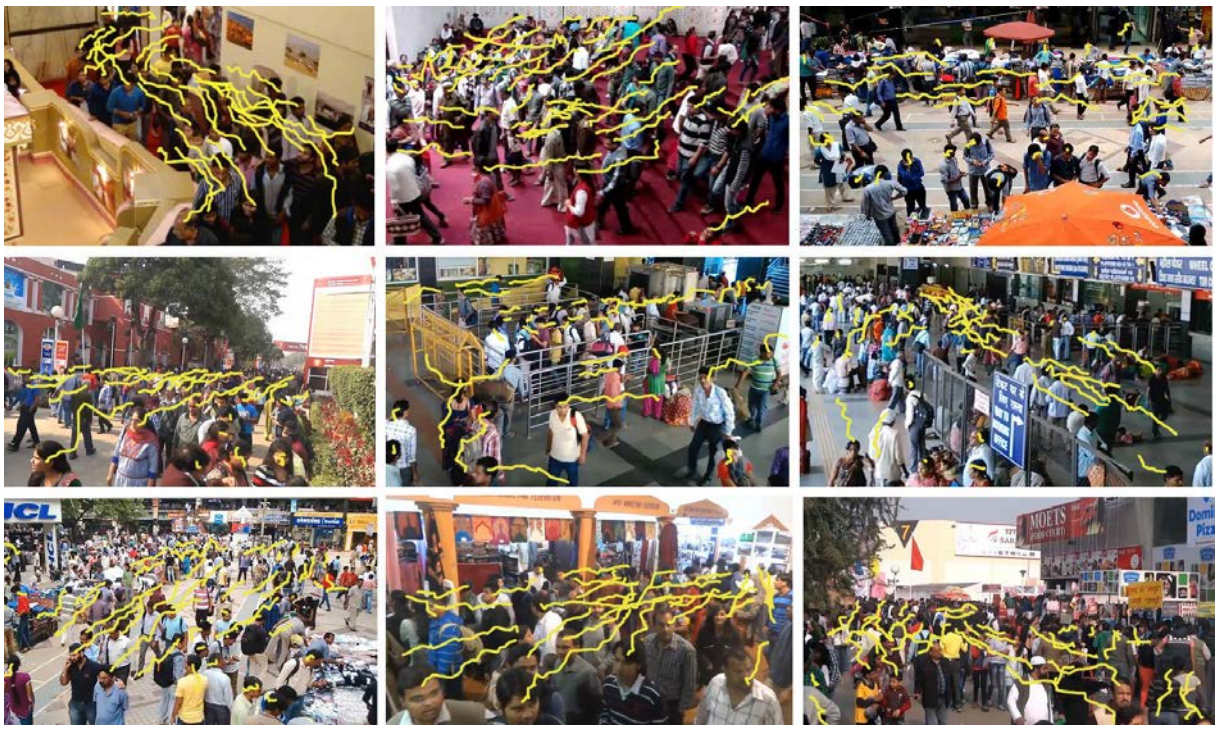


Figure 5.16 – The results of our approach on some challenging datasets. From top to bottom, left to right: IITF-1, IITF-2, NPLC-1, IITF-3, NDLS-2, NDLS-1, NLPC-2, IITF-4, IITF-5.

Independently of the overall tracking, we compare the position prediction ability of our Mixture of Motion Models. As Figure 5.17 shows, our approach is able to more accurately predict agents' future states (thus offering a better motion prior for the tracker); and the more so as the density increases.

Computation cost Finally, we evaluate the computational overhead of our optimization framework with the particle filter system in terms of computation time. As can be seen on Figure 5.18, the computational cost of the Mixture of Motion Models is small compared to the overall system.

2.5 Limitations, Conclusions, and Future Work

In this chapter, we show how our work on evaluation and parameter estimation can be applied in the context of pedestrian tracking. This resulted in a real time algorithm for pedestrian tracking in crowded scenes that are needed for next generation traffic management systems as well as the design of architectural models and urban environments. We highlight the performance of our approach on many pedestrian datasets, showing that it can track crowded scenes in real time on a PC with a multi-core CPU. Furthermore, we highlight the improved accuracy and the performance in complex benchmarks with low, medium, and high density crowds.

Just like the evaluation and parameter estimation framework on which this approach relies, it is simulator-agnostic, meaning that any simulation algorithm can be added to the Mixture. Our approach's main limitations are therefore related to our motion model

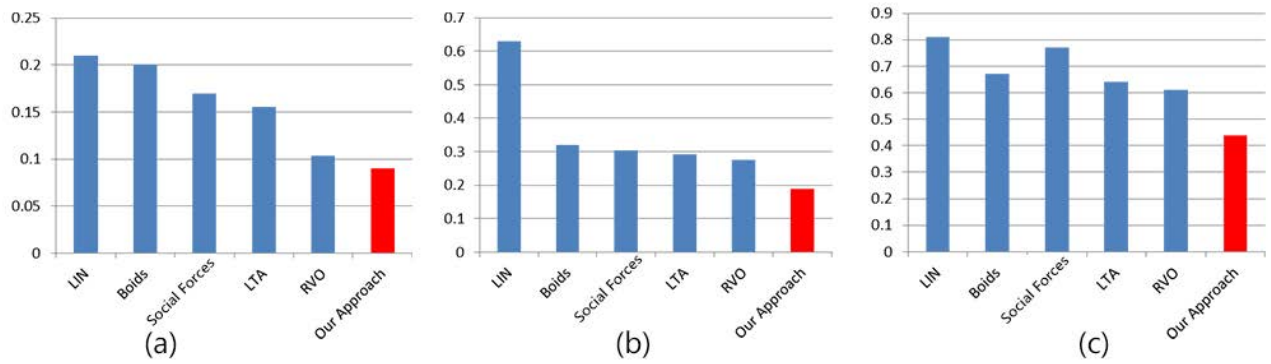


Figure 5.17 – This is the root-mean-square error in the predicted position compared to the ground truth. For an unbiased comparison, all measurements are in ground-space (meters). We have divided our dataset into 3 categories (refer to Table 2): (a) Low-density datasets (b) Medium-density datasets (c) High-density datasets. We find that our approach considerably lowers error for future-state prediction as the density increases.

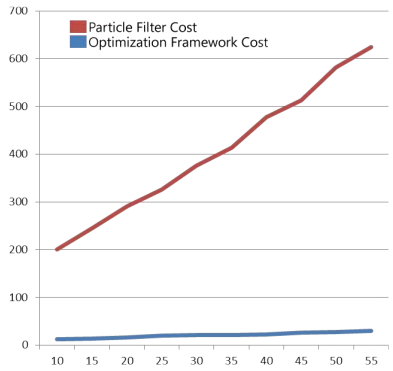


Figure 5.18 – Computation cost comparison between the particle filter system and the optimization framework. The x-axis represents the number of tracked people and the y-axis represents the computation time (in milliseconds)

set. Our motion model set does not take into account physiological and psychological pedestrian traits. All pedestrians are modeled with the same sensitivity towards gender and density; our model set does not take into account heterogeneous agent characteristics, which affect the final behavior. These behavior characteristics can introduce additional errors in our confidence estimation. In practice, the performance of the algorithm can vary based on various other attributes of the input video.

As part of our future work, we would like to incorporate the personality characteristics of the pedestrians, along with other characteristics, such as "fundamental diagrams" from pedestrian dynamics. Essentially, we would like to use improved algorithms (such as *WarpDriver*, presented in Chapter 4, which was not yet finished during this project) to increase the accuracy of our tracker and use them in complex environments where we can simultaneously track the pedestrians and vehicles for traffic management systems.

Conclusion and Future Work 6

In this work, we have addressed the realism of crowd simulation algorithms from the point of view of both evaluation and design. We have tackled the evaluation of simulation algorithms from a data-driven point of view, and investigated the effects of their parameters on both their evaluation and flexibility. We have then followed research opportunities opened by this work with applications to insect simulation and pedestrian tracking. Finally, with observations resulting from these projects, we have proposed a new crowd simulation algorithm, solving common simulation artifacts of previous approaches. In this section, we summarize our contributions and discuss possible future work.

1 Contributions

Evaluation and Parameter Estimation We first focused on the evaluation of crowd simulation algorithms. Our goal being to rate how realistic algorithms are with respect to each other, we approached this topic from a data-driven point of view. Additionally, as parameter values have a profound effect on the resulting simulations, we have integrated them into our approach as well.

Consequently, we propose a general framework for the evaluation of crowd simulation algorithms, where metrics are used both to quantify the similarity between ground-truth, real-world data and the corresponding simulations, and to tune the simulation algorithms in order for them to perform as best as possible. The direct result is a means to objectively and fairly compare algorithms among each other, where none can be tweaked post-comparison to improve its score. In addition to comparisons of algorithms, we have also studied how we can use this process to tune algorithms to adapt them to certain tasks, such as replicating either German or Indian pedestrians. Finally, we have investigated how this process can help artists/animators in simulating certain situations without individually tweaking each agent's trajectory.

Collision Avoidance In our last contribution, we have focused on collision avoidance. Collision avoidance is at the base of every crowd simulator and is largely responsible for the plausibility of the synthesized crowd. While existing approaches have used first-order information (positions) followed by second-order information (instantaneous velocities), progressively improving the quality of simulations, many artifacts still persist, suggesting that these sources of information are not enough. Thus, we have sought to solve the remaining artifacts by considering more sources of information while performing collision avoidance.

As a result, we propose a new collision-avoidance algorithm, where steering decisions are based on perceived probabilities of collision between agents. In order to compute these probabilities, we first define the *Intrinsic Field* which gives the collision probabilities between agents that are only due to their co-existance (assuming an agent is not reduced to a point). We further warp this field using *Warp Operators* which model every source of information we take into account. Our simulated agents are thus capable of avoiding collisions based on multiple sources of information, such as positions, velocities, future perception uncertainty, environment layout (non-linear), interactions with obstacles (non-linear), and past trajectories (non-linear). By taking into account these various sources of information, we are able to solve commonly observed simulation artifacts of previous techniques such as slowdowns and visually erroneous agent agglutinations, unnatural oscillation motions, or exaggerated/last-minute/false-positive avoidance manoeuvres.

Applications to Evaluation and Parameter Estimation We have first applied our work on evaluation and parameter estimation to insect simulation. With the recent availability of data on swarming insects' trajectories, we have applied our previous work to the simulation of such swarms. With a lack of available data-driven solutions for this task, we have sought to fill this void by implementing the three observed levels of insect behavior: microscopic, mesoscopic, and macroscopic. We propose a data-driven, biologically-inspired insect simulator, where parameter estimation is used to select and tune a collision-avoidance algorithm for microscopic behavior, statistically correct mesoscopic trajectories are generated from observed noisy trajectories, and where the global swarm behavior is ensured by the Metropolis-Hastings algorithm. The resulting simulations are easy to control by artists/animators (e.g. using sketches), the swarms can avoid obstacles in the environment and it is also possible to give them different shapes and behaviors. Overall, the produced simulations are realistic as evidences by commonly used metrics.

Second, we have applied our work on evaluation and parameter estimation to pedestrian tracking. For this application, we based ourselves on particle filters, which are a commonly used tracking approach that yields good tracking results and that is applicable in real time. This particle filter approach futher has the property of relying on motion priors to track pedestrians. Thus, we propose an improved particle-filter-based tracking algorithm where the motion prior is given not by one simulator with a static set of parameter values, but by a mixture of simulators with varying parameter values. At each frame of video data, we use our parameter estimation and evaluation work to tune all available crowd simulation algorithms to match the past observed pedestrian trajectories as closely as possible, and subsequently select the best simulator. This best simulator with the optimal set of parameter values is then used as motion prior for the particle filter. With this approach, we are able to considerably improve tracking accuracy as evidenced by our tests on challenging datasets.

2 Future Work

Evaluation Following our work on evaluation and parameter estimation, we would like to study how parameters can be generalized beyond a given scenario. In our current applications, particular sets of parameters are very specific to the situation they were "learned on", during evaluation, we wish the algorithm to perform as well as possible on the tested case, and in tracking we wish the algorithm to describe (reproduce) the currently observed situation as closely as possible. We have already dealt a bit with this generalization in our work on insects, though purely thanks to the relatively "massive" amount of trajectory data involved. More generally, this question ties into the question of the "representativeness" of the data: can we find situations which are widely representative of most types of interactions? This is also a good reason to collect more data, hence the need for a good tracking algorithm for instance.

The second research direction, given that parameter values optimal with respect to a current situation yield better results than general-purpose ones, is to tune a simulation algorithm's agent's parameter values with respect to the "context" the agent is in (similar to what we have done in the context of pedestrian tracking). An agent could thus have a certain set of values when walking out in the open with few interactions, and a different set when part of a dense crowd at a concert for instance. Though these two examples are trivially different, a method could be devised to automatically tune agents during much subtler changes of context.

Collision Avoidance Finally, in terms of collision avoidance, many possible evolutions are possible. First, as the approach is very flexible, we could investigate additional sources of information, such as social factors: left/right conventions, yielding to elders, etc. The approach is also suitable to incorporate the dangerousness of interactions: one would likely be willing to collide with another person if it meant avoiding a collision with a car. The information on environment layout could also be learned from data (similar to estimating the pedestrian flows during tracking, as mentioned earlier).

Another direction is the commonly studied case of group motion, which could be trivially implemented by changing the *Intrinsic Field*. Finally, the approach can benefit a lot from work on accelerating it, mainly work on raytracing from the field of rendering (as the simulator can be viewed as a ray-marching algorithm).

Applications to Evaluation and Parameter Estimation The first direction for future work in terms of insect simulation is the extension of our approach to mixed interactions of various types of insects. This would likely involve work on differentiating the trajectories (e.g. through clustering) of each type of insect found in the source data. Of course, it would also involve collecting the necessary data. Thus far, we have used our simulation pipeline on 3D insect trajectories exlusively. While our approach is capable of handling 2D motions we have not been able to test this as we lack the necessary data (again highlighting the need to collect more data). Finally, we use a static field to guide the global motion of insects, and it would be interesting to consider dynamic fields, automatically adapting to changes of destination and environment.

In terms of pedestrian tracking, our current algorithm only considers the sources of information that are classically used by microscopic, agent-based crowd simulation algorithms (mainly due to the available simulators). We would like to extend this to take other sources of information into account. Concretely, other tracking approaches detect agent flows in videos which they then use as motion prior. For instance with our new simulator WarpDriver (which uses sources of information such as the layout of the environment) as part of the tracker, we could detect pedestrian flows and feed them as environment layout information to WarpDriver. Adding such types of additional information to the tracking system would likely further improve tracking results.

3 Summary

Overall, many of the investigated topics as well as research avenues for future work are linked between these various topics. Our new simulation algorithm WarpDriver has benefited from observations made while working on evaluation, and the progress we made developing this simulator can improve our tracking approach (especially considering adding more sources of information), which in turn can help us collect more data, based on which we can learn context-dependent sets of parameter values, that could further push forward the realism of crowd simulation algorithms.

Résumé en Français

7

La demande pour la simulation de foules a fortement progressé au cours des dernières années, avec l'industrie du divertissement et la sécurité urbaine au premier rang de ses applications. Des longs-métrages et jeux vidéo de plus en plus ambitieux font appel à des armées et foules d'arrière-plan de plus en plus conséquentes, tandis que des règles de sécurité de plus en plus strictes requièrent des architectes une prédiction de plus en plus précise des comportements de foules.

Les superproductions en particulier préfèrent maintenant les foules numériques ("foules" se réfère ici à toute collection d'entités représentées à l'écran) à l'emploi de grandes quantités de figurants en vue de peupler leurs scènes. Ainsi, il est mainetant possible d'observer des foules synthétiques dans des travaux et contextes variés. Par exemple, de très larges armées sont au centre de longs-métrages ou séries télévisées tels que *Le Seigneur des anneaux*, *300*, *Le Trône de fer*, etc. Ces foules numériques peuvent aussi prendre d'autres formes telles que les larges quantités de zombies dans *World War Z*, de singes dans *La Planète des singes : l'affrontement*, de minions dans la saga *Moi moche et méchant*, etc. Le pré-requis principal pour animer ces foules (en plus de la qualité des mouvements), est un degré élevé de contrôle de l'artiste sur la simulation, imposant les comportements et les styles nécessaires à chaque projet.

Dans les jeux vidéo, les simulateurs sont en charge des déplacements de tout personnage (interactif) non joueur, allant des passants déambulant les rues d'*Assassin' Creed* aux innombrables soldats de la série de jeux *Dynasty Warriors*. Alors que les jeux vidéo requièrent tout comme les œuvres télévisuelles des forts degrés de contrôle et de réalisme/crédibilité, les jeux vidéo nécessitent en plus que les personnages soient interactifs et donc autonomes, avec des simulateurs les animant en temps réel.

En termes des sécurité et de conception urbaine, la simulation de foules est similairement largement d'actualité. Par exemple, les organisateurs de larges évènements "ouverts" (e.g. concerts) peuvent utiliser des simulateurs pour faire des prédictions statistiques sur le comportement des participants dans le but d'améliorer leurs installations ou projets de parcours (espérant éviter des drames comme ce fut le cas durant le festival de musique *Love Parade* à Duisburg, en Allemagne, en juillet 2010). Evidemment, des techniques similaires peuvent être employées pendant la conception de nouveaux bâtiments tels que les aéroports/gares, galleries commerciales, bureaux, bateaux de croisière, etc. De plus, dans le cas de telles structures, il est aussi possible de suivre les personnes pendant les évacuations pour anticiper l'accès aux sorties, et finalement de guider les personnes le long d'itinéraires optimaux, évitant les congestions. Il est aussi possible d'utiliser un simulateur pour des applications plus spécifiques comme l'amélioration de files d'attentes dans les parcs d'attractions, aéroports/gares, etc. Globalement, dans ces cas, l'aspect le plus important est la précision (principalement statistique) des foules

simulées ainsi que leur capacité à anticiper les issues possibles à une situation spécifique, avec des cas tels que les évacuations nécessitant aussi de fonctionner en temps réel.

Avec cette multitude d'applications, de nombreux algorithmes ont été développés, souvent dans des buts bien précis. Par conséquent, le choix d'un algorithme pour une tâche particulière n'est pas simple.

1 Problème

Supposant un large éventail d'algorithmes disponibles pour simuler des foules dans un projet donné, la tâche de l'utilisateur faisant le choix de quel algorithme utiliser est de répondre à un certain nombre de questions fondamentales.

Performances A quel point l'algorithme doit-il être rapide ? Les applications interactives (e.g. les jeux vidéo) requièrent des foules animées en temps réel alors que les œuvres télévisuelles peuvent se permettre d'utiliser des techniques plus coûteuses. De même, un système de guidage pendant une évacuation doit pouvoir s'adapter rapidement à toute éventualité alors que la validation d'un nouveau bâtiment est moins urgente.

Autonomie La question suivante est : quel est le degré d'implication nécessaire de l'utilisateur pour réaliser son but ? L'utilisateur peut-il s'attendre à un résultat correct en précisant juste "1000 humains, d'ici à ici, bougeant vite, apeurés", ou doit-il vérifier et modifier le comportement de chaque individu à la main (tout en ayant conscience que toute modification peut causer une réaction en chaîne affectant la simulation dans son ensemble) ?

Contrôle Une autre facette concerne le contrôle (ou flexibilité), i.e. est-ce possible pour l'utilisateur d'utiliser le simulateur pour arriver à ses fins ? Ce même simulateur pourrait-il produire une autre foule suivant une autre spécification telle que "1000 touristes, d'ici à ici, bougeant lentement, curieux" ou a-t-il un domaine de validité restreint ?

Réalisme La dernière question est évidente : la simulation sera-t-elle réaliste/crédible ? Un algorithme ne serait pas d'une grande utilité à un artiste si les entité simulées brisaient l'immersion des spectateurs, où à un architecte étudiant les cas d'évacuation si les personnages simulés ne se comportaient pas suffisemment comme des humains. Un simulateur pouvant être plus adapté à certains usages que d'autres, cette question est aussi liée à celle du domaine de validité de l'algorithme : supposant une palette d'algorithmes, lequel est le plus adapté à un usage donné ?

Alors que toutes ces questions doivent être considérées lors du choix (ou du développement) d'un algorithme, certaines sont plus difficiles que d'autres. En règle générale, la question des performances a une réponse simple, soit par estimation de la complexité théorique de l'algorithme, soit en l'essayant brièvement. Les questions d'autonomie et de contrôle sont plus difficiles, puisqu'elles demandent un degré de familiarité avec les

capacité d'un algorithme : principalement sont domaine de de validité (instabilités à certaines densités, personnages holonomes/non-holonomes, etc.) et les effets des paramètres (influençant par exemple le degré de "méfiance" des personnages entre eux). Enfin, la question du réalisme est de loin la plus difficile puisque : (1) dans bien des cas "réalisme" n'a pas de définition nette, et (2) certains algorithmes pourraient en théorie produire le résultat attendu mais leur paramétrisation n'est ni connue ni triviale.

Ainsi, l'objectif principal de cette thèse est d'améliorer de manière générale le réalisme des algorithmes de simulation de foules en : (1) mettant au point un schéma général pour l'évaluation du degré de réalisme des algorithmes (tout en gardant à l'esprit les questions d'autonomie et de contrôle), et en (2) développant des algorithmes plus avancés.

2 Approche

En tant que point de départ à notre travail, nous avons choisi d'approcher cette question du réalisme du point de vue des algorithmes microscopiques basés agent (détaillés dans la Section 2), puisqu'ils représentent une classe d'algorithmes fortement utilisés grâce à leur facilité d'utilisation et d'implémentation ainsi que leur flexibilité.

De ce point de vue, nous avons abordé en premier l'évaluation des algorithmes de simualtion de foules. Les deux objectifs principaux de ce travail étaient de valider les algorithmes existants et de développer un cadre de travail pour valider les idées futures. Nous avons appuyé notre schéma d'évaluation sur des données réelles, en utilisant des métriques variées pour comparer les simulations aux comportement réels. En complément, nous avons aussi incorporé à cette méthode l'estimation automatique de paramètres nous permettant de faire deux choses. Premièrement la possiblité de comparaisons impartiales des algorithmes, chacun d'eux étant configuré de manière optimale pendant les tests, deuxièmement l'exploration de la question du contrôle, la configuration automatique des simulateurs pouvant aider l'utilisateur à les adapter à différentes tâches.

Durant ce travail il est devenu évident que malgré les avancées récentes sur les simulateur microscopiques basés agent, il reste de nombreux artéfacts et erreurs de simulation. Cela nous a conduit au deuxième travail principal du présent document, consernant le développement d'algorithmes de simulation. Succintement, alors que les algorithmes de "premier ordre" (voir Chapitre 2, Section 2) sont simples à implémenter, étendre et utiliser, leurs résultats de simulation ne sont pas aussi bons que ceux produits par les algorithmes de "second ordre" (qui anticipent les collisions par extrapolation linéaire du mouvement des agents), qui sont par contre plus complexes à étendre, et produisent aussi des artéfacts évidents. Pour continuer d'améliorer les résultats de simulation, nous avons conçu un algorithme facilement extensible, fonctionnant dans un espace tridimensionnel (positions 2D plus temps) où les agents *perçoivent* les probabilités de collision entre eux. Ces probabilités sont calculées sur la base d'un champs *intrinsèque* de probabilités de collision, qui représente les probabilités de collisions induites par leur co-existence (les agents ne sont pas réduits à des points), et qui est ensuite *déformé* par chaque source d'informations que nous souhaitons prendre en compte. Ainsi, nous avons implémenté des *opérateurs de distortion* qui modélisent : l'erreur de perception des agents (il perçoivent mieux les situations imminentes que celles plus éloignées dans le

temps), la taille des agents, l'anticipation de trajectoires par extrapolation (linéaire) de leur vitesse , les erreurs de perception liées à la vitesse, la'anticipation liée à l'agencement de l'environnement (non-linéaire), l'anticipation liée aux trajectoires passées des agents (non-linéaire), et l'anticipation liée aux conséquences des interactions entre les agents et les obstacles (i.e. l'impossibilité d'un agent de traverse un obstacle; non-linéaire). De plus, nous pouvons facilement visualiser ces probabilités de collision pendant la vérification des résultats de simulation, rendant l'algorithme plus simple à analyser et étendre.

En parallèle, en explorant et cherchant des données, notre travail sur l'évaluation et l'estimation de paramètres a donné lieu à deux autres applications. Dans ces applications, nous abordons la question du domaine de validité de chaque algorithme de simulation. Ainsi, nous utilisons notre travail en tant qu'outil de sélection pour déterminer quel algorithme est le plus approprié pour (1) une simulation ou (2) un instant d'une simulation.

La première application concerne la simulation d'essaims d'insectes, puisque des données sur leurs trajectoires de vol ont récemment été rendus accessibles. Par conséquent, nous avons utilisé lors de la conception d'un simulateur d'insectes basé données. Nous avons utilisé l'estimation de paramètres pour choisir l'algorithme le plus approprié et le configurer pour reproduire les comportements de bas niveau des insectes, puis nous avons complété le système avec des méchanismes statistiques supplémentaires pour prendre en charge les trajectoires en "zig-zags" des insectes (niveau intermédiaire) et leur comportement de haut-niveau au sein de l'essaim.

La seconde application découlant de notre travail sur l'évaluation et l'estimation de paramètres concerne le suivi de piétons. Ici, nous avons adopté une approche connue où un algorithme de suivi est associé à un simulateur qui l'aide et prédisant le mouvement des piétons. Nous nous sommes occupés à améliorer cette approche d'une manière générale et indépendante des simulateurs utilisés. Conformément à nos observations lors de notre travail sur l'estimation de paramètres, il est plus facile de reproduire complètement le comportement de foule à tout instant avec plusieurs simulateurs bien configurés plutôt qu'avec un seul algorithme avec un seul jeu de paramètres. Ainsi, nous utilisons les portions de trajectoires déjà identifiées pour estimer les paramètres optimaux de plusieurs simulateurs concurrents. Ensuite, nous sélectionnons le simulateur (configuré) qui décrit le mieux ces portions déjà connues pour prédire les positions suivantes des piétons et ainsi aider l'algorithme de suivi, améliorant fortment la qualité du suivi.

3 Contributions

Suivant le titre de cette thèse, "Simulation microscopique de foules : **évaluation** et **développement** d'algorithmes", nos deux contributions principales (en tant que premier auteur) sont comme suit :

Evaluation et estimation de paramètres, Chapitre 3 Nous proposons une méthode visant à évaluer le réalisme des algorithmes de simulation de foules d'une manière objective et impartiale. "Objective" grâce à des métriques quantifiant la similitude entre les simulations et des donnés acquises en situation réelle. "Impartiale" grâce à l'estimation de paramètres permettant d'étalonner automatiquement les algorithmes en vue de décrire

au mieux les données (par rapport aux métriques), permettant de comparer les algorithmes au mieux de leur capacité. Nous explorons aussi comment ce processus permet d'augmenter le niveau de contrôle d'un utilisateur sur la simulation tout en réduisant son implication.

Evitement de collision, Chapitre 4 Nous proposons un nouvel algorithme d'évitement de collisions. Alors que les algorithmes existants prédisent les collisions en extrapolant linéairement les trajectoires des agents, nous allons au-delà grâce à une approche probabiliste et non-linéaire, prenant en compte entre autres la configuration de l'environnement, les trajectoires passées et les interactions avec les obstacles. Nous éliminons ainsi des simulations résultantes des artéfacts tels que: les ralentissements et les agglomérats dérangeants d'agents, les mouvements oscillatoires non naturels, ou encore les manœvres d'évitement exagérées/fausses/de dernière minute.

Evaluation et estimation de paramètres : applications Dans une troisième contribution, nous abordons aussi l'utilisation de notre travail sur l'évaluation et l'estimation de paramètres dans le cadre de systèmes plus larges (en tant que second auteur).

Simulation d'insectes, Chapitre 5, Partie 1 Premier auteur : Weizi Li, étudiant en thèse au Gamma Group de l'université de Caroline du Nord à Chapel Hill, Caroline du Nord, Etats-Unis. Nous appliquons notre travail à la simulation d'insectes, prenant en charge leur comportement local. Après avoir complété le système aux niveaux intermédiaire et global, cette approche basée-données est capable de simuler correctement des essaims d'insectes.

Suivi de piétons, Chapitre 5, Partie 2 Premier auteur : Aniket Bera, étudiant en thèse au Gamma Group de l'université de Caroline du Nord à Chapel Hill, Caroline du Nord, Etats-Unis. Nous appliquons aussi notre travail au suivi de piétons, construisant un "méta-algorithme" servant à calculer la probabilité de transition d'un filtre particulaire, et surpassant les systèmes existants.

Appendices

Craal: Parameter Estimation and Comparative Evaluation of Crowd Simulations

A

1 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):

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

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

1.1 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:

$$D_k = \|\mathbf{z}_k - \mathbf{x}_k\|. \tag{A.2}$$

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

$$L_k = (\mathbf{z}_{k+1} - \mathbf{z}_k) - (\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{A.3}$$

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:

$$I_k = \sum_P |\mathbf{x}_k^i - \mathbf{x}_k^j| - \sum_P |\mathbf{z}_k^i - \mathbf{z}_k^j|. \tag{A.4}$$

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.*

$$P_k = \|\mathbf{z}_{k+1} - f(\mathbf{z}_k, speed(\mathbf{z}_k), \mathbf{z}_m, \mathbf{p})\|. \tag{A.5}$$

1.2 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:

$$V_k = \mathbf{z}_k - (\nabla \times \overrightarrow{v}), \tag{A.6}$$

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 ($\frac{numberAgentsInsideGate}{areaOfGate}$) at an agent's location when the agent is inside this gate.

$$F_k = |\mathbf{z}_k(d_k) - \|\mathbf{v}_k\||, \tag{A.7}$$

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.

2 Optimization Techniques

2.1 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.

2.2 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: 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
```

Algorithm 1 gives the pseudocode for the process where:

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.

2.2.1 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 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).

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
```

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).

2.2.2 Covariance Matrix Adaptation

Lastly, we also tested the CMA optimization algorithm [HO96], 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.

3 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)

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 A.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.

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 Section 4.

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.

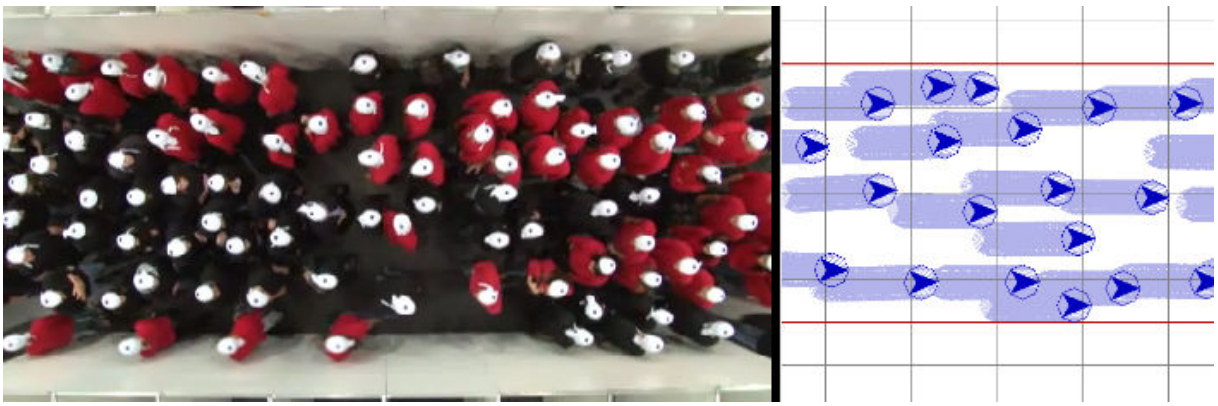
We batched several tests into three sets of scenarios. The first set of scenarios (Fig A.1a) involved a small number of agents (2-5 agents) crossing paths to reach their goals; for this evaluation, we used the Difference metric (D). Here GA+G and GA+SA algorithms give the best score improvements and the CMA algorithm is the fastest. The second set of scenarios comes from the data in [ZKSS12] (Fig A.1c). 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 A.1b), 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.

Figure A.3 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 A.4 shows the Friedman and Wilcoxon tests' results

(a) Crossing Scenario

(b) Circle-24 Scenario



(c) Hallway Scenario

Figure A.1 – 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]).

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 A.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.

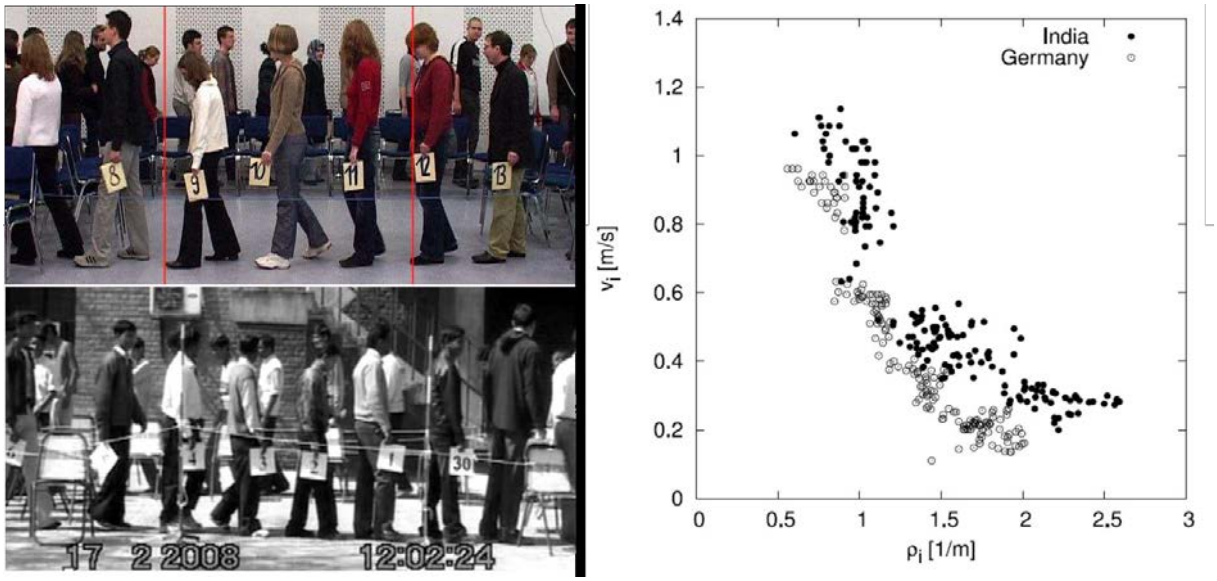


Figure A.2 – .
]Figures extracted from [Chattaraj et al. 2009]. Subjects from India (top) and Germany (bottom) were asked to walk in a line. Video analysis was performed to extract fundamental diagrams (speed vs. density).

for the score. Figure A.5 shows the Friedman and Wilcoxon tests' results for runtime.

4 Initial Parameters for Optimization

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 A.3 – Default values for simulation parameters for the 5 models integrated to the framework

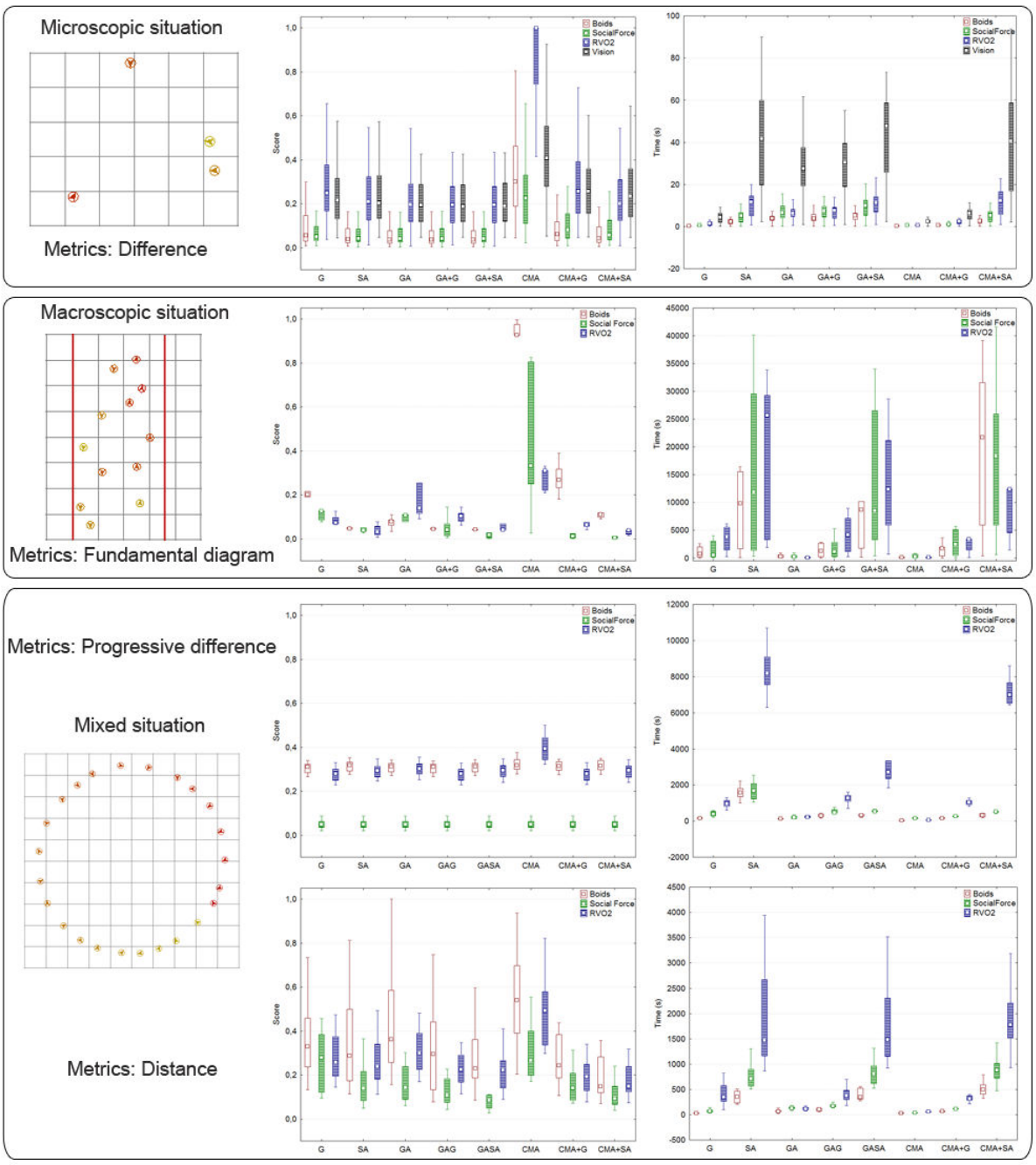


Figure A.3 – Summary of the experiment testing optimization algorithms. Each row shows results for a different type of data. In each row, the left side describes the data and metric. The middle shows $\frac{scoreAfterClibration}{scoreBeforeClibration}$ with a lower value indicating better optimization. The right shows time in seconds. Each graph shows its results for each model for each optimization method.

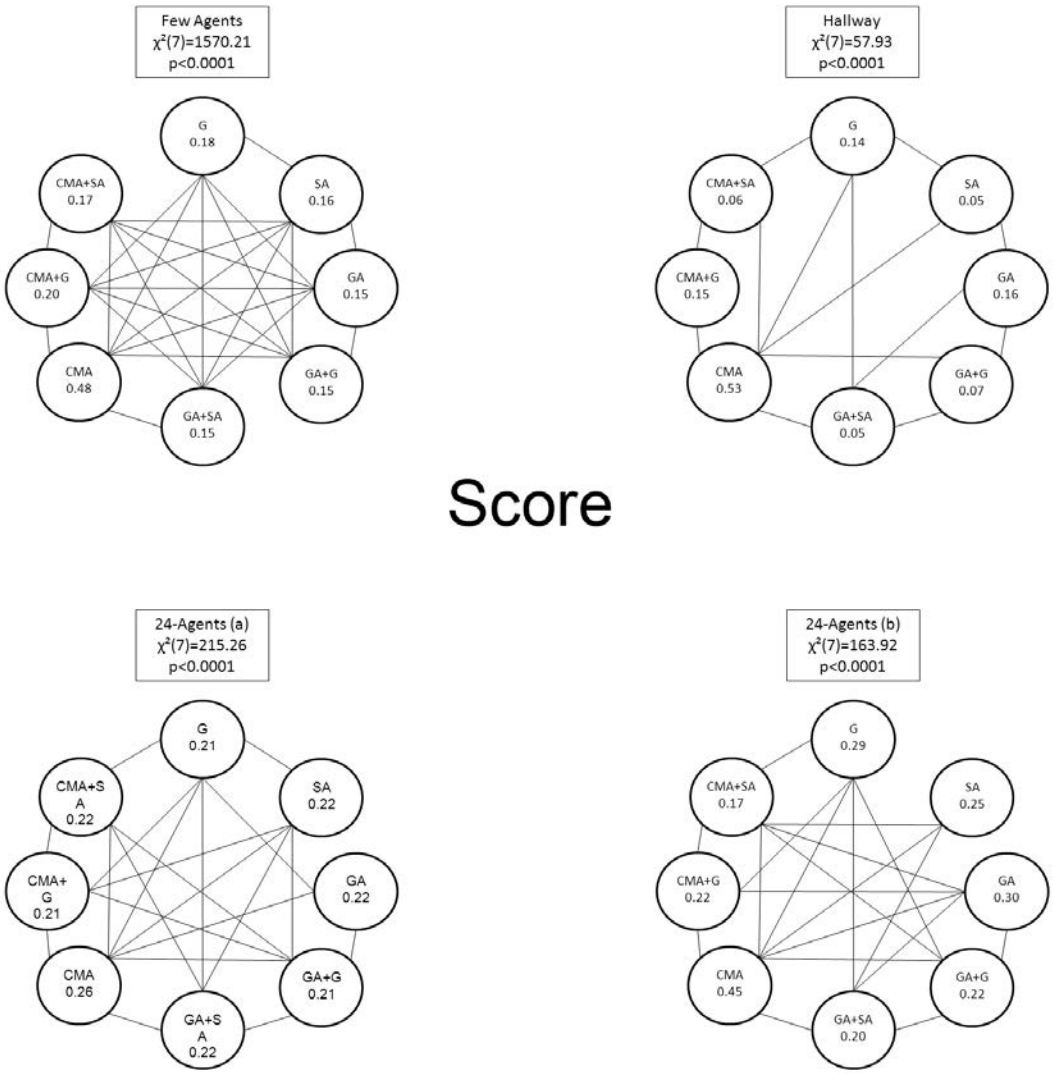


Figure A.4 – Statistical results of the Friedman's Anova and Post-Hoc Wilcoxon signed rank tests on the score depending on the optimization algorithm. Mean values are reported for each algorithm. Friedman's Anova showed an influence of the optimization algorithm on the score for each type of data (Few Agents, Hallway, 24-Agents (a), 24-Agents (b)). Significant differences between algorithms are represented through a line ($p < 0.0018$).

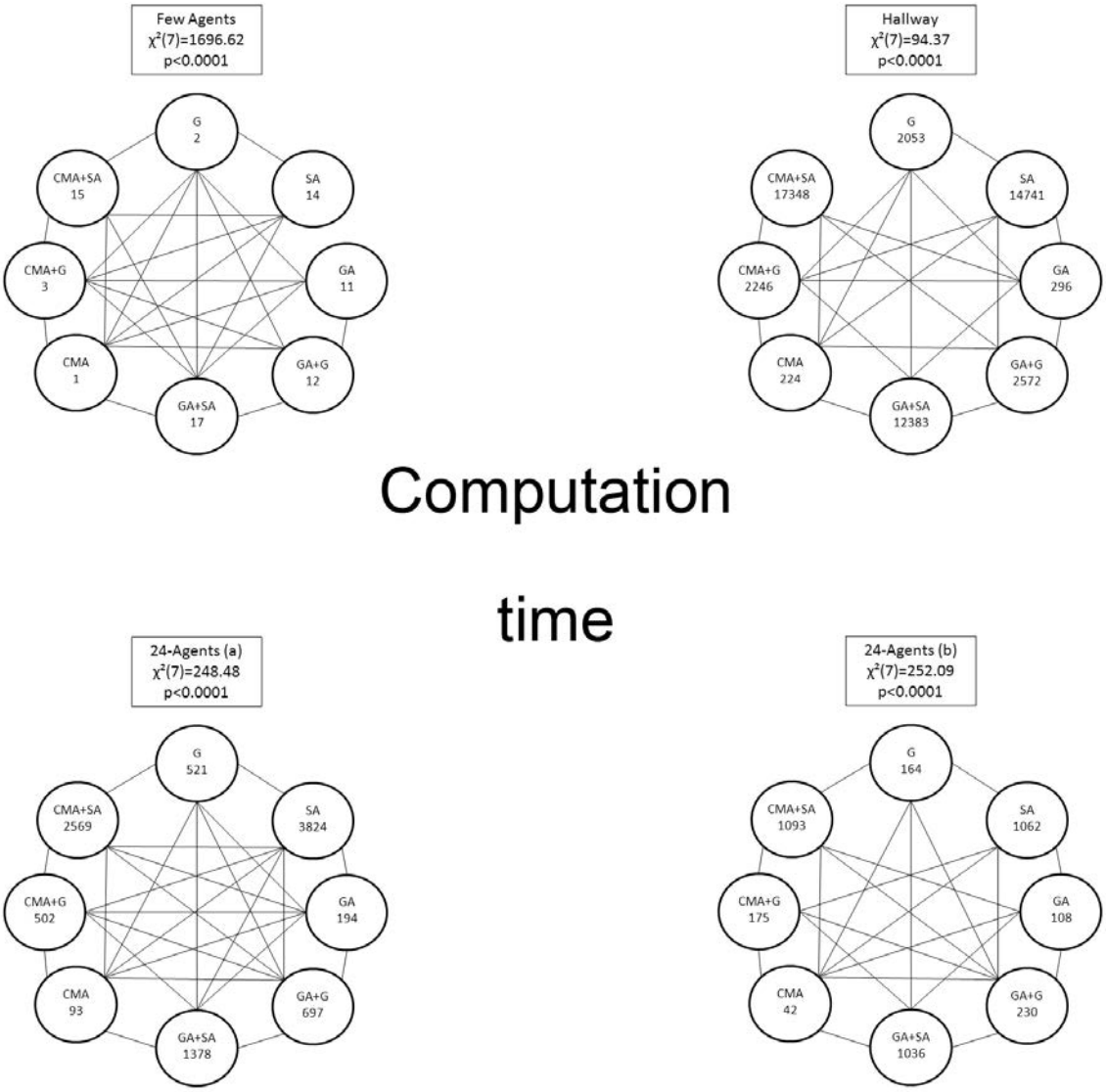


Figure A.5 – Statistical results of the Friedman's Anova and Post-Hoc Wilcoxon signed rank tests on the computation time depending on the optimization algorithm. Mean values (s) are reported for each algorithm. Friedman's Anova showed an influence of the optimization algorithm on the computation time foreach type of data (Few Agents, Hallway, 24-Agents (a), 24-Agents (b)). Significant differences between algorithms are represented through a line ($p < 0.0018$).

B WarpDriver: Context-Aware Probabilistic Motion Prediction for Crowd Simulation

1 Warp Operators

Warp Operators model each agent *property* that we want to include in the algorithm. As mentioned in Chapter 4 Section 3, these could be: shape, size, position, velocity, followed path, etc. Mechanically, *Warp Operators* warp the *Intrinsic Field* defined for each *perceived* agent b in its *agent-centric* space-time $\mathcal{S}_{b,k}$ into the *perceiving* agent a's *agent-centric* space-time $\mathcal{S}_{a,k}$.

In this sub-section, we describe *Warp Operators* modeling agent-related and context-related *properties*.

1.1 Agent-Related Operators

This section details *Warp Operators* modeling agent-related *properties*. For each *Warp Operator* W, we give its direct and inverse expressions:

- $W(\mathbf{s})$ for a point $\mathbf{s} \in \mathcal{S}$,
- $W^{-1}(p)$ for a probability $p \in [0,\ 1]$,
- and $W^{-1}(\mathbf{g})$ for a probability gradient $\mathbf{g} \in \mathbb{R}^3$.

Position and Orientation The *Warp Operator* W_{local} models the agents' position and orientation *properties*. It is a simple change of referential between $\mathcal{S}_{a,k}$ and $\mathcal{S}_{b,k}$.

Time Horizon To avoid collisions in a time horizon $\mathcal{T}$ (beyond the normalized 1 second in the *Intrinsic Field*), we define a time horizon operator W_{th} such that:

$$W_{th}(\mathbf{s}) = \mathbf{s} \star (1,\ 1,\ 1/\mathcal{T}), \tag{B.1}$$
$$W_{th}^{-1}(p) = p, \tag{B.2}$$
$$W_{th}^{-1}(\mathbf{g}) = \mathbf{g}. \tag{B.3}$$

Time Uncertainty This operator W_{tu} models the increased uncertainty on the states of other agents the further we look in time. It is a dilation by a coefficient which scales from 1 at $t = 0$ to $1 + \alpha$ at $t = 1$ (α being a user-set parameter):

$$W_{tu}(\mathbf{s}) = \mathbf{s} \star (\beta,\ \beta,\ 1), \tag{B.4}$$
$$W_{tu}^{-1}(p) = p\beta^2, \tag{B.5}$$
$$W_{tu}^{-1}(\mathbf{g}) = (\beta\mathbf{g}.\mathbf{x},\ \beta\mathbf{g}.\mathbf{y},\ \gamma_1\mathbf{g}.\mathbf{x} + \gamma_2\mathbf{g}.\mathbf{y} + \mathbf{g}.\mathbf{t}), \tag{B.6}$$

where $\beta = 1/(1 + \alpha\mathbf{s}.\mathbf{t}),\ \gamma_1 = -\alpha\beta^2\mathbf{s}.\mathbf{x},\ \gamma_2 = -\alpha\beta^2\mathbf{s}.\mathbf{y}$.

Radius To change the radius of the agents, we dilate space along the $\mathbf{x}$ and $\mathbf{y}$ axes by a factor equal to the sum α of the perceiving and perceived agents' radii (radius of their Minkowski Sum):

$$W_r(\mathbf{s}) = \mathbf{s} \star (1/\alpha,\ 1/\alpha,\ 1), \tag{B.7}$$
$$W_r^{-1}(p) = p, \tag{B.8}$$
$$W_r^{-1}(\mathbf{g}) = \mathbf{g}. \tag{B.9}$$

Velocity We model the agent's instantaneous velocity as a displacement along the $\mathbf{x}$ axis:

$$W_v(\mathbf{s}) = \mathbf{s} + (-v_{a,k}\mathbf{s}.\mathbf{t},\ 0,\ 0), \tag{B.10}$$
$$W_v^{-1}(p) = p, \tag{B.11}$$
$$W_v^{-1}(\mathbf{g}) = \mathbf{g} + (0,\ 0,\ -v_{a,k}\mathbf{g}.\mathbf{x}). \tag{B.12}$$

Velocity Uncertainty Depending on the speed of an agent, that agent could be more or less likely to make certain adaptations to its trajectory. For instance, the faster an agent travels, the more likely it is to accelerate/decelerate rather that turn. Given two user-set parameters α_1 and α_2 and the agent's preferred speed v_{pref}:

$$W_{vu}(\mathbf{s}) = \mathbf{s} \star (\beta_1,\ \beta_2,\ 1), \tag{B.13}$$
$$W_{vu}^{-1}(p) = p\beta_1\beta_2, \tag{B.14}$$
$$W_{vu}^{-1}(\mathbf{g}) = \mathbf{g} \star (\beta_1,\ \beta_2,\ 1), \tag{B.15}$$

where: $\beta_1 = 1/(1 + \alpha_1\frac{v_{a,k}}{v_{pref}}),\ \beta_2 = 1 + \alpha_2\frac{v_{a,k}}{v_{pref}}$.

1.2 Context-Related Operators

The following operators provide information based on the Environment Layout (operator W_{el}), Interactions with Obstacles (operator W_{io}) and Observed Behaviors of agents (operator W_{ob}). These operators, where applicable, **replace** the *Local Space operator* $W_{local,a\to b}$. We call W_{ref} the resulting operator: $W_{ref} = \{W_{local}\ or\ W_{el}\ or\ W_{io}\ or\ W_{ob}\ \}$. To achieve their goals, these operators rely on a common algorithm (although there can be variations in the implementation to accelerate the execution time where possible) which can deform (and split) space to arbitrary shapes defined as graphs. We first introduce the algorithm and then describe what shapes/graphs we use for each operator. Chapter 4 Figure 4.4 shows examples of curved motion prediction.

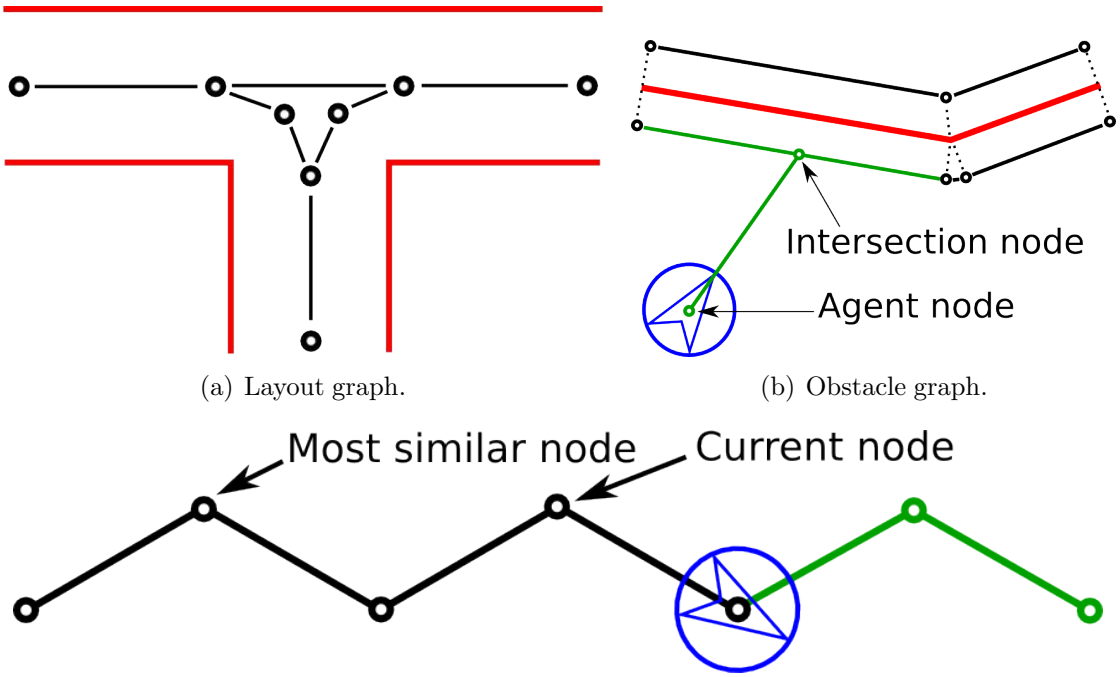
Warping Space to Arbitrary Shapes To describe the graph-based, arbitray-shape-warping algorithm, we first introduce a few more notations.

- $\mathcal{N}$ is the set of nodes composing the input, shape-defining graph.
- $n \in \mathcal{N}$ with various subscripts denotes nodes from this set. All operations that apply to points $\mathbf{s} \in \mathcal{S}$ also apply to nodes. n_a is the node that is closest to an agent a.
- $\forall \mathbf{v} \in \mathbb{R}^3$, $\|\mathbf{v}\|$ denotes the norm of $\mathbf{v}$.
- $\forall n_1, n_2 \in \mathcal{N}$, $edgeDistance(n_1, n_2)$ is the distance between nodes n_1 and n_2 along the shortest path linking them in the graph.
- $\forall n_1, n_2 \in \mathcal{N}$, v_{n_1,n_2} denotes the vector between nodes n_1 and n_2.
- $\forall v_1, v_2 \in \mathbb{R}^3$, $angle(v_1, v_2)$ is the angle in radians between vectors v_1 and v_2.
- $\forall \mathbf{s} \in \mathcal{S}$, $\forall \alpha \in \mathbb{R}$, $\forall \mathbf{v} \in \mathbb{R}^3$, $rotate_{\alpha,\mathbf{v}}(\mathbf{s})$ rotates $\mathbf{s}$ around $\mathbf{v}$ by α radians.

Given these notations, the procedure is described in Algorithm 3. First, we find the starting node n_a which is the closest node to the considered agent a. Then, we find the two nodes n_1 and n_2 that are closest to point $\mathbf{s}_{in}$ and reorder them so that, from these two nodes, n_c is the node that is closer to n_a and n_f is further from n_a (first four lines in Algorithm 3). Given these two nodes, $\mathbf{s}_{local}$ represents $\mathbf{s}_{in}$'s coordinates in the new, local, referential, where n_c is the local origin and v_{n_c,n_f} is the local $\mathbf{x}$ axis (next three lines in Algorithm 3). Finally, from this local point $\mathbf{s}_{local}$, we can compute the final $\mathbf{s}_{out}$ coordinates (last line in Algorithm 3). $\mathbf{s}_{out}$'s x coordinate is the distance between n_a and $\mathbf{s}_{local}$'s projection on the local $\mathbf{x}$ axis (sum of $edgeDistance(n_c, n_a)$ and $\mathbf{s}_{local} \cdot \mathbf{x}$). $\mathbf{s}_{out}$'s y coordinate is $\mathbf{s}_{local}$'s y coordinate ($\mathbf{s}_{out} \cdot \mathbf{y}$). And, finally, the time coordinate is left unchanged.

Algorithm 3: Algorithm which aligns space with a graph. From a point $\mathbf{s}_{in} \in \mathcal{S}$, it gives the new coordinates $\mathbf{s}_{out}$ along a graph $\mathcal{N}$.

$n_1 \leftarrow \text{argmin}_{n \in \mathcal{N}} \|\mathbf{s}_{in} - n\|$
$n_2 \leftarrow \text{argmin}_{n \in \mathcal{N} \setminus n_1} \|\mathbf{s}_{in} - n\|$
$n_c \leftarrow \text{argmin}_{n \in \{n_1, n_2\}} edgeDistance(n, n_a)$
$n_f \leftarrow \text{argmax}_{n \in \{n_1, n_2\}} edgeDistance(n, n_a)$
$\alpha \leftarrow angle(\mathbf{x}, v_{n_c,n_f})$
$\mathbf{s}_{local} \leftarrow \mathbf{s}_{in} - n_c$
$\mathbf{s}_{local} \leftarrow rotate_{-\alpha,\mathbf{t}}(\mathbf{s}_{local})$
$\mathbf{s}_{out} \leftarrow (edgeDistance(n_c, n_a) + \mathbf{s}_{local} \cdot \mathbf{x},\ \mathbf{s}_{local} \cdot \mathbf{y},\ \mathbf{s}_{in} \cdot \mathbf{t})$



(a) Layout graph.

(b) Obstacle graph.

(c) History graph.

Figure B.1 – Graphs. (a): Example of a graph (in black) defining probable trajectories at an intersection between three hallways (walls in red). (b): Example of an interaction between an agent (blue) and an obstacle (red). In black is the obstacle graph and in green are the temporary agent and intersection nodes (as well as their connections to the obstacle graph). (c): Simple example of a prediction based on an agent's observed history. In black is the past, observed trajectory of the agent (in blue), in green is the predicted trajectory based on the current node and the node most similar to it.

Environment Layout When navigating in an environment, based on its layout, we can predict what trajectories other pedestrians are likely to follow. In a series of hallways, for instance, when not threatened by collisions with other pedestrians, one would stay roughly in the middle of the hallway and take smooth turning trajectories at intersections (an agent could turn either left or right in Figure 4.4(a)). When navigating on curved paths, one would, again, have a tendency to stay roughly in the middle, resulting in a curved trajectory (Figure 4.4(b)). The operator W_{el} models this knowledge by warping space to "align" it with these probable trajectories, which are ultimately very similar to navigation graphs as shown on Figure B.1(a). The space-shape-warping algorithm then uses such a graph as its input.

Interactions With Obstacles Where the environment layout operator focuses on other agents' probable trajectories assuming they will continue travelling, this operator W_{io} takes care of possible interactions between agents and obstacles. These interactions are essentially much more drastic changes to an agent's locomotion than paths, such as full stops. These can occur if, for instance, an agent comes up to a wall (Figure 4.4(c)) (to interact with an ATM, look out the window, check a map...). This can also happen with an agent coming into contact with a small/temporary/unexpected obstacle which could force it to stop and then "hug" the obstacle to get around it.

To achieve this, we construct a graph around each obstacle (an obstacle being modelled as a series of connected line segments) as shown in Figure B.1(b). When an agent's predicted trajectory intersects with an obstacle, we construct two additional nodes, one at the agent's position and one at the intersection between the predicted trajectory and the obstacle's graph. The intersection's node is then connected to the agent's node as well as the obstacle's two closest nodes (see Figure B.1(b)). The resulting graph is then used as input for the space-shape-warping algorithm.

Observed Behaviors With the last operator W_{ob}, we aim to improve the prediction of agents' future motions by looking at their past ones. In the worst case, we might not find any useful information, which won't impact the prediction. However, we might also find some behaviors similar to what the agent is currently doing (e.g. turning in a particular way) or, in the best case, we might find patterns (e.g. agents going in near-circles, zig-zags...) that we can extend to the currently-observed situation.

In order to take this information into account for an agent a at timestp k, we keep a history of this agent's positions during h previous timesteps. For each of these timesteps $i \in [k-h, k-1]$, we record a node $n_{a,i}$ to which we associate the agent's position at that timestep as well as the change of orientation $\alpha_{a,i} = angle(v_{n_{a,i},n_{a,i-1}}, v_{n_{a,i-1},n_{a,i-2}})$. An agent a's history $\mathcal{H}$ is then the collection of these h nodes $\mathcal{H} = \{n_{a,k-h}, ..., n_{a,k-1}\}$. Then, at timestep k, we look for the most similar timestep in the past $i^* = \text{argmin}_{i \in [k-h,k-1]} |\alpha_{a,i} - \alpha_{a,k}|$ (note that is is possible to use more than one node to check the similarity). If $|\alpha_{a,i^*} - \alpha_{a,k}|$ is under a certain threshold β, the situation is similar enough and can be used. In that case, we link the nodes $n_{a,k}$ and n_{a,i^*} (effectively making timestep i^*'s future become timestep k's future) and the resulting, updated graph (as seen on Figure B.1(c)) can be used as input for the space-shape-warping algorithm.

Bibliography

[ABV14] A. Alahi, M. Bierlaire, and P. Vandergheynst. Robust real-time pedestrians detection in urban environments with low-resolution cameras. *Transportation Research Part C: Emerging Technologies*, 39(0):113 – 128, 2014. 85

[ACC+13] Alessandro Attanasi, Andrea Cavagna, Lorenzo Del Castello, Irene Giardina, Stefania Melillo, Leonardo Parisi, Oliver Pohl, Bruno Rossaro, Edward Shen, Edmondo Silvestri, and Massimiliano Viale. Collective behaviour without collective order in wild swarms of midges. *Cornell University Library*, arXiv:1307.5631, 2013. 75, 82

[AMBT06] Gianluca Antonini, Santiago Venegas Martinez, Michel Bierlaire, and Jean Philippe Thiran. Behavioral priors for detection and tracking of pedestrians in video sequences. *International Journal of Computer Vision*, 69(2):159–180, 2006. 89

[AS08] Saad Ali and Mubarak Shah. Floor fields for tracking in high density crowd scenes. In *ECCV*, pages 1–14. 2008. 89

[BC13] Abbas Ali Butt and Robert T Collins. Multi-target tracking by lagrangian relaxation to min-cost network flow. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1846–1853. IEEE, 2013. 87

[BCM12] Loris Bazzani, Marco Cristani, and Vittorio Murino. Decentralized particle filter for joint individual-group tracking. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1886–1893. IEEE, 2012. 88

[BFF09] Jerome Berclaz, Francois Fleuret, and Pascal Fua. Multiple object tracking using flow linear programming. In *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, pages 1–8. IEEE, 2009. 88

[BKBS13] James Brecht, Theodore Kolokolnikov, AndreaL. Bertozzi, and Hui Sun. Swarming on random graphs. *Journal of Statistical Physics*, 151(1-2):150–173, 2013. 74

[BM09] Laurence Boudet and Sophie Midenet. Pedestrian crossing detection based on evidential fusion of video-sensors. *Transportation Research Part C:*

Emerging Technologies, 17(5):484 – 497, 2009. Artificial Intelligence in Transportation Analysis: Approaches, Methods, and Applications. 85

[BM14] Aniket Bera and Dinesh Manocha. Realtime multilevel crowd tracking using reciprocal velocity obstacles. In *Proceedings of Conference on Pattern Recognition, Sweden*, 2014. 88, 89, 91, 97

[BMD+13] Sachit Butail, Nicholas Manoukis, Moussa Diallo, José M. C. Ribeiro, and Derek Paley. The dance of male anopheles gambiae in wild mating swarms. *J. Med. Entomol.*, 50(3):552–559, 2013. 75

[BMP11] Martin Burger, Peter Markowich, and Jan-Frederik Pietschmann. Continuous limit of a crowd motion and herding model: analysis and numerical simulations. *Kinet. Relat. Models*, 4(4):1025–1047, 2011. 19

[BRL+09] Michael D Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Robust tracking-by-detection using a detector confidence particle filter. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1515–1522. IEEE, 2009. 88

[BSC+06] J. Buhl, D. J. T. Sumpter, I. D. Couzin, J. J. Hale, E. Despland, E. R. Miller, and S. J. Simpson. From disorder to order in marching locusts. *Science*, 312(5778):1402–1406, 2006. 74

[BYB11] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Robust object tracking with online multiple instance learning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(8):1619–1632, 2011. 87

[CARH13] Julien Cividini, Cecile Appert-Rolland, and Hendrik-Jan Hilhorst. Diagonal patterns and chevron effect in intersecting traffic flows. *EPL (Europhysics Letters)*, 102(2):20002, 2013. 14, 55

[CC07] Nicolas Courty and Thomas Corpetti. Crowd motion capture. *Computer Animation and Virtual Worlds*, 18(4-5):361–370, 2007. 16, 17

[CC14] P. Charalambous and Y. Chrysanthou. The pag crowd: A graph based approach for efficient data-driven crowd simulation. *Computer Graphics Forum*, 33(8):95–108, 2014. 15, 16

[CDF+03] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2003. 71, 79

[Che04] S. Chenney. Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 233–242, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association. 16, 17

[CkJ+02] Iain D. Couzin, Jens krause, Richard James, Graeme D. Ruxton, and Nigel R. Franks. Collective memory and spatial sorting in animal groups. *J. Theory. Biol*, 218:1–11, 2002. 73

[CM12] Sean Curtis and Dinesh Manocha. Pedestrian simulation using geometric reasoning in velocity space. In *Pedestrian and Evacuation Dynamics (PEDS)*, 2012. 29, ii

[CSC09] Ujjal Chattaraj, Armin Seyfried, and Partha Chakroborty. Comparison of pedestrian fundamental diagram across cultures. In *Advances in Complex Systems*, pages 393–405, 2009. 25, 33, 35, ii

[DARM+13] Pierre Degond, Cécile Appert-Rolland, Mehdi Moussaid, Julien Pettré, and Guy Theraulaz. A hierarchy of heuristic-based models of crowd dynamics. *Journal of Statistical Physics*, 152(6):1033–1068, 2013. 19

[DARPT13] Pierre Degond, Cécile Appert-Rolland, Julien Pettré, and Guy Theraulaz. Vision-based macroscopic pedestrian models. *arXiv preprint arXiv:1307.1953*, 2013. 19

[DCBC06] M. R. D'Orsogna, Y. L. Chuang, A. L. Bertozzi, and L. S. Chayes. Self-propelled particles with soft-core interactions: Patterns, stability, and collapse. *Phys. Rev. Lett.*, 96:104302, 2006. 73

[EG09] Markus Enzweiler and Dariu M Gavrila. Monocular pedestrian detection: Survey and experiments. *PAMI*, pages 2179–2195, 2009. 86

[Feu00] Franck Feurtey. Simulating the collision avoidance behavior of pedestrians. Master's thesis, Department of Electronic Engineering, University of Tokyo, 2000. 10, 46

[FGLO99] G. Flierl, D. Grünbaum, S. Levins, and D. Olson. From individuals to aggregations: the interplay between behavior and physics. *Journal of Theoretical Biology*, 196(4):397–454, 1999. 73

[Fri37] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. v

[GCK+09] Stephen. J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: Highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 177–187, New York, NY, USA, 2009. ACM. 11, 23, 46

[GCLM12] Stephen J. Guy, Sean Curtis, Ming C. Lin, and Dinesh Manocha. Least-effort trajectories lead to emergent crowd behaviors. *Phys. Rev. E*, 85:016110, Jan 2012. 11, 23, 24, 46

[GGB06] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *BMVC*, volume 1, page 6, 2006. 87

[GNL13] Abhinav Golas, Rahul Narain, and Ming Lin. Hybrid long-range collision avoidance for crowd simulation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pages 29–36, New York, NY, USA, 2013. ACM. 13, 46

[GvdBL+12] Stephen J. Guy, Jur van den Berg, Wenxi Liu, Rynson Lau, Ming C. Lin, and Dinesh Manocha. A statistical similarity measure for aggregate crowd dynamics. *ACM Trans. Graph.*, 31(6):190:1–190:11, November 2012. 12, 25, 46

[HF09] Rob Hess and Alan Fern. Discriminatively trained particle filters for complex multi-object tracking. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 240–247. IEEE, 2009. 88

[HFV00] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000. 8, 23, 24, 27, 46

[HHD14] Qing He, K. Larry Head, and Jun Ding. Multi-modal traffic signal control with priority, signal actuation and coordination. *Transportation Research Part C: Emerging Technologies*, 46(0):65 – 82, 2014. 85

[HM95] Dirk Helbing and Péter Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995. 8, 10, 23, 24, 44, 46, 64, 73, 78, 96

[HO96] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 312–317, May 1996. 30, iv

[Hol92] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–72, 1992. 30

[HST11] Sam Hare, Amir Saffari, and Philip HS Torr. Struck: Structured output tracking with kernels. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 263–270. IEEE, 2011. 87

[Hug03] Roger L Hughes. The flow of human crowds. *Annual review of fluid mechanics*, 35(1):169–182, 2003. 18, 73

[JB12] Zhixing Jin and Bir Bhanu. Single camera multi-person tracking based on crowd simulation. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3660–3663. IEEE, 2012. 89

[JCP+10] E. Ju, M.G. Choi, M. Park, J. Lee, K.H. Lee, and S. Takahashi. Morphable crowds. *ACM Trans. Graph.*, 29:140:1–140:10, 2010. 15, 16, 45

[JFL07] Hao Jiang, Sidney Fels, and James J Little. A linear programming approach for multiple object tracking. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007. 88

[JHS07] Anders Johansson, Dirk Helbing, and Pradyumn K Shukla. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. *Advances in complex systems*, 10(supp02):271–288, 2007. 9

[KBD04] Zia Khan, Tucker Balch, and Frank Dellaert. An mcmc-based particle filter for tracking multiple interacting targets. In *Computer Vision-ECCV 2004*, pages 279–290. Springer, 2004. 88

[KGL+14] Sujeong Kim, Stephen J. Guy, Wenxi Liu, David Wilkie, Rynson W.H. Lau, Ming C. Lin, and Dinesh Manocha. Brvo: Predicting pedestrian trajectories using velocity-space reasoning. *The International Journal of Robotics Research*, 2014. 13, 46

[KHBO09] Ioannis Karamouzas, Peter Heil, Pascal Beek, and Mark H. Overmars. A predictive collision avoidance model for pedestrian simulation. In *Proceedings of the 2Nd International Workshop on Motion in Games*, MIG '09, pages 41–52, Berlin, Heidelberg, 2009. Springer-Verlag. 9, 10, 25, 46, 73

[KHHL12] Manmyung Kim, Youngseok Hwang, Kyunglyul Hyun, and Jehee Lee. Tiling motion patches. In *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*, pages 117–126. Eurographics Association, 2012. 17

[KJV83] Scott Kirkpatrick, D. Gelatt Jr., and Mario P Vecchi. Optimization by simulated annealing. *science*, 220:671–680, 1983. 30

[KLLT08] Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Shigeo Takahashi. Group motion editing. In *Proc. of the 2008 ACM SIGGRAPH Conference*, SIGGRAPH '08, pages 1–8, 2008. 73

[KN12] Louis Kratz and Ko Nishino. Going with the flow: pedestrian efficiency in crowded scenes. In *ECCV*, pages 558–572. 2012. 89

[KO13] Douglas Kelley and Nicholas Ouellette. Emergent dynamics of laboratory insect swarms. *Scientific Reports*, 3(1073), 2013. 75

[KR08] Bernardin Keni and Stiefelhagen Rainer. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008, 2008. 98

[KS08] Tobias Kretz and Michael Schreckenberg. The f.a.s.t.-model. *CoRR*, abs/0804.1893, 2008. 13, 45

[KSG14] Ioannis Karamouzas, Brian Skinner, and Stephen J. Guy. Universal power law governing pedestrian interactions. *Phys. Rev. Lett.*, 113:238701, Dec 2014. 12, 25, 46

[KSHF09] Mubbasir Kapadia, Shawn Singh, William Hewlett, and Petros Faloutsos. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 215–223, New York, NY, USA, 2009. ACM. 12, 46

[KWS+11] Mubbasir Kapadia, Matt Wang, Shawn Singh, Glenn Reinman, and Petros Faloutsos. Scenario space: characterizing coverage, quality, and failure of steering algorithms. In *Proc. of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 53–62, New York, NY, USA, 2011. ACM. 24, 39

[LCHL07] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 109–118. Eurographics Association, 2007. 15

[LCL07] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. *Computer Graphics Forum*, 26(3):655–664, 2007. 14, 15, 45

[LCLM14] Wenxi Liu, Antoni B. Chan, Rynson W. H. Lau, and Dinesh Manocha. Leveraging long-term predictions and online-learning in agent-based multiple person tracking. 2014. 88, 89

[LCSCO09] Alon Lerner, Yiorgos Chrysanthou, Ariel Shamir, and Daniel Cohen-Or. Data driven evaluation of crowds. In *Proc. of the 2nd International Workshop on Motion in Games*, MIG '09, pages 75–83, Berlin, Heidelberg, 2009. Springer-Verlag. 24

[LCSCO10] Alon Lerner, Yiorgos Chrysanthou, Ariel Shamir, and Daniel Cohen-Or. Context-dependent crowd evaluation. In *Computer Graphics Forum*, volume 29, pages 2197–2206. Wiley Online Library, 2010. 24

[LD04] Fabrice Lamarche and Stéphane Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. In *Computer Graphics Forum*, volume 23, pages 509–518. Wiley Online Library, 2004. 9

[LWPCL15] Weizi Li, David Wolinski, Julien Pettré, and Ming C. Lin. Biologically-inspired visual simulation of insect swarms. *Computer Graphics Forum*, 34(2):425–434, 2015. 76

[MEK99] Alexander Mogilner and Leah Edelstein-Keshet. A non-local model for a swarm. *Journal of Mathematical Biology*, 38(6):534–570, 1999. 74

[MHT11] Mehdi Moussaïd, Dirk Helbing, and Guy Theraulaz. How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences*, 108(17):6884–6888, 2011. 12, 19

[MTPS04] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. In *ACM Transactions On Graphics (TOG)*, volume 23, pages 449–456. ACM, 2004. 31

[NGCL09] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C. Lin. Aggregate dynamics for dense crowd simulation. *ACM Transactions on Graphics*, 28:122:1–122:8, 2009. 18, 19, 45, 73

[NWK90] J. Neter, W. Wasserman, and M.H. Kutner. *Applied Linear Statistical Models, 3rd edition*. Burr Ridge, 1990. 75

[Oku86] Akira Okubo. Dynamical aspects of animal grouping: Swarms, schools, flocks, and herds. *Adv. Biophys.*, 22:1–94, 1986. 71, 75

[OMCP12] Anne-Hélène Olivier, Antoine Marin, Armel Crétual, and Julien Pettré. Minimal predicted distance: A common metric for collision avoidance during pairwise interactions between walkers. *Gait & posture*, 36(3):399–404, 2012. 12, 44, 46

[OPOD10] J. Ondřej, J. Pettré, A.-H. Olivier, and S. Donikian. A synthetic-vision based steering approach for crowd simulation. *ACM Trans. Graph.*, 29(4):123:1–123:9, July 2010. 12, 19, 23, 24, 25, 28, 46, 73

[OTDF+04] Kenji Okuma, Ali Taleghani, Nando De Freitas, James J Little, and David G Lowe. A boosted particle filter: Multitarget detection and tracking. In *Computer Vision-ECCV 2004*, pages 28–39. Springer, 2004. 88

[PCBS11] Matthias Plaue, Minjie Chen, Günter Bärwolff, and Hartmut Schwandt. Trajectory extraction and density analysis of intersecting pedestrian flows from video recordings. In *Proc. of the 2011 ISPRS conference on Photogrammetric image analysis*, PIA'11, pages 285–296, Berlin, Heidelberg, 2011. Springer-Verlag. 33

[PESVG09] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 261–268, Sept 2009. 9, 10, 24, 25, 46, 89, 97

[PEVG10] Stefano Pellegrini, Andreas Ess, and Luc Van Gool. Improving data association by joint modeling of pedestrian trajectories and groupings. In *Computer Vision–ECCV 2010*, pages 452–465. Springer, 2010. 97, 98

[PKB07] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007. 31

[PKO14] James Puckett, Douglas Kelley, and Nicholas Ouellette. Searching for effective forces in laboratory insect swarms. *Scientific Reports*, 4(4766), 2014. 75, 79

[POO+09] 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. 10, 11, 22, 24, 25, 28, 31, 46

[PPD07] Sébastien Paris, Julien Pettré, and Stéphane Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Computer Graphics Forum*, 26(3):665–674, 2007. 10, 11, 23, 24, 25, 44, 46

[PSH+06] A.G.A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and Wensheng Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 666–673, June 2006. 88

[PvdBC+11] Sachin Patil, Jur van den Berg, Sean Curtis, Ming C. Lin, and Dinesh Manocha. Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics*, 17:244–254, February 2011. 17, 73, 80

[Rey87] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Computer Graphics*, 21(4):25–34, 1987. 8, 46, 64, 73, 74, 78

[Rey99] C.W. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference 1999*, pages 763–782, 1999. 9, 10, 27, 46, 97

[RS11] Mikel Rodriguez and Josef et al. Sivic. Data-driven crowd analysis in videos. In *ICCV*, pages 1235–1242, 2011. 89

[SBR14] Stefan Seer, Norbert Brändle, and Carlo Ratti. Kinects and human kinetics: A new approach for studying pedestrian behavior. *Transportation Research Part C: Emerging Technologies*, 48(0):212 – 228, 2014. 85

[Sch01] Andreas Schadschneider. Cellular automaton approach to pedestrian dynamics - theory. page 11, 2001. 13, 45

[SESG08] Jessica Strefler, Udo Erdmann, and Lutz Schimansky-Geier. Swarming in three dimensions. *Phys. Rev. E*, 78(3):031927, 2008. 73

[SF15] Sebastian Seriani and Rodrigo Fernandez. Pedestrian traffic management of boarding and alighting in metro stations. *Transportation Research Part C: Emerging Technologies*, 53(0):76 – 92, 2015. 85

[Sha] Shark. `http://image.diku.dk/shark/sphinx_pages/build/html/index.html`. iv

[SHN12] Pramod Sharma, Chang Huang, and Ram Nevatia. Unsupervised incremental learning for improved object detection in a video. In *CVPR*, pages 3298–3305, 2012. 87

[SKFR09] Shawn Singh, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinman. Steerbench: a benchmark suite for evaluating steering behaviors. *Computer Animation and Virtual Worlds*, 20(5-6):533–548, 2009. 24, 25, 39

[Sum10] DJT Sumpter. *Collective Animal behavior*. Princeton University Press, 2010. 71

[TBL06] Chad Topaz, Andrea Bertozzi, and Mark Lewis. A nonlocal continuum model for biological aggregations. *Bulletin of Mathematical Biology*, 68(7):1601–1623, 2006. 74

[TCP06] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. In *SIGGRAPH '06*, pages 1160–1168, New York, NY, USA, 2006. ACM. 18, 45, 73

[TYK+09] Shigeo Takahashi, Kenichi Yoshida, Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Sung Yong Shin. Spectral-based group formation control. *Computer Graphics Forum*, 28(2):639–648, 2009. 73

[VCBJ+95] Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet. Novel type of phase transition in a system of self-driven particles. *Physical Review Letters*, 75(6):1226–1229, 1995. 8, 73, 75, 82

[vdBLM08] J. van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation*, pages 1928–1935, May 2008. 10, 11, 13, 23, 27, 44, 46, 64, 73, 89, 95, 96, 97

[vdBSGM11] J. van den Berg, J. Snape, S.J. Guy, and D. Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3475–3482, May 2011. 13, 46, 78

[WGO+14] David Wolinski, Stephen Guy, Anne-Helene Olivier, Ming Lin, Dinesh Manocha, and Julien Pettré. Parameter Estimation and Comparative Evaluation of Crowd Simulations. *Computer Graphics Forum*, 33(2):303–312, 2014. 12, 46, 64, 65

[Wil45] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945. v

[WJDZ14] Xinjie Wang, Xiaogang Jin, Zhigang Deng, and Linling Zhou. Inherent noise-aware insect swarm simulation. *Computer Graphics Forum*, 2014. viii, 73, 74

[WLY13] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. pages 2411–2418, 2013. 86

[YBOB11] Kota Yamaguchi, Alexander C Berg, Luis E Ortiz, and Tamara L Berg. Who are you with and where are you going? In *CVPR*, pages 1345–1352, 2011. 89, 97

[YEE+09] Christian A. Yates, Radek Erban, Carlos Escudero, Iain D. Couzin, Jerome Buhl, Ioannis G. Kevrekidis, Philip K. Maini, and David J. T.

Sumpter. Inherent noise can facilitate coherence in collective swarm motion. *Proceedings of the National Academy of Sciences*, 106(14):5464–5469, 2009. 74

[YJS06] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm Computing Surveys (CSUR)*, 2006. 86

[YMPT09] Barbara Yersin, Jonathan Maïm, Julien Pettré, and Daniel Thalmann. Crowd patches: populating large-scale virtual environments for real-time applications. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 207–214. ACM, 2009. 17, 18

[YN12] Bo Yang and Ram Nevatia. Multi-target tracking by online learning of non-linear motion patterns and robust appearance models. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1918–1925. IEEE, 2012. 89

[ZGM12] Xuemei Zhao, Dian Gong, and Gérard Medioni. Tracking using motion patterns for very crowded scenes. In *ECCV*, pages 315–328. 2012. 89

[ZKSS12] J Zhang, W Klingsch, A Schadschneider, and A Seyfried. Ordering in bidirectional pedestrian flows and its influence on the fundamental diagram. *Journal of Statistical Mechanics: Theory and Experiment*, 2012(02):P02002, 2012. v

[ZLN08] Li Zhang, Yuan Li, and Ramakant Nevatia. Global data association for multi-object tracking using network flows. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 87

[ZMMS15] Sohail Zangenehpour, Luis F. Miranda-Moreno, and Nicolas Saunier. Automated classification based on video data at intersections with heavy pedestrian and bicycle traffic: Methodology and application. *Transportation Research Part C: Emerging Technologies*, 56(0):161 – 176, 2015. 85

[ZTW12] Bolei Zhou, Xiaoou Tang, and Xiaogang Wang. Coherent filtering: detecting coherent motions from crowd clutters. In *Computer Vision–ECCV 2012*, pages 857–871. Springer, 2012. 55

[ZZY12] Kaihua Zhang, Lei Zhang, and Ming-Hsuan Yang. Real-time compressive tracking. In *ECCV*, pages 864–877. 2012. 87