Tel-Aviv University

The Raymond and Beverly Sackler Faculty of Exact Sciences

The Blavatnik School of Computer Science

Fitting Behaviors to Pedestrian Simulations

Thesis submitted toward an M.Sc. degree

by Eitan Fitusi

Under the supervision of Prof. Daniel Cohen-Or

June 2009
Abstract

This paper presents a data-driven approach for fitting behaviors to simulated pedestrian crowds. This is a general method that receives as input trajectories of pedestrians, or other similar crowds, generated by any available means, such as crowd simulators, manually defined trajectories or real tracked data. The method fits each pedestrian with action-tags that complement its behavior, according to its stimuli. A stimulus can be another pedestrian, a point of interest, or its recent behavior. Animating the agents according to the tagged trajectories significantly enhances the impression that they are not merely walking, but are behaving and interacting with one another and with the environment. To put in layman's terms, the fitted behaviors add a realistic quality to the crowds that makes them more believable.

Our approach is data-driven. Based on a video of a real crowd, we define examples that we use to estimate the probability that a simulated agent will perform an action given its stimuli. In a sense, the agents are "imitating" the behaviors which were observed in the video. An example stores a specific configuration of stimuli that motivated the performance of an action. The relative importance of each stimulus is defined by a stimuli-map. The stimuli required to perform an action is defined by a validity-map. All of this information is stored inside an action-graph, which can be conceived as a probabilistic state machine for assigning action-tags. At run time, given an agents stimuli configuration, the importance of each stimulus within the configuration is determined and compared to the stimuli observed in the examples. Thus, an approximation of the probability for the agent to perform each action is obtained and an action-tag is assigned accordingly. We applied our technique to crowds generated by different simulators, such as flocks, rule-based and example-based simulators, enriching their behaviors with various mundane actions and enhancing the simulations natural appearance.
# Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHAPTER 1 - INTRODUCTION</strong></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td><strong>CHAPTER 2 - RELATED WORK</strong></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2.1 Data-Driven Approaches</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2.2 Cognitive Models</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>2.3 Rules and Animation</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td><strong>CHAPTER 3 - PREPROCESSING</strong></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>3.2 Collecting graph information</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>3.3 Building the Graph</td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>3.4 Examples</td>
<td></td>
<td>27</td>
</tr>
<tr>
<td>3.5 Stimuli-Maps</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>3.6 Validity-Maps</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>3.7 Points of Influence</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>3.8 Preprocessing Summary</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td><strong>CHAPTER 4 - FITTING BEHAVIORS</strong></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>4.2 Similarity Function</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>4.3 Similarity over Time</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>4.4 Example Weight and Cluster Weight</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>4.5 Online Behavior Fitting</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td><strong>CHAPTER 5 - RESULTS AND ANALYSIS</strong></td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>5.1 Implementation</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>5.2 Data Preparation</td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>5.3 Run-time Performance</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>5.4 Evaluation of Fitted Behaviors</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>5.5 Educating Your Crowd</td>
<td></td>
<td>51</td>
</tr>
<tr>
<td><strong>CHAPTER 6 - CONCLUSIONS</strong></td>
<td></td>
<td>53</td>
</tr>
</tbody>
</table>
List of Figures

1.2 Simulated people behaviors. 5

2.1 Two characters avoiding collision 9
2.2 Tracked trajectories from a video 10
2.3 Group behavior models learned from crowd videos 11
2.4 Crowds by Example Overview. 12
2.5 Cognitive modeling - different layer of simulation. 14
2.6 Artificial fishes in their physics-based world 15
2.7 An Informed Environment Graph 17
2.8 States behavior graph 18
2.9 Yu et al Sub Network example 19
2.10 A simple motion graph 21

3.1 Preprocessing overview 22
3.2 Run time overview 23
3.3 Three layers of information 24
3.4 The behavior-graph 26
3.5 Time-line with action tags 27
3.6 Region of influence over the marked video 28
3.7 Stimuli example 29
3.8 Stimuli-maps 30
3.9 The Gaussian filter 31
3.10 Validity-maps 32
3.11 A point of influence 33

4.1 Weights over time 39

5.1 Screenshot of the user interface 42
5.2 Visualization of the probability function 43
5.3 A Reynolds flocking simulation and a Rule-based simulation 48
5.4 Example-based crowd Simulation with behaviors. 52
Chapter 1

Introduction

In recent years, the quality of computer generated crowds has risen to a degree where they are commonly used in virtual environment applications such as motion pictures and computer games. Hordes of fighting soldiers, people fleeing a monster or cheering their favorite team can be found in many films and games. However, simple pedestrian crowds are not as common. Although the quality of animation, rendering and crowd simulation has improved tremendously, the generation of a believable pedestrian crowd remains a challenge. One of the reasons for this is that we, as pedestrians, are used to seeing such crowds in our day-to-day lives. Any peculiar behavior on the part of the simulated agents decreases the overall realism of the crowd.

![Figure 1.1: Simulated people do more than just walk; they interact with one another and with the environment.](image)

Usually, pedestrian crowd simulation techniques focus on generating realistic crowds at the trajectory level. However, people do more than just walk. They talk to one another, glance at the surrounding environment, talk on the phone, scratch their heads or perform other similar actions, as seen in Figure 1.1. The absence of these mundane actions reduces the credibility of the simulated crowd. On the other hand, adding actions in inappropriate times or situations may seem odd and unrealistic. In this work we show how these actions can be learned from captured data and added to simulated agents trajectories. When the crowd is animated according to the fitted behaviors, the
aggregate effect of the performed actions enhances the impression that the agents are behaving, interacting with one another and with the environment.

Behavior can be defined as the aggregate actions and reactions of a person to internal and external stimuli. The internal stimulus emanates, among other things, from one's personality and state of mind. The external stimuli can emanate from anything in the surrounding environment. In this work we consider the people and object in the vicinity of the agent as external stimuli. Cognitive modeling methods can take in account both internal and external stimuli and have been shown to perform well on individual agents. However, they can be quite complex and computationally intensive, therefore not the most appropriate for a crowd.

When we look at an arbitrary real world pedestrian crowd, all we know about each person is derived from what we observe during the few moments that they are in our view. Their interactions, movements within the environment and consistency of actions are the factors by which we judge them. Therefore, in principle, a system based only on the observed stimuli of an agent should be able to recreate a realistic crowd. Although reactive rule-based systems have already been proposed for adding reactions of agents to environmental stimuli, the interactions within a crowd are so rich and varied that great skills and effort are required to define the rules that will faithfully capture them.

Actions and trajectories have a mutual effect on each other. Some actions are more likely to occur along certain trajectories while some trajectories are more likely to be taken when certain actions are performed. Despite the strong link between the two, numerous methods exist for simulating convincing trajectories without taking actions into account. We propose a technique that enhances the realism of crowds simulated with these methods by fitting the agents with secondary actions. Since the trajectories are already defined, the actions are affected by them; however, they do not affect them.

The proposed technique is data-driven. From a video of a real crowd, example configurations of observed stimuli and the actions motivated by them are defined. The examples form a training set out of which a graph is derived. In the graph, nodes hold
the observed actions and edges the example stimuli configurations that form the conditions leading from one action to another

At run time, given the configuration of a simulated agent, it is matched with one of the configurations in the training set, weighted, and validated by stimuli and validity maps. These maps, together with the global distribution of the examples, capture the essence of crowd behavior. Although agents are assigned behaviors on an individual basis, the behaviors admit both local and global characteristics. When looking at individual agents they seem to act naturally, while at the same time, the crowd as a whole presents consistent pedestrian behaviors.

During a simulation each agent is asked three questions: When should it perform an action? Which action should it perform? And how long should the action last? Our method answers these questions based on the stimuli surrounding the agent and their similarity to stimuli which motivated an action in the real world.

We employ our method to annotate the trajectories of agents from various sources: real-data different from the input, an example-based simulation, a rule-based simulation and a flocking simulation. In all cases the agents appear to interact with each other and the environment in a believable manner, thereby making the simulated crowd appear more natural.
Chapter 2

Related Work

Crowd simulation research is an active theme in a number of fields, such as computer graphics, sociology and robotics. There are several approaches for simulating crowds. Some derive ideas from fluid mechanics while others use particles or social forces. The most popular approach is the rule-based approach. Where each agent perceives the environment and nearby agents, and based on a set of relatively simple rules decides its actions. This model is popular since the integration of individual rule-based actions becomes a collective behavior of enormous complexity. The downside of it is that the crowd behavior is affected indirectly, in an unintuitive manner, by tuning parameters or changing rules. For this reason, it is difficult to generate the desired behaviors for the entire crowd using the rule-based approach.

Although methods for animating a single human character advanced greatly, automatically animating a believable pedestrian crowd remains a challenge. There are many ways to create a simulated crowd, but much fewer ones to create natural behaviors for this crowd. In this chapter we address the techniques for generating crowd behaviors, and data driven approaches used to simulate crowds and are related to this work.

Most of the crowd simulation literature focuses on the navigational aspect. Our work focuses on the behavioral aspect and assumes that some system exists to provide us with agent trajectories.

2.1 Data-Driven Approaches

In recent years, data-driven approaches, which are common in many fields, have been applied to crowd simulations. For example, Metoyer and Hodgins [7] allow the user
to define specific examples of behaviors, which are actually pedestrian trajectories, and then generating reactive paths that follow these examples.

![Figure 2.1: This scene shows two characters initially on a collision course. In the simulation, one pedestrian chose to go-around-right in order to avoid the collision. This decision resulted in a trajectory different from its desired path shown with the arrow. The path adjustment is subtle but effective in avoiding the collision.](image)

The suggested system allows novice users to control the path planning. Instead of specifically defining trajectories for each person, the user marks boundaries, like a wall or a fence, obstacles in the given environment, like a single tree, and the general path that he wants the pedestrians to follow. Given this information, the system creates the trajectories and calculates most of the character motions automatically. The user can affect the results by directing the characters with navigation primitives. These primitives are used to build a model of the desired reactive behavior, which is used by the system under similar circumstances. The reactive navigation, user supplied paths and learned model are combined to produce the final results. (See Figure 2.1)

Musse et al. [8] define the navigational model based on real trajectories, which were extracted automatically from an input video of a real crowd. The trajectories are extracted automatically using computer vision algorithms that track the people in the video, as seen in Figure 2.2. In order to improve the quality of the extracted trajectories, or add new ones, the user can edit the result trajectories.
The trajectories are then clustered together and from each cluster a velocity field is generated. The fields are used as stimuli that guide the virtual humans during the simulation. A physics based model is used to simulate the virtual human. It tries to simulate trajectories similar to the input trajectories, while avoiding collisions, by guiding the agents through the velocity fields. In order to handle possible collisions, attraction and repulsion forces are used.

In more recent works (Lee et al. [5], Lerner et al. [6]), trajectories learned from videos of crowds are stored along with some representation of the stimuli that affected them. During a simulation, agents match their stimuli to the ones stored in the database, and navigate accordingly.

Lee et al. [5] define an agent based simulation where the agents learn from examples how to navigate. Here, trajectories are extracted from a video of a crowd and used as examples for the simulation. The trajectories are tracked using a semi-automatic system that allows the user to correct the trajectories extracted by the automatic algorithm.
From the trajectories a database of state-action pairs are defined. A state, which can be seen as stimuli, that stored the configuration, and an action that follow this state. The agent decision model is in charge of guiding the agent through the simulation. The behavior model consists of a set of low-level action models and when to transition between them. Each low-level model describes a primitive action that can be represented by a simple learning model. The action model can be considered as a function that takes the state of an agent as input and produces a desired action for the agent at the next time instance. Given a novel state $s$ observed in the simulation, the action model has to define an action with respect to the training data. (See figure 2.3)

To store states as fixed and relatively low dimensional vectors the space around the agent is divided into a set of eight radial regions. Within each radial region, the distance to the nearest agent is a determining factor. In order to reflect the temporal change of the neighborhood formation, the state vector includes feature values from two successive (the current and the previous) frames. The state vector also includes information about speed and direction, comparing a state is therefore simply a matter of comparing the state vectors.
Lee et al. simulate believable crowd behaviors by learning navigation and interaction behavior models from a video. But this behavior is context depend a person may talk with a neighbor only in the correct context. In any given moment an agent either navigates through the environment or interacts with other agents, however it cannot do both. In our work the agents can walk and talk at the same time and can easily shift from one behavior to another if the appropriate conditions exist.

Lerner et al. [6] also simulate trajectories based on real world examples. Examples store a short trajectory segment and a representation of the stimuli that motivated it. During a simulation agents search for examples whose stimuli match their own, and copy the example's trajectories. Here the trajectories are tracked manually through a simple, user friendly system. During a simulation, autonomous agents search the example database for examples that closely match the situation that they are facing. The matching example points to a trajectory segment take by a real person under similar circumstances. The trajectories are copied to the simulated agents, resulting in seemingly natural behaviors, see Figure 2.4.

Figure 2.4: Crowds by Example Overview. The top row depicts the construction of the database, which takes place during preprocessing: the input video is manually tracked generating a set of trajectories. These are encoded as examples and stored in the database. At run-time, bottom row, the trajectories of the agents are synthesized individually by encoding their surroundings (forming a query) and searching the database for a similar example. The trajectory from the example is copied over to the simulated agent.
An example consists of two components. The first component is a segment of the trajectory (path) taken by a person in the video. The second component of an example is the configuration of the factors which might have influenced the trajectory of the subject. This might be another individual or some dynamic or static geometry in the scene. The work focuses on the steering aspects of the agent's behavior, aiming to synthesize a realistic trajectory for each individual in the crowd that will produce believable crowd behavior.

Ashida et al. [1] model subconscious upper body actions based on a statistical distribution extracted from a video. The added action includes hand gestures, head and torso movements. The statistical distribution of the actions is obtained by studying a video of people as they walked along a section of sidewalk. Counting the occurrences of different actions defines a distribution, which represents the frequency in which each action is performed by an average person. Here the authors account also for the emotional state of an agent and therefore alter the distribution based on the agent’s mood. Subconscious actions are affected by a person's mood or internal state. The authors model this by a set of parameters; for example, a very tired or very sad character would be unlikely to perform actions in the same manner as a very energetic character would. In order to preserve the consistency of the characters, they linked high level parameters, such as tiredness and sadness, with the low level animation parameters such as position and velocity. Similarly, it seems that some character traits influence the probability of a character performing some subconscious actions. For example, a fearful character is less likely to perform behaviors such as scratching.

This work tries to reflect a person's feelings or mood. It focuses on a single person and tries to mimic the internal process of the subject person in terms of upper body movements. Ours is a more general approach which is less focused on the continues behavior of a single person, but rather tries to present an overall believable crowd behavior.
Cognitive models reproduce the cognitive decision making mechanism that eventually leads to a specific behavior. Cognitive models go beyond behavioral models and govern what a character knows, how that knowledge is acquired, and how it can be used to plan actions, as explained in Funge et al. [3] (see Figure 2.5).

Funge et al. present simulated agents that perceive the environment and learn from it which is the most suitable behavior to choose out of a predefined set.

They focus on two related sub-tasks: domain knowledge specification and character direction. Domain knowledge specification involves administering knowledge to the character about its world and how that world can change. Character direction involves instructing the character to try and behave in a certain way within its world in order to achieve specific goals.

The CML language is used to create the guidelines for a character with domain knowledge and preconditions, so the character can focus its behavior on certain goals. This approach allows behaviors to be specified more naturally and intuitively. The CML, can imbue a character with domain knowledge, specified in terms of actions, their preconditions and their effects, and then direct the character’s behavior in terms of goals.
With cognitively empowered characters, the animator need only specify an outline or “sketch plan” and, through reasoning, the character will automatically work out a detailed sequence of actions satisfying the specification. The system integrates sensing into the framework, to enable the autonomous characters to generate action plans even in highly complex dynamic virtual worlds. A character can decide what to do using a simplified mental model of its world, sense the outcome, and perform follow up actions if things don’t turn out as expected.

Terzopoulos et al. [11] propose a computational framework for creating fully functional artificial fishes. Artificial fishes are autonomous agents with functional bodies controlled by brains. Their appearance, motivations, and complicated group interactions aspire to be as faithful as possible to nature’s own. The algorithm emulates not only the appearance, movement, and behavior of the individual fish, but also the complex group behaviors evident in many aquatic ecosystems.

An artificial fish is an autonomous agent situated in a simulated physical world. The agent has a body with internal muscles and fins, that make it move. It also has sensors, including eyes that can view the environment and a brain with motor, perception, behavior, and learning centers.

The brain’s perception center includes a perceptual attention mechanism which allows the artificial fish to train its sensors for specific tasks, filtering out sensory information superfluous to its current behavioral needs. For example, the artificial fish attends to sensory information about nearby food sources when foraging.
The behavior center of the artificial fish’s mind mediates between its perception system and its motor system. The intention generator combines the habits and mental state with the incoming stream of sensory information to generate dynamic goals for the fish, such as to hunt and feed on prey. It ensures that goals have some persistence by exploiting a single-item memory. Figure 2.6 shows an example from an automatically produced animation of a virtual underwater world.

Both the work of Funge et al and of Terzopoulos et al, are rather inefficient and therefore not suited for crowds. Furthermore, when striving to generate realistic behaviors, they are, like most rule-based systems, complicated to define.

Farenc et al. [2] store information within the environment which triggers agents to perform various actions. They call such an environment an informed environment. The Informed Environment is based on a hierarchical decomposition of an urban scene into Environment Entities (see Figure 2.7), which provide geometrical information as well as semantic notions, allowing a more realistic simulation of human behavior.

In order to create realistic simulations, the environment must integrate several semantic notions about specific areas such as “a sidewalk is a space dedicated to pedestrian motion”. This informed environment which is dedicated to the simulation of virtual humans provides all the necessary data to guide virtual humans with coherent behaviors. This information includes a list of objects present in this area and a list of behaviors or actions associated with this place. Knowledge about objects is used for dealing with collision avoidance or for interacting with them. Moving objects can be pedestrians, cars, buses, bicycles or similar. Such objects use certain surfaces for displacement. The surface has associated semantic information. A single environment surface can be composed of different kinds of objects such as objects associated with obstacles (trees or walls for example) and smart objects used for specific interactions (doors or escalators). A set of elementary rules can be used, with respect to various location characteristics, to define human behaviors during interactions with objects or with other humans. The urban knowledge attached to the informed environment has been defined in correlation with human perception and
analysis in the context of urban life. The “walking on a sidewalk” action involves recognition of places called “sidewalks” and the agent’s location.

![Diagram](image-url)

**Figure 2.7:** An Informed Environment- Graph representative of the hierarchy structure of the Inform Environment the same surface can belong first to a sidewalk, then to a street, then to a block and at the highest level, to the city.

This approach aims to obtain an urban model with integrated knowledge adapted to human behavior simulation. In order to structure the virtual city data, they define a hierarchical model valid for different kinds of simulation of human life (see Figure 2.7). This model is linked to methods of scene construction and usage.

One can regard the information stored within the environment as stimuli, however, they are prescribed and associated with specific objects. In our approach the stimuli are not attached to specific objects.

### 2.3 Rules and Animation

In the film industry crowds are simulated using proprietary systems, such as Massive Software™. These are rule-based systems which also have rules for performing actions that mimic interactions between agents.
There are several conflicting goals in crowd animation. Simple characters are more efficient to evaluate, but complex characters can capture more realistic crowd behaviors. Some works do not define rules explicitly. Sung et al. [10] represent a set of behaviors as a finite state machine with probabilities associated with the edges. The probabilities are updated in real-time based on behavior functions that describe the actions for each simple behavior through probability distributions.

To overcome performance obstacles Sung et al suggest a scalable approach for controlling the behaviors of agents in a crowd. This method is scalable in the sense that increasingly complex crowd behaviors can be created without a corresponding increase in the complexity of the agents. The method has a situation-based control structure. Where basic agents have very limited behaviors, but when they enter new situations, additional, situation-specific behaviors are composed on the fly to enable agents to respond appropriately, as we seen in the graph on Figure 2.8. For example, a person walking in the city will be avoiding others, trying to move towards a goal, trying to obey traffic laws, and so on. So behavior functions that describe the actions for each of these simple behaviors through probability distributions are defined. The final action selection method is created by composing distributions of different behavior functions. In this way the combination of simple functions results in more complex aggregate behaviors.

Figure 2.8: The states are organized as a graph structure and the graph is extended by adding a new sub-graph when the agent is in some situation.
The crowd is simulated on two levels. At the high level by situation-based distributed control mechanism that gives each agent in a crowd specific detail about how to react at any given moment based on its local environment and at the low level by probability scheme which computes probabilities over state transitions.

This method relies on the fact that the crowd is anonymous, In such a crowd, it does not matter who is performing which action, all that matters is that the actions seems correct. Therefore the actions of the crowd should be driven by situations and not by individuals. The viewers do not identify individual agents, but rather view the crowd as a whole, the actions of an individual matter only in its short-term contribution to the crowd's behavior, and not in their long-term planning.

Although this method represent more scalable environment then traditional rule base simulation, it is still not optimal for a large crowd, and its behavior functions is not trivial to define.

Yu et al. [12] define a decision network framework for generating behaviors which can account for uncertainties in behavior, and affect both trajectories and actions. This framework is used for advanced behavioral animation of virtual pedestrians based on hierarchical decision networks. In particular, it addresses the level of decision-making that enables the characters to interact appropriately with their perceived environment and other humans. The decision network is scalable and easy to define.

Figure 2.9: Sub Network example, this network use to interpret character B’s intended action with A and described the possible action that B will take. based on his current state: In Meet. Start Meet.
The framework combines probabilities, decisions, and graph theories for complex behavior modeling and intelligent action selection subject to internal and external factors in the presence of uncertain knowledge.

The use of decision networks provides a convenient way to control how the character makes decisions. Adjusting the conditional probabilities and the utility functions influences how decisions get made. To avoid the potential intractability of large decision networks, the behavioral model is structured as a hierarchical set of relatively small decision networks. At the lower level a smaller decision network structure is implemented for each decision item, while at higher level(s) the decision network structure at each node represents how a decision is made based on results from its children nodes.

One can start a network construction process by considering the root cause for the behavior. The additional levels of the network represent additional factors that influence the behavior and the relationships between these factors. See Figure 2.9 for a sub network example.

These self-animating pedestrians can independently assess the interrelationships among all the relevant factors to make rational decisions in the presence of uncertainty. This makes them suitable for animating the detailed behavioral interactions of small social groups, but it’s to complicate to support a massive crowd. The creation of decision networks, even with a friendly user interface, and the manipulation of the different probabilities, makes it much too complicated to use by an average animator. To facilitate the composition of long animated sequences from a set of short clips, motion graph approaches, such as Kovar et al. [4], are used.

Realistic animation of human motion is a challenging task, as people have proven to be adept at discerning the subtleties of human movement and identifying inaccuracies. Although motion capture is a reliable way of acquiring realistic human motion, it is only a technique for reproducing motion. The Motion capture data is difficult to modify, and editing techniques are reliable only for small changes to a motion. This in particular is a problem for applications that require motion to be synthesized dynamically, such as interactive environments.
Kovar et al. try to retain the realism of motion capture while giving the ability to control and direct a character. This includes characters that can perform multiple actions, and not just walking around.

A motion graph is a finite state machine where each node represents a set of animation clips and edges smooth transitions between them. Given a set of motion capture data, it compiles a data structure called a motion graph that encodes how the captured clips may be re-assembled in different ways. The motion graph is a directed graph wherein edges contain either pieces of original motion data or automatically generated transitions (see Figure 2.10). The nodes then serve as choice points where these small bits of motion join seamlessly. This method automatically detects and creates transitions between motions, so users needn't capture motions specifically designed to connect to one another. Motion can be generated simply by building walks on the graph. Motion graphs transform the motion synthesis problem into simple graph walks. So it is possible to extract graph walks that satisfy certain properties, thereby giving a control over the synthesized motions.

A motion graph is a convenient method for automatically animating characters, and is often used to animate crowds. The graph representation of transitions between actions which appears in our work is a means for assigning action-tags regardless of the method used to animate them. However, we used a motion graph to animate the crowds that appear in the accompanying video and various figures throughout the paper.
Chapter 3

Preprocessing

3.1 Overview

The focus of this work is a scheme for annotating the trajectories of simulated agents with action-tags, based on real-world examples. We assume that two separate dedicated systems exist and run in parallel to ours, one for simulating the agent’s trajectories and another for executing their animations based on the tagged trajectories.

Our approach runs in two stages. At a preprocessing stage the annotated trajectories of an input video of a crowd are analyzed and examples of stimuli configurations leading from one action to another are defined. At run-time, agents approximate the probability of performing different actions and stochastically select one. The approximation is based on the similarity between the agent’s stimuli and the stimuli stored in the examples.

Preprocessing: The input for the method is a set of annotated trajectories. The input is extracted from video data [5, 6, and 8] and trajectories that appear in it are annotated with action-tags. Objects of interest are considered as well and are annotated with the no-action tag.
The stimuli surrounding a person at a certain time motivates an action to be performed shortly after, or more precisely, a transition between actions to occur. From the annotated trajectories we define examples of such stimuli configurations, as shown in Figure 3.1.

Based on these examples, stimuli and validity-maps are constructed for each action separately. Depending on the action, some stimuli might be more important than another. This importance is captured by the stimuli-maps, which are density based influence functions. The validity-maps impose constraints over the stimuli required for performing an action, as observed in the input video. For instance, it would assure the presence of a person to the left of the subject person when performing the talk-left action.

The above information is encoded into an action-graph, where a node represents an action and a directed edge an observed transition between actions. The examples are stored on the corresponding edges, and the maps on the appropriate nodes.

**Run-time:** At run time, the valid actions that an agent might perform are determined by testing its stimuli against the validity-maps of the neighboring graph nodes. For each valid action, the most similar examples are collected using the stimuli-maps and a similarity function. The probability of performing an action is determined by the number of matching examples and their degree of similarity to the agent’s stimuli. An action is chosen accordingly and the associated action-tag is assigned to the agent, see Figure 3.2.

**Figure 3.2:** For each simulated agent a query stimuli configuration is defined. Using the maps, examples and a similarity function, the probability of performing each action is approximated and an action is chosen accordingly.
3.2 Collecting Graph Information

The input for the method contains three layers of information: the input video, the tracking information of the people and their behavior information, as shown in Figure 3.3.

![Figure 3.3: Three layers of information. The video is shown in the background. The blue curves represent the tracked trajectories of the people and the red dots show the labels that represent the beginning of the action that a person performed. The blue dots represent the behavior labels of the people at the current frame. If their trajectories are visible then they are marked in yellow.](image)

1. Video

   The basic input layer is the actual video. It shows unaware pedestrians that are behaving in a similar manner to our desired behaviors for the simulated crowd. The video should be taken from an elevated viewpoint. This layer is the basis for creating the other layers, however, once the other layers have been created, the video is discarded and is of no additional use to our system.

2. Tracking information

   The people in the video are tracked, such that for each person that appears in the video we have its position and direction in every frame in which he appears.
3. Behavior Information

The behavior information stores for each tracked trajectory tags that indicate which action was performed, when it was performed and for how long. The different action tags do not necessarily show all of the actions that appear in the video, but rather the ones that the user wishes to fit to the simulated crowd.

From this information we construct the action-graph which is a probabilistic finite automaton that provides a convenient means for fitting action-tags to simulated agents. In the graph an action is represented by a node which stores the actions’ stimuli and validity-maps. An observed transition between actions is represented by a directed edge, to which the examples representing the transition are assigned, see Figure 3.4.

Assisted by a user-friendly interface we manually track the people in a video of a crowd. The trajectory of each person is represented by a spline, to which we assign tags representing the actions that the person performs in the video. Each tag includes the type of behavior, the frame in which it begins and the length of the action.

Except for the tags, the user is not required to specify any additional information regarding the observed behaviors. There is no need to "describe" the conditions under which the action can or cannot be performed. This simplifies the tracking stage and simplifies the user's interaction with the system. Additionally, it makes the system general so that adding new behaviors only requires marking them in the input video.

The behavioral marking process requires approximately two hours for a minute of video of an average sized crowd. Obviously this can vary depending on the density of the crowd. It is important to point out that a video is marked only once and can be used indefinitely in as many simulations as needed. Additionally, no technical or creative skills are required to mark a video, unlike the definition of a rule-based system.

Theoretically, there is no limit to the amount of actions that the system can support. However, in order to achieve a believable result the marked actions should appear in the video enough times so that the algorithm can faithfully estimate their probabilities.
We marked the following actions in a video of a sparse crowd walking on the sidewalk in front of a department store:

- **A** - Talking to someone on the right side
- **B** - Talking to someone on the left side
- **C** - Looking to the right
- **D** - Looking to the left
- **E** - Looking down
- **F** - Looking back
- **G** - Pointing to the left
- **H** - Pointing to the right
- **I** - Talking on the cellular
- **J** - Arranging / Combing hair
- **K** - Looking at ones watch (checking the time)

### 3.3 Building the Graph

The graph is constructed according to the actions and transitions between actions that were observed on each of the input video's trajectories. In our construction, we consider "no-action" to be an action and create for it a node in the graph. In the
action-graph each action is represented by a node. Directed edges between nodes reflect transitions between actions that were observed in the video within a short time window. The length of the window is a predefined constant $\Delta$ which in our experiments was equal to 25 frames (1 second). If the length of an action that was performed exceeds $\Delta$ then a self referencing edge is defined for the action. See Figure 3.5.

To the edges of the graph we attach examples, which are defined in 3.4, and to the nodes we attach stimuli and validity-maps, which are defined in 3.5 and 3.6.

![Diagram](image)

**Figure 3.5:** A time-line with several action tags appears at the bottom of the image. A short time window, indicated in red, is used to define transitions between actions, according to which the graph was defined.

### 3.4 Examples

An example is defined around a person, which we term the *subject person*. It contains a representation of the internal and external stimuli that motivated the person to perform a certain action. The external stimuli are represented by the people and objects of influence in the subject person's vicinity. Examples are defined from the trajectory of each person that appears in the video, using one frame intervals.

We define a region of influence surrounding the subject person to be of an elliptical shape. It is roughly 3.5 by 7 meters large. The subject person is placed on one side of the ellipse facing its long axis, see Figure 3.6. People and objects of interest that fall within this region are considered as external stimuli. In our experiments we found this region to be adequate at capturing the external stimuli that most frequently affect ones
behavior while part of a pedestrian crowd, in an urban setting. Obviously, the size and shape of the region can be changed in order to fit any other environment or density of crowd.

![Figure 3.6](image)

Figure 3.6: Region of influence over the marked video, the subject person in Yellow, the region itself in transparent red, and the potentially influential people are marked with Orange dots.

The region of influence is used for quickly filtering out irrelevant stimuli. This does not mean that all the people or objects within the region affect ones behavior, but rather that they have more potential of doing so. This quick elimination is useful both for performance’ sake and for removing outliers that might have an effect on the output.

Each example stores both internal and external stimuli. The internal stimuli are represented by the subject person, while the external by the surrounding people and points of interest. Each stimulus stores its recent trajectory and relevant action tags.

An example \( E \) represents the stimuli configuration that motivated an action, \( A_k \), to be performed shortly after. Therefore, it accounts for a transition from the current action \( A_j \) to action \( A_k \). An example is defined with reference to a person, \( p \), denoted as the subject person, in the video at frame \( t \). The transition is represented by a pair of actions \( (A_j; A_k) \), where \( A_j \) is the action \( p \) performed at frame \( t \) and \( A_k \) the action at frame \( t+\Delta \). Note that action \( A_k \) can be no-action or the same action as \( A_j \). The example stores the observed configuration of stimuli surrounding the subject person, which consists of both internal and external stimuli. We assume that the internal stimulus
can be inferred from \( p \)’s annotated trajectory, and consider the people and objects that fall within a region surrounding \( p \) as external stimuli. For each stimulus, internal or external, the example stores its annotated trajectory, over the frames \([t - \delta, t]\), in the local coordinate system of the subject person, as seeing in Figure 3.7. \( \delta \) is a predefined constant which, in our experiments, is equal to 25.

Examples are generated from every frame along the trajectory of each person that appears in the input video. They are stored such that the subject person’s local coordinate system is aligned with a global coordinate system.

![Figure 3.7](image)

**Figure 3.7:** An example representing the stimuli that motivated subject person \( p \) to stop performing action \( B \) and start action \( D \). It consists of the subject person, the surrounding people (\( e_i \)) and objects, and their annotated trajectories over the past several frames.

The example set represents all the observed stimuli configurations that motivated an action to be performed. This example set is the basis according to which we will fit behaviors to the simulated crowd.

The example set is encoded into the action-graph, such that each example is stored on the edge corresponding to the transition between the actions that it represents. Thus, at run time, we compare the stimuli that exist in the simulation to that stored in the examples and decide how to traverse the graph and assign actions.
3.5 Stimuli-Maps

A stimuli-map acts as an influence function for a given action. It is used to estimate the relative amount of influence that a stimulus has for the given action, see Figure 3.8. In many works, the influence of a stimulus is merely distance-based; where the closer the stimulus is to the subject person the more influence it has on its actions. Stimuli-maps allow the influence to be arbitrarily involved. For example, an action such as waving to someone should be influenced by the existence of such a person, rather than by the people in the vicinity of the subject person.

For each action, a stimuli-map is used to approximate the importance, or rather, the amount of influence that each stimulus has within a set of stimuli for that behavior. At run-time, the appropriate map is used to evaluate the similarity between a query stimuli configuration \((Q)\) of a simulated agent and that of an example \((E)\).

![Figure 3.8: Stimuli-maps are density based influence functions surrounding the subject person (red arrow). Areas of high influence are marked in red and of low influence in white.](image)

A stimuli-map is constructed, for a certain behavior, according to the examples leading into the corresponding behavior node. A map is a two dimensional regular subdivision of the region of influence, in our experiments a map of size 400 x 200 was used. It is constructed for action \(A_k\) according to the examples leading up to it, i.e. examples representing transitions from any action \(A_j\) to action \(A_k\). From each one of the relevant examples, the last frame of the stimuli configuration is overlaid over the map and each cell accumulates the stimuli that fall within it. A Gaussian filter is applied such that each stimulus not only contributes to its own cell, but also to the surroundings ones (see Figure 3.9).
Given an axis-aligned configuration of stimuli, the amount of influence, $w_i$, associated with an external stimulus $e_i$ depends on the action being evaluated and the position of the stimulus. When evaluating the potential action $A_k$, the configuration is overlaid on top of $A_k$’s stimuli-map. The influence, $w_i$, equals the value stored in the cell to which the stimulus belongs. The internal stimulus, represented by the subject person’s annotated trajectory, is assigned the value $w_p$, which equals the average influence value of the external stimuli. The influence values are then normalized such that

$$w_p + \sum_i w_i = 1$$

The stimuli-map can be thought of as a height map. When a configuration is mapped onto it then the stimulus standing on the highest point, relative to the other stimuli, is the most influential stimuli in the configuration. The stimuli standing at the lowest point is the least influential. By weighing the influence of each stimulus in this manner we are able to better evaluate the similarity between two configurations.

### 3.6 Validity-Maps

Some actions require the presence of a specific stimulus in order to be performed. This stimulus is part of the actions inherent nature, for example, the presence of a
person to the left of the subject person when performing the talk-left action, see Figure 3.10.

The matching function, which is used to compare between two stimuli configurations and is described in section 4.2, does not enforce such requirements. Therefore, it will assign a similarity value even to configurations that are missing the requirement. The validity-maps are used to confirm that the stimuli that are mandatory for the performance of the behavior exist in a given query. The maps are defined and the requirements enforced without any user intervention or additional markings.

![Dense Crowd](image1.png)
![Sparse Crowd](image2.png)

**Figure 3.10:** Validity-maps represent regions where a stimulus must be present prior to performing the corresponding action. Crowds of different natures might produce different regions for the same actions.

A validity-map is constructed for a certain behavior according to the examples leading into the corresponding behavior node. As in the Stimuli-maps, a validity-map for action $A_k$ is constructed by overlaying the examples representing transitions from any action $A_j$ to action $A_k$, over the map. However, in this construction, the cells accumulate example ids. After all the relevant examples have been processed, a circular region is grown from each cell of the map until it encapsulates most of the examples. In our experiments, a 95% threshold was used. The regions with the smallest radii are used to define the required regions and the rest are discarded.
One condition is imposed over the regions, which is that the subject person cannot be included within them. The reason for this is twofold. First, the subject person exists in all of the examples, therefore, allowing him to be included within the regions will bias towards the ones closer to him. Second, behaviors that have such requirements, for example talking to someone on your left, or pointing to the right, tend to be directional. If the regions are allowed to grow around the subject person then the directionality element is lost.

In the validity-maps shown in Figure 3.10, the required regions for the point-left and point-right actions are very small. The reason is the limited number of times these actions were observed in the input video (only once for point-left), see Table 1. The result is a hard constraint over the stimuli configurations under which these actions can be performed. Since this occurs due to insufficient coverage of the configuration space, it can be resolved by using additional input data.

### 3.7 Points of Influence

A person interacts with the people and objects of interest in his surrounding environment. In the input video we mark objects that attract people's attention as Points of Influence, or POI’s for short. A point of influence may be a dummy in a shop window or a garbage can. Points of influence are considered as external stimuli in the same manner as a person is, see Figure 3.11.

![Figure 3.11: A point of influence represents an object in the environment. It acts as an external stimulus. Here the agent chooses to look towards the point, however the actions motivated by such a point are not limited to looking at it.](image)

A POI is an external stimulus whose position is considered as a trajectory annotated with a no-action tag. Therefore, it adheres to our representation of an external
stimulus. Although both People and POI’s influence the subject person, in the matching process people should not be matched to POI’s, otherwise unwanted behaviors, such as people talking to inanimate objects might occur. To accommodate the separability between people and POI’s for each action two stimuli and validity-maps are defined, one for the people and another for the POI’s. During the matching process, the appropriate maps are used depending on the type of the external stimulus. Note that the actions motivated by a POI are not limited to looking at it.

3.8 Preprocessing Summary

The preprocessing stage gets an annotated video of a crowd as input and produces an action-graph where:

- Nodes define observed actions.
- Each node stores the appropriate validity and stimuli-maps.
- Edges define transitions between actions.
- Each edge stores the examples that motivated the corresponding transition between actions.

Other than annotating the video, no additional user intervention is required. In our implementation between 2 to 5 minutes are need to process a minute of annotated input video. The time varies depending on the number of people and actions that appear in the video. Once this stage is complete, the data structure can be stored for future use.
Chapter 4

Fitting Behaviors

4.1 Overview

Once the preprocessing stage is done, we are ready to start fitting behaviors to simulated crowds. We are given the trajectories of a simulated crowd and by traversing the action-graph we fit behaviors to them.

Fitting behaviors is an online process in the sense that all the information that is needed for fitting behaviors to an agent at a given time is already available at that time. Therefore, if implemented efficiently enough, it can run alongside the crowd simulator and fit behaviors to the agents as they are being simulated. Ours is not an ideal implementation and still, for a reasonable sized crowd it runs in real-time.

The simulated crowd can be generated by any means available. Our algorithm does not rely on any specific simulation engine. In our experiments we used a Reynolds flocking crowd, a simple rule based crowd and an example based one. As long as the simulated crowd bears some resemblance to a pedestrian crowd, our algorithm should be able to fit natural looking behaviors to it.

Our system runs in parallel to the crowd simulation engine whose output trajectories are redirected as input for our system. For each simulated agent in the current frame that requires an action-tag, we find its stimuli configuration, $Q$, and current action $A_j$, in the same manner as for an example in the video. Our goal is to approximate the probability of each potential action. In terms of the action-graph, $A_k$ is a potential action if a directed edge exists between the node of action $A_j$ and that of action $A_k$ and $Q$ passes $A_k$’s validity test. The probability of action $A_k$ is the probability that $Q$ belongs to the examples representing this transition. By computing a similarity measure between an example $E$ leading from action $A_j$ to action $A_k$, and the query $Q$
we estimate the likelihood that a real person would have performed action $A_k$ given the configuration $Q$.

For each outgoing edge of the current graph node, $Q$ is checked against the destination nodes validity-map. If the query is invalid, then $Q$ does not have the necessary stimuli for performing the corresponding action, and the action receives a zero probability. For each edge that passes the validity test, we search its example set for the examples most similar to $Q$ and sum the similarity values. After normalizing the values from the different edges the probability of choosing an edge, or more precisely, the probability of performing the corresponding action, is approximated.

At run-time there is a need to periodically validate the assigned actions since the surrounding environment is constantly changing. The action is allowed to continue as long as it passes the validation test and an example similar to the agents’ current stimuli configuration exists on the self referencing edge of the current graph node. The reasoning behind this is that although the configuration changes over time, we can always find a real person in the input video that performed the action under similar circumstances. An action should not be cut shorter than the minimal time required for the animation to get in and out of it.

The following is a high level description of the behavior fitting scheme:

```plaintext
For each agent (i) in the current frame that requires an action assignment
{
    Create a query configuration $Q$ with agent i as the subject person
    Find agent i's action node ($A_i$) in the graph
    For each one of $A_i$'s neighboring nodes ($A_k$)
    {
        Validate $Q$ against $A_k$ validity-map.
        If $Q$ passes the test then
        {
            Compute the similarity between $Q$ and the examples of the edge $A_i\rightarrow A_k$
            Store the best matches
            Calculate edge probability
        }
        Randomly select edge according to probability
        Assign the chosen action to the agent
    }
}
```
4.2 Similarity Function

The similarity function, $Sim(Q, E)$, quantifies the similarity between two stimuli configurations. A query configuration, $Q$, originating from a simulated agent and an example configuration, $E$, leading to action $A_k$. Generally, $Q$ will not match exactly any of the examples which exist in the graph, there are always going to be unmatched or displaced stimuli. Therefore, the similarity between $Q$ and $E$ is a weighted sum of the similarities that do exist. Each stimuli $q_i \in Q$ is matched to the stimulus $e_j \in E$ most similar to it according to the similarity function $S(q_i, e_j)$. The weight, $w_i$, assigned to stimulus $q_i$ is determined according to the stimuli-map of action $A_k$.

$$Sim(Q, E) = w_p S(p, e_p) + \sum_{q_i \in Q} w_i \max_{e_j \in E} \{S(q_i, e_j)\}$$

Where $q_p$ and $e_p$ are the subject people of the configurations, representing the internal stimuli. The function $S(q_i, e_j)$ computes the similarity between two stimuli by taking into account the difference in position, $dP(q_i, e_j)$, and action, $dB(q_i, e_j)$, along their trajectories.

$$S(q_i, e_j) = \sum_{t} c_t S_t(q_i, e_j)$$

$$S_t(q_i, e_j) = 1 - \alpha dP(q_i, e_j) - \beta dB(q_i, e_j)$$

where $\alpha$ and $\beta$ are predefined constant weights whose sum equals 1 and in our experiments were equal to $\frac{2}{3}$ and $\frac{1}{3}$ respectively. $c_t$ is the relative weight of time $t$ along the trajectory and $\sum_{t} c_t = 1$.

The difference between actions, $dB(q_i, e_j)$, is the topological distance in the graph between the action-tags of $q_i$ and $e_j$ at time $t$, divided by the maximal topological distance in the graph.
The difference between positions, \(dP(q_i, e_j)\), is computed using the Euclidian distance, \(dist(q_i, e_j)\), between the positions of the stimuli at time \(t\). If the distance is over a user defined upper bound, \(r_{max}\), then the difference is 1. If it is under the lower bound, \(r_{min}\), then the difference is 0. Anywhere in between it equals the squared ratio between the difference of \(dist(q_i, e_j)\) and \(r_{min}\) and the difference of \(r_{max}\) and \(r_{min}\).

\[
P(q_i, e_j) = \begin{cases} 
0 & \text{dist}(q_i, e_j) < r_{min} \\
\left(\frac{\text{dist}(q_i, e_j) - r_{min}}{r_{max} - r_{min}}\right)^2 & r_{min} < \text{dist}(q_i, e_j) < r_{max} \\
1 & r_{max} < \text{dist}(q_i, e_j)
\end{cases}
\]

The \(r_{min}\) lower bound is proportional to the distance of \(q_i\) to the subject person \(q_s\) at time \(t\).

### 4.3 Similarity over Time

As we said before, determining if two people are behaving similarly requires that we check their behavior over time. Since we are trying to fit actions to some internal or external stimulus, then we need to match the recent stimuli of the people.

Obviously, as the length of the compared path grows, the less likely it is to find the two trajectories similar. Therefore, we limit the comparison to a predefined constant length, which in our experiments was equal to 12, and use a decaying function to weigh the importance of each time step. We defined these weights, which appear in the function \(S(q_i, e_j)\) as the constants \(C_t\), by sampling a Gaussian function at regular intervals (see Figure 4.1). This function emphasizes the last frames over previous ones, and represent balance between the flexibly of the comparison and it accuracy.
4.4 Example Weight and Cluster Weight

Ours is a data driven approach that needs to store a large database of examples. Having a big set of unique examples will certainly lead to a higher quality of fitted behaviors. However, having many similar, or practically identical examples, will mainly affect the amount of time needed in order to fit the behaviors but not their overall quality. On the other hand, there is meaning and significance to the fact that some configurations are more common than others. For example if two different action find a match of the same quality, however, one action relies on a configuration that was observed hundreds of times and the other action on one that appeared only a few times, then obviously, the first action should have a higher probability than the second one. For this reason, we cluster examples, eliminating duplicates. However, each cluster is given a weight that is equal to the number of examples that it represents.

The comparison function that we use is the same one that is described in section 4.2. If the similarity value of two configurations if higher than some predefined constant, in our experiments 85% was used, then the configurations are considered as duplicates.
Having clustered the examples, when fitting behaviors we still want to find the best matching examples. Instead of taking the top x% of the matches, we find the best match and then add all the matches that have no more than a 10% difference in their score from it. In this manner the matching accounts both to the quality of the match and to its frequency in the real world.

4.5 Online Behavior Fitting

Actions are assigned probabilistically. For the current agent, we look at the graph node of its current action. The outgoing edges represent possible actions whose probability of being assigned is determined by the function $\text{Sim}(Q,E)$. After the probability function is defined. The agent samples it and is assigned the corresponding action-tag.

An agent is not assigned a new actions-tag in every frame. Rather, each action is allowed to continue as long as its stimuli configuration passes the actions validity test and a similar example exists for this action. Additionally, once a new action-tag has been assigned we allow it a minimal time period before it can be changed. We found that 12 frames are sufficient for an animation system to portray a reasonable performance of the action. Although action-tags can be assigned every 12 frames the actions usually continue for a longer period of time. The reason is the use of the action-graph along with the probability approximation. Each node has a self-referencing edge. As long as this edge holds an example that matches the agents’ current stimuli configuration, then there's a good chance for the action to continue.

Despite the fact that 12 frames is a short period of time, it is long enough for an action to seem peculiar if it is not valid. Over the course of the 12 frames the environment constantly changes. A person that was there a moment ago may have left, leaving the talking person all alone. For this reason, during the 12 frame interval, we constantly validate the actions, using the validity-maps, and stop any action that doesn't pass the test.
Chapter 5

Results and Analysis

5.1 Implementation

The method was implemented in C# under .NET framework. The implementation contains 3 major components:

- The video annotation user interface component.
- The action-graph creation component.
- The behavior fitting component.

All the results that appear in the paper and in the accompanying video were animated automatically using a motion-graph and were rendered off-line in Maya 7.0.

Video annotation user interface:

Our user interface is simple and intuitive. Trajectories are defined by clicking on each person's position once every several frames. Similarly, the trajectories are annotated by clicking on the trajectory and choosing the observed action. No complicated procedures are required nor any prior knowledge regarding human behavior needed. Even non-technical users can easily use our annotation system (see Figure 5.1).

Action-graph creation:

We implemented the action-graph, which was described in section 3.3. The system generates all the necessary examples and maps and stores them on the graph. For debugging purposes, we implemented different components that allow the user to examine the different maps that were created, overlay the examples on top of the input video and more.
Fitting Behaviors:

In our implementation, the simulated trajectories are read from a file. However, the system fits behaviors as though a simulator is running in parallel to it, and none of the information regarding an agent's future trajectory is processed beforehand. In this component as well, several visualizations were implemented in order to verify the system's performance. One such visualization allowed the user to choose an agent, as the behaviors are being fitted, and see its stimuli configuration as well as the most similar examples of each candidate action and their associated probabilities (see Figure 5.2).

Our system is not limited to fitting a single action at a time. Both the input model and the data structures support multiple actions-tokens. Real people can talk on the phone while combing their hair. Our system can easily support such complicated actions, however, due to limitations on the part of our animation system we avoided assigning multiple tags to the simulated agents.
The results presented here are based on a single video. The specific video is five minutes long and captured unaware pedestrians walking in front of a department store. The maximal number of pedestrians per frame is 18, with an average of 5-6 people per frame. They either walk on their own or in small groups of 2-4 people. We used our system to rapidly annotate their trajectories. Additionally, four dummies in a shop window, which attracted people’s attention, were marked as points of influence. For defining the examples, an oval region of influence contained in a rectangle of size 200 X 400 pixels was used. This translated roughly into a 3.5 by 7 meter region surrounding the subject person. Twelve actions were used; the eleven presented in Section 3.2 and an additional action representing no-action. The method can easily accommodate a wider selection of actions with no significant degradation in performance. The number of annotated actions in the input video is 253, which results in 49,269 examples.

An action-graph of 12 nodes and 63 edges was constructed. For each node a stimuli-map was created. Validity-maps were created for six of the twelve actions, which were found to have distinct validity regions, see Figure 3.1. All maps have a
resolution of 200 X 400 pixels. A square 9 X 9 Gaussian filter was used in their creation, see Figure 3.9. The memory required to store the complete data-structure is under 100 MB.

Generally speaking, the number of potential stimuli configurations motivating the performance of an action is immense and therefore the amount of input data used has a direct influence on the quality of the probability approximation. However, even a short video of a typical crowd shows enough variety of actions and configurations, so that the overall behavior seen in the video can be captured and fitted to a simulated crowd.

5.3 Run-time Performance

The following are the system configuration and simulation parameters that were used to monitor the performance of the C#.Net implementation of the algorithm.

**Testing environment:**
- **OS:** Windows Vista
- **Processor:** AMD Athlon 64 X2 Dual Core 5200+ 2.61GHz
- **Memory:** 2046 MB

**Simulation Info:**
- 3000 Frames (25FPS) length (2 minutes)
- Total 8029 unique Examples learned (from 49876 Examples DB)
- **Action validity checked every 2 frames.**

The run-time performance does not deal with the creation of the database. It also does not deal with the actual simulation process or any animation / visualization that may be performed. The run-time performance is concerned only with the task of fitting behaviors to the simulated agents. It depends mostly on the number of examples tested for each agent.

To improve the performance of the method and reduce the memory requirements we performed two optimizations. First, the number of examples was reduced by clustering the similar examples on each edge. The second optimization involves the no-action node. Over two thirds of the examples relate to this node. Even after clustering the number of remaining representatives is large. We found that for most
query stimuli configurations a fairly similar one exists in the no-action example set. We make the assumption that no-action can be performed under every possible configuration of stimuli. Therefore we create a single cluster for all the examples leading to the no-action node and give them a single constant weight.

The main cost of the method is the comparison between stimuli configurations. The number of comparisons depends on (a) the number of examples and (b) the density of the stimuli (people per square meter). Option (b) also increases the cost of a single comparison since the number of stimuli in a query configuration rises. To test the scalability of the method we ran several tests.

We assigned behaviors to the same 3000 frames (2 minutes) long simulation three times. The number of examples based on which the action-graph was constructed varied between the tests by a multiple of 5 and 10. The number of example clusters was 1855, 7439 and 13035 and the time required to assign the behaviors was 35 sec, 65 sec and 78 sec respectively (see Table 5.1). Obviously, there is an increase, although sub-linear, in the required computational time. The number of clusters increases in a slower rate than the number of examples since new examples can be added to existing clusters.

Next, we varied the density of the crowd in a small region. We constructed a single action-graph and used it to assign behaviors to four 3000 frames (2 minutes) long simulations. The simulations varied in the number of people, and had 6, 11, 20 and over 40 simulated people all of them using our default input that contained 13,035 cluster examples. The required computational time was 18 sec, 35 sec, 78 sec and 226 sec (see Table 5.1).

A clear dependence between the computational cost and the density can be seen. However, this dependence has an upper bound as there is a limit to the number of people that fit in a square meter. It seems like the simulation which has over 40 people in it is close to the upper limit for a pedestrian crowd.
Results and Analysis

All of the tests were performed as a single threaded implementation. It should be pointed out that the action fitted to an agent at a certain time has no affect on the actions fitted to other agents at the same time. Therefore, a multithreaded implementation seems like a natural and simple extension.

The implementation presented here was not aimed at optimal performance, but rather at flexibility. The goal was to be able to add / subtract / change parts of the code easily and test new ideas as the research progressed. We chose a C# .NET environment which runs under a virtual machine and uses managed memory with a garbage collector. This implementation can be translated easily to a more efficient environment and programming language thereby increasing the systems performance by several factors.

For offline simulations and for research purposes, our implementation proved to be efficient enough. Other than implementing the algorithm again using a more efficient environment and a multithreaded approach, one can improve the efficiency of the algorithm at the sake of the fitted behaviors quality by reducing the number of comparisons and simplifying the calculations.

<table>
<thead>
<tr>
<th>Test Scenario</th>
<th>Avg. #people per frame</th>
<th>#Clusters</th>
<th>Memory Consumption</th>
<th>Computational Time (secs)</th>
<th>Computation Time Relative to Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduced input example set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preprocess 10% of input. Average density crowd</td>
<td>20 people</td>
<td>1855</td>
<td>92MB</td>
<td>35.1</td>
<td>44.96%</td>
</tr>
<tr>
<td>Preprocess 50% of input. Average density crowd</td>
<td>20 people</td>
<td>2956</td>
<td>102MB</td>
<td>65.4</td>
<td>83.81%</td>
</tr>
<tr>
<td>Baseline - Preprocess 100% of input. Average density crowd</td>
<td>20 people</td>
<td>13035</td>
<td>119MB</td>
<td>78.0</td>
<td>100.00%</td>
</tr>
<tr>
<td>Variant crowd density</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very light crowd</td>
<td>6 people</td>
<td>13035</td>
<td>116MB</td>
<td>18.8</td>
<td>24.10%</td>
</tr>
<tr>
<td>Low density crowd</td>
<td>11 people</td>
<td>13035</td>
<td>117MB</td>
<td>35.4</td>
<td>45.32%</td>
</tr>
<tr>
<td>Baseline - Average density crowd</td>
<td>20 people</td>
<td>13035</td>
<td>119MB</td>
<td>78.0</td>
<td>100.00%</td>
</tr>
<tr>
<td>Very dense crowd</td>
<td>60+ people</td>
<td>13045</td>
<td>113MB</td>
<td>226.1</td>
<td>289.93%</td>
</tr>
</tbody>
</table>

Table 5.1: Changing the size of the example set and density of the crowd affects the systems performance. The table shows that the memory consumption and computational time are not linearly dependent on the number of clusters. However, the computational time is linearly dependent on the average number of people in the simulation.
Reducing the number of comparisons:
Most of the computational time is spent on comparing the query configuration to the examples. Reducing the number of examples would reduce the number of comparisons and therefore alleviate some of the effort. The idea is not to use fewer examples, but rather use fewer clusters. We still want to have a large variety of examples, but if we make them less unique then we can reduce their overall number. This can easily be done by changing the threshold value used for determining if two examples are the same during the database construction process. Another option for reducing the number of comparisons is to change the length of the interval between action assignments, which was described in section 4.5. In our experiment, an interval of 12 frames was used. However, the length of the interval can be increased thereby reducing the computational cost. Of course, there is a limit as to how long this interval can be extended without noticeable degradation in the quality of the fitted behaviors.

Reducing calculation complexity:
The computational complexity of matching a query configuration to an example configuration depends on two factors; the number of stimuli in each configuration and the length of the trajectories compared for each stimulus. In the comparison process each one of the query stimuli is compared against each one of the example stimuli. Therefore, if we reduce the size of the influence region, the number of stimuli within the configurations will be reduced and the comparison made simpler. Additionally, when two stimuli are compared then their trajectories are compared. Using shorter trajectory segments will reduce the computational complexity as well.

Let us reemphasize the point that all of the above mentioned modifications will increase the algorithm's efficiency, however may reduce the naturalness of the fitted behaviors.
5.4 Evaluation of Fitted Behaviors

To show the generality of the method we present results for four different types of crowd trajectories:

- Real captured data.
- A flocking simulation [9], Figure 5.3 top.
- A Rule-based simulation, Figure 5.3 bottom.
- An Example-based simulation [6], Figure 5.4.

Two different experiments were run on real-data. We checked the frequency of the fitted actions and their average length, see Table 5.2. The distribution of actions produced by the action-graph method (Column 3) corresponds quite closely to the real-data (Column 2). Obviously, our method is stochastic in nature, and therefore, cannot be expected to reproduce the exact same frequencies. A random assignment of actions, based on their frequencies in the input data, yields, as expected, a distribution similar to the input data (Column 4). However, there are several significant problems
with random selection. First, the assigned actions do not look natural. People talking to thin air are common in these simulations. One might argue that these behaviors can be easily eliminated using a simple set of rules. Instead of manually defined rules, we used the validity-maps and filtered out these actions (Column 5). Although this resolved one problem, it did not resolve un-natural behaviors caused by the length of the assigned actions. The average length of an action produced by the action-graph is similar to the average length in the real-data. The same cannot be said for both random assignments, where very short actions are frequently found. Again, one might argue that this can be easily resolved by assigning a fixed or variable length to the chosen action. However, in our method there is no need to determine the length or the frequency of the actions in advance. Rather, they are determined according to the stimuli surrounding the agent during the simulation and their similarity to stimuli which motivated the same actions in the real world.

<table>
<thead>
<tr>
<th>Action Type</th>
<th>Real Data</th>
<th>Action-Graph</th>
<th>Random</th>
<th>Random &amp; Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Count</td>
<td>%</td>
<td>Count</td>
<td>%</td>
</tr>
<tr>
<td>Talk to right</td>
<td>54</td>
<td>18.4%</td>
<td>68</td>
<td>11.7%</td>
</tr>
<tr>
<td>Talk to left</td>
<td>48</td>
<td>17.4%</td>
<td>53</td>
<td>9.3%</td>
</tr>
<tr>
<td>Look at right</td>
<td>53</td>
<td>21.3%</td>
<td>83</td>
<td>27.5%</td>
</tr>
<tr>
<td>Look at left</td>
<td>43</td>
<td>14.0%</td>
<td>77</td>
<td>12.4%</td>
</tr>
<tr>
<td>Looking down</td>
<td>3</td>
<td>0.7%</td>
<td>25</td>
<td>2.0%</td>
</tr>
<tr>
<td>Look back</td>
<td>15</td>
<td>4.5%</td>
<td>59</td>
<td>5.3%</td>
</tr>
<tr>
<td>Point to left</td>
<td>1</td>
<td>0.2%</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td>Point to right</td>
<td>3</td>
<td>0.7%</td>
<td>5</td>
<td>0.6%</td>
</tr>
<tr>
<td>Talk on cellular</td>
<td>11</td>
<td>14.0%</td>
<td>51</td>
<td>13.4%</td>
</tr>
<tr>
<td>Arrange hair</td>
<td>15</td>
<td>7.0%</td>
<td>59</td>
<td>9.7%</td>
</tr>
<tr>
<td>Look at watch</td>
<td>7</td>
<td>1.8%</td>
<td>50</td>
<td>8.1%</td>
</tr>
</tbody>
</table>

Table 5.2: A comparison, against the input data, of the frequency and length of the behaviors selected by three different methods: our method, random selection and random selection filtered with the validation-maps.

To emphasize the last point, we applied our method, using the same set of input examples, onto three very different situations: a flocking, a rule-based and an example-based simulation. The flock consists of a large group of agents walking together. The rule-based simulation mostly generates individual trajectories, while the example-based simulation is a mix of both, as can be seen in the accompanying video. The experiment showed that our method accounts for the specific circumstances of each character. The resulting fitted behaviors reflect the different natures of the simulations. For instance, talking accounts for 52.8% of the actions fitted to the flock,
for 20.4% of the example-based actions and only 3.5% of the rule-based ones. The average length of a talking action is also significantly different. For the flock and example-based simulations the average length is approximately 64 frames while for the rule-based simulation it is only 31 frames long. Looking in any direction, is an action that is as likely to be performed when walking alone, as when one is part of a group. However, talking on the phone, looking down and looking at the watch are actions that were performed more by individuals in our input data. These actions account for 23.6% of the total actions in the rule-based simulation, while for only 12.3% and 10.4% of the example-based and flock actions. A random assignment has a specific frequency of actions assigned to it. A rule-based assignment of behaviors would have to be modified and tweaked to fit the nature of the crowd. On the other hand, our method is more general and fits behaviors based on the surrounding stimuli and their similarity to the examples.

Our system provides the user with control over the frequency of behavior in two ways: (a) Scaling the cluster weight of the no-action node, which affects the overall frequency of actions. By scaling the weight, either up or down, the user changes the probability of performing no-action, therefore, increasing or decreasing the frequency of the other actions, see Table 5.3. (b) Scaling the weight of an arbitrary node in the graph which affects the frequency of a specific action. These controls can be applied globally to all the agents or just to specific ones. For example, if some individual is known to frequently speak on the cellular phone, then we can scale the corresponding weight for this specific agent. Note that this control does not violate the validity of the action selection algorithm. By scaling up the weight of a certain action we reduce the importance of the stimuli. So, conceivably, an improper behavior might be selected. However, the presence of the validity-map assures that hard constraints are always satisfied.
5.5 Educating Your Crowd

The approach that we present in this work is example-based. For the most part, when fitting behaviors one would like to obtain the examples from the real world. However, that is not the only option. If, for some reason, unique actions are required then specific synthetic examples can be defined and added to the system. This can be useful to "direct" the behaviors of the crowd and add unique behaviors to it. All that is required from the animator is to supply a few examples of the behaviors that he wants to fit to the crowd. In some respects this is similar to a rule-based assignment, however in our case, the animator is not required to defined rules, but rather supply an example.
Figure 5.4: The trajectories in these images were taken from an example-based crowd Simulation.
Chapter 6

Conclusions

In this paper we presented a data-driven method for fitting behaviors to pedestrian crowds. The aggregate effect of the performed actions increases the realism of the simulated crowd. The method runs in real-time for crowds up to several dozen agents in size. It can be used in real-time applications such as games to enhance the ambient crowds, or it can be employed in off-line productions to lighten the load of the animator. Additionally, the output of the fitted behaviors can be redirected and used as additional input for the crowd simulator. This will allow the simulator to generate trajectories that take into account the actions that the people perform.

The system has its limitations. The circles used for defining the validity regions, can cover a larger area than required. Therefore, in rare cases a person standing on the edge of a region might cause a less than natural behavior. However, this and the restriction which is applied during the regions construction can be alleviated by using a different clustering technique. Another limitation is that an action is not directed at a specific target. For example, while talking, there are occasions when the person being talked to leaves and another passerby takes his place. The action is valid but unrealistic. This could possibly be solved by having targeted animations.

Conceptually, an important contribution of this work is the introduction of the stimuli and validity-maps. They provide a detailed, non-linear method for assessing the importance of each feature when evaluating a complex situation. This information is derived directly from example data, avoiding the need to define weights for all features/situations. We believe that by studying stimuli and validity-maps one can extract information regarding interactions among people in a crowd that can be used to enhance rule-based systems or for quantitative behavior analysis of people in a crowd.
References


Acknowledgments

We would like to thanks to Alon Lerner for his major support in this work and for the tracked video, simulation trajectories data, and for the video rendering engine and animation that you can see in the accompanying video. We would also like to thank to Yiorgos Chrysanthou for his assistance in writing to original article.