1-1-2015

Steering Contexts for Autonomous Agents Using Synthetic Data

Cory Boatright
University of Pennsylvania, coryb@seas.upenn.edu

Steering Contexts for Autonomous Agents Using Synthetic Data

Abstract
Data-driven techniques have become synonymous with replication of real-world phenomena. Efforts have been underway to use these techniques in crowd simulation through a mapping of pedestrian trajectories onto virtual agents using a similarity of circumstance. These works have exposed two fundamental issues with data-driven crowds.

First, robust real-world data is logistically difficult to accurately collect and filled with unknown variables, such as a person's mental state, which change behavior without providing a means to replicate their effects. Second, current data-driven approaches store and search the entire set of training data to decide the next course of action for each agent. A straightforward single-model system would alleviate the burden of storing and searching the data. The problem with a monolithic model, though, is that a single steering policy cannot handle all possible scenarios. To counter this we propose the splitting of possible scenarios into separable contexts, with each context in turn learning a model. The model used by an agent can then be dynamically swapped at runtime based on the evolving conditions around the agent. This results in a more scalable approach to data-driven simulation.

In lieu of tracked data from real pedestrians, we propose the use of an oracle steering algorithm. This algorithm stands in for real data and can be queried for a steering decision for any combination of factors. This allows us to more thoroughly explore the problem space as needed. Furthermore, we can control all variables and capture behavior from scenarios that are otherwise infeasible to adequately sample in reality. This synthetic source of training data allows for a scalable and structured approach to training machine-learned models which virtual agents can use to navigate at runtime.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Norman I. Badler

Subject Categories
Computer Sciences

STEERING CONTEXTS FOR AUTONOMOUS AGENTS USING SYNTHETIC DATA

Cory D. Boatright

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial

Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2015

Supervisor of Dissertation	Graduate Group Chairperson
Norman I. Badler, Professor, Computer and Information Science	Lyle Ungar, Professor, Computer and Information Science

Dissertation Committee

Petros Faloutsos, Associate Professor, York University (External Committee Member)	Ladislav Kavan, Assistant Professor, Computer and Information Science
Stephen H. Lane, Associate Professor of Practice, Computer and Information Science	Aaron Roth, Assistant Professor, Computer and Information Science

STEERING CONTEXTS FOR AUTONOMOUS AGENTS USING SYNTHETIC DATA

To my grandfathers, Howard D. Boatright and James C. Kuntz, and my cousin Ryan C. Boatright who unexpectedly joined them far too young. They did not see me finish my education in Person, but I know they are watching in Spirit.

Acknowledgements

While it's common to thank "all the little people" who contributed in our accomplishments, but I would like to thank two by name. Kimberly Detwiler sat next to me in Intro to Comp Sci so many years ago and her request for help kicked off my interest in teaching. Costel Ionita was the first to tell me getting a PhD is not *that* big a deal and that I should consider it, even though he'd just found me skipping his Calc III class. Looking back, there's a fair chance he got the last laugh on that one. These two people inadvertently started the ripples that would lead to this day.

Once the ripples started, I have Oberta Slotterbeck and Ellen Walker from Hiram College to thank for pushing me to the University of Pennsylvania, and my advisor Norm Badler to thank for taking a chance and pulling me in. I have often thought he lost the GRE page of my application. His humble approach to mentoring a young wannabe professor was often exactly what I needed. While at UPenn I've been the beneficiary of support from many amazing people: Aline Normoyle, Amy Calhoun, Ben Sunshine-Hill, and Chris Czyzewicz to name a few in particular. I also have major appreciation for Jennie Shapira's assistance. She and I would agree I *could* have done all this without her, but it would not have been anywhere near as fun. She made all the figures in this dissertation that required more effort than hitting "prt sc" which is where my illustrative abilities abruptly end.

Finally my family's unwavering support has been a blessing on which I can always

depend. My Mom, Dad, and sisters Cari and Dayna, who have supported me my whole life in all of my endeavors, even when they didn't really understand them. Of course I've saved the best for last, my wife Dr. Kaitlyn Boatright. She has seen me at my best, got me through my worst, and never had trouble with believing in me...something I cannot claim of myself. Thanks, princess.

ABSTRACT

STEERING CONTEXTS FOR AUTONOMOUS AGENTS USING SYNTHETIC DATA

Cory D. Boatright

Norman I. Badler

Data-driven techniques have become synonymous with replication of real-world phenomena. Efforts have been underway to use these techniques in crowd simulation through a mapping of pedestrian trajectories onto virtual agents using a similarity of circumstance. These works have exposed two fundamental issues with data-driven crowds.

First, robust real-world data is logistically difficult to accurately collect and filled with unknown variables, such as a person's mental state, which change behavior without providing a means to replicate their effects. Second, current data-driven approaches store and search the entire set of training data to decide the next course of action for each agent. A straightforward single-model system would alleviate the burden of storing and searching the data. The problem with a monolithic model, though, is that a single steering policy cannot handle all possible scenarios. To counter this we propose the splitting of possible scenarios into separable contexts, with each context in turn learning a model. The model used by an agent can then be dynamically swapped at runtime based on the evolving conditions around the agent. This results in a more scalable approach to data-driven simulation.

In lieu of tracked data from real pedestrians, we propose the use of an oracle steering algorithm. This algorithm stands in for real data and can be queried for a steering decision for any combination of factors. This allows us to more thoroughly explore the problem space as needed. Furthermore, we can control all variables and capture behavior from scenarios that are otherwise infeasible to adequately sample in

reality. This synthetic source of training data allows for a scalable and structured approach to training machine-learned models which virtual agents can use to navigate at runtime.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

And so it begins...

Gandalf the White

Simulations of pedestrian behavior range from "multiagent simulations" of a few people to "crowd simulation" with thousands to millions of virtual agents all vying for system resources. This pressure on compute power creates a tense battle between the scale of the simulation and the algorithms needed to display acceptably realistic behavior. As the number of agents increases, the algorithms must be leaner in their resource use to accommodate the population. Adding to the challenge, more robust steering algorithms require the agent to be capable of handling an increasing number of possible situations. In this dissertation we explore our hypothesis that appropriate machine learning techniques can address the problem of crowd steering while also mitigating scalability problems and leveraging the realism benefits of empirical, data-driven simulations.

1.1 Population Versus Realism

Two dominant forces continuously push the development of crowd simulation. Both of these factors are manifest through a problem of scalability. When we ordinarily think of scale, we think of the number of inputs to a system. However with a goal as complex as crowd simulation, we must also consider scale with respect to the possible behaviors produced by the artificial intelligence.

The first force is the count of agents in the simulated scene. We want and need to simulate ever-higher numbers of simulated humans. With larger populations, we can simulate more complex scenarios and move towards duplicating the crowding phenomena of the real world and what we witness in day-to-day life. Modeling more agents strains against the available processing power of a single machine, as each agent requires processor time to make navigation decisions and to avoid collisions to ensure that, just as in reality, the population as a whole moves to its target configuration without agents striking one another.

The second force is realism. The practice of simulating large numbers of humans is not complete with just the existence of a massive population. Each member of that population, *a priori*, is expected to navigate in a manner consistent with our intuition of human action such as through use of individualized strategies or policies that lead to some goal or destination. In short, typical steering challenges do not cause confusion in a human, and thus a simulated person should be capable of appropriately handling the same challenges. Creating computational solutions for such a complex system often rely on ad hoc mathematical models or preconceived designer intuitions. Both generalizations may be lacking in correctness and require scalable, efficient algorithms to be effective on typically tight runtime and hardware constraints. For the sake of scalability we must strive to reduce the computation

needed to replicate plausible human behavior in order to allow more people to be simulated within the targeted, often real-time[1], framerate.

1.2 Machine Learning as Precomputation

To properly extend the science of crowd simulation we need an efficient, scalable strategy for steering given a virtual environment. Cognitive algorithms have shown promise, but ultimately suffer from a combinatorial explosion of the possible factors contributing to the decision-making process. Rather than striving to replicate the true thought processes of a human and subsequently devoting resources to carry out these thoughts for every agent in the simulation, we instead model input stimuli and observe the decision-making which uses these stimuli. While human factors such as personality and emotional factors likely influence results, we can set those variations aside in favor of explicitly computable navigation directives. In essence we treat cognition as a "black box" such that external influences are input, decisions are the output, and the mechanics behind producing those decisions only matters insofar that the decisions are reasonable upon inspection. Thus we abstract away the complexities of a true-to-life thought process.

Precalculating information for caching and future lookup helps improve the speed of many algorithms by reducing the amount of computation required at runtime by replacing it with a lookup to the cached results. In this spirit, data-driven tools have been demonstrated which select proper behavior via a best-match search against a database of samples. These samples serve as a surrogate for past experience which can be consulted when a similar situation arises. However, these techniques suffer from scalability issues of their own. The phrase "throw more data at it" cannot handle unbounded numbers of samples, and sampling bias can lead to holes in the resulting

action space since unobserved behavior cannot be represented by the system. Further complicating matters, numerous scenarios can have different actions because of factors not accounted for by the current feature space in use by the agents. Expanding the feature space creates an arms race between possible outcomes and unique combinations of features, ultimately making generalization impractical without machine learning algorithms.

With machine learning, general models can be fit rather than using all data for a best-first match. Learning fits hyperplanes and other hypersurfaces around the data with three key advantages. First, the model itself can be saved and loaded into memory rather than the full collection of samples. This can drastically reduce the size of the memory footprint. Second, the trained models can be executed faster than an extensive database can be searched. Finally, the machine-learned models allow predictions in novel situations where the input is only approximate compared to samples acquired during observation. Generating these models are not without its challenges. We gain these benefits but still have an issue of generalization versus memorization, where the algorithm overtrains to the data. We break apart the problem space into what we have named "steering contexts." These steering contexts can allow the training of smaller models on more manageable amounts of data.

1.3 Data Sources

To create the best models, training samples should be of the highest possible quality. Tracked trajectories of real-world pedestrians have thus far been the gold standard of training examples, however this type of data is logistically challenging to collect, likely incomplete with respect to behavior coverage, and at times simply impossible to gather in an accurate manner.

In naïve pedestrian observation, discrete cameras are placed to record an unaware population. While those seen in the recordings are not affected by the observation, uncommon scenarios may never be encountered and factors unanticipated and unaccounted for by the researchers may lead to hidden influences. For instance, a person's navigational goal may change due to meeting a friend on the street, receiving a phone call, or reacting to an event outside of the camera's view.

Another option for data collection is to gather volunteers and take direct control through the assignment of starting positions and end goals. Unlike the naïve case detailed above, the artificial nature of the scenario leads to the observer effect and thus calls into question the resulting behavior of participants. In particular we must question if we are seeing truly natural behavior, or instead what the participants *think* should be natural.

With the advances made in crowd simulation over the past decades, we find at our disposal numerous software systems which have been published as suitable representations of human behavior while steering through crowded environments. These artificial humans provide us with a powerful opportunity. The computer agents do not care—or know—if they are being observed yet can be given any start and goal configuration needed. These virtual agents' behavior can be used to supply data to in turn generate training data for machine learning algorithms.

1.4 Context-Sensitive Steering

This dissertation explores the next push forward in multiagent steering by addressing each of the following issues.

- Scalability, both in the space of possible situations and in the number of agents.
- Fitting models to data representing the vast problem space.

- Finding reliable training data.

We have developed a framework which uses a novel context-sensitive approach to create a collection of machine-learned models for multiagent steering. This process starts with a slow, offline algorithm and produces a faster algorithm based on the slower algorithm's results. The workflow consists of the following steps.

1. An algorithm derived from Iterative Deepening A* (IDA*), which we call our oracle algorithm, is used to create a nearly optimal plan for stochastically generated simulations.

2. The recorded simulations are grouped into steering contexts and mined for training samples.

3. Models are fit to the data for each steering context, and an additional model is used to dynamically switch between models as the situation merits.

4. At runtime the agent uses the model matched to its current situation to choose its next action[2].

1.5 Dissertation Structure

Chapter 2 surveys the literature most pertinent to this work. We then define steering contexts and our initial set of intuitively-defined contexts in Chapter 3. Subsequently, Chapter 4 begins the detailed explanation of this dissertation's use of synthetically generated data to stochastically sample scenarios from each context. With contexts defined, a pipeline which uses this context-sensitive technique to steering is detailed in Chapter 5 along with a preliminary implementation of the pipeline. We explore potential refinements to this implementation through the use of machine

learning in Chapter 6, which includes both the introduction of clustering to define contexts and an analysis of various classification algorithms. An adjusted implementation of the pipeline which takes these improvements into account is presented in Chapter 7. We conclude and speculate upon future avenues of research based on the framework established by this dissertation in Chapter 8.

Notes

[1]A minimum of 20 frames per second, or 50ms per frame. An interactive framerate is approximately 10 frames per second, or 100ms per frame.

[2]In this dissertation all agents share the same model, however the models could theoretically vary by agent to accommodate different roles and other variation of behavior.

Chapter 2

Related Work

> If I have seen further it is only by standing on the shoulders of giants.
>
> *Isaac Newton*

Crowd simulation is a well-developed field of research with several focus areas. Of particular interest to this dissertation are those of agent steering and evaluation of the generated crowd with respect to real-world fidelity. A general overview of some key milestones in the field is given in Sections 2.1 and Section 2.2. Additional, broader surveys of steering and behavior can be found in [49, 72, 63]. In Section 2.3 we specifically focus on data-driven approaches which are more relevant to this dissertation.

2.1 Agent Steering and Collision Avoidance

The most basic crowds model pedestrians walking from waypoint to waypoint in the virtual world. They can appear to wander as they walk towards randomly assigned goal points. There are at least three sources of randomness in crowd simulation:

how an individual selects its goals, how it creates its path to the goal, and how the agent decides on actions when it encountered obstacles—and other agents—along its path. An agent needs a technique for selecting goals to give them a destination for walking. This can be done stochastically [62], through utility functions which can give the agent needs [3], or can simply be hardcoded as experimental arrangements [59, 60]. The path an agent will follow can be generated in a myriad of ways ranging from geometrically-oriented [16], to flow-based [25, 26], and to cognitively-driven [57] algorithms. Once decided, the agents' paths are not necessarily guaranteed to be safe, as previously unknown obstacles may be discovered which invalidate the plan. Other agents may also move into the path and thus invalidate it. Adjusting for these new environmental constraints adds to the overall complexity and apparent randomness of the scene.

Crowd simulation was launched with the seminal work of Reynolds [54] on animating grouped movement behaviors of collections of agents. He focused on populations of animals, especially birds, to drive a particle-system approach to crowd simulation. His "boids" paved the way for studying the group dynamics of human crowds and animating substantial populations of interacting agents. This rule-based approach also serves as one of the first examples of emergent behaviors, which are characteristics of a simulation that are a result of agents' individual decision-making but has the appearance of higher-level coordination. Boids may fly in a common path but each agent is not consciously trying to create an ordered flock. This work was followed by [55] where human maneuvering was defined. These definitions led to boids with more natural behavior in a pedestrian simulation by providing building blocks for more complex behaviors, but the complex behaviors are not emergent and instead identified by the programmer at design-time.

While Reynolds focused on steering while taking collision avoidance into account,

other research has focused almost exclusively on collision avoidance during a simulation through purely geometric solutions. Velocity obstacles [16] are a common approach that look at velocity-space for potential collisions and avoid them by navigating around the troublesome velocities themselves. This allows an agent to reduce or increase their speed in advance of a collision and make other natural adjustments. A drawback to this approach is oscillation inherent in each agent constantly adjusting to the other agents' adjustments. By changing one's speed, the velocity obstacle changes which leads to a different viable solution. The technique was improved upon by [69] to create reciprocal velocity obstacles (RVO) which extended agent considerations to that of the colliding agents' response. This alleviated most of the oscillation problems associated with the original method but at the cost of making true—but poorly justified—assumptions in their model. In the RVO algorithm, it is assumed that an approaching agent is also using the RVO algorithm. While this is true in a homogenous simulation and leads to orderly passing of one another in cramped conditions, real life is not so symmetric. The ClearPath algorithm [21] further transforms velocity obstacles into a truncated cone and presents a highly parallel solution supporting hundreds of thousands of agents in a matter of milliseconds, also with a symmetric assumption. The environment has also been used to provide hints to agents as to the behavior needed for better navigation, as seen in [70].

Anticipating crowd densities and planning around them is the tactic used in the 2009 work by Kapadia et al. [29]. Their algorithm also adds some human factors into consideration when planning an agent's next steering decision. A coarser view of the world exists further away from the agent, and a path through occupied space is planned and refined as the agent gets closer to its goal. While a step forward, the algorithm lacks a viewing frustum so obstacles all around the agent are always considered in steering. Complex human reactions in the real world often take place when a

person is surprised by an unanticipated obstacle suddenly entering their perception. Such a startled reaction occurs because the new obstacle must be rapidly assessed and accommodated for without the benefit of long-term planning. This work has just recently been parallelized in [30].

Until very recently, steering and collision avoidance algorithms have worked with path-planning and with animation handled as a separate process. Singh et al. published a new algorithm which considers actual footsteps when steering [61]. This not only supports a new level of realism in steering and collision avoidance but also provides an improvement for the visualization of the simulation itself. Instead of depending on potentially poorly matched or interpolated walk cycles the animation can be derived for each footstep. This also allows for sidestepping during collision avoidance, which is an almost universally overlooked strategy to resolving such a problem.

Higher-level abstractions exist. Viewing crowds as a collection of people under the effect of social forces is a common simplification that casts a crowd into a less autonomous space as in [24, 40, 67, 50]. In this area of the literature particle physics is adapted to use social forces rather than natural forces like gravity [23]. Each participant in the simulation is treated as a particle subject to these forces and moved accordingly with each time step. The crowd becomes a unified organism with no individual drives, making specific breakout characters harder to motivate and inject into the simulation without partially abandoning the very assumption made by the underlying rules of the simulation. For instance, [50] allowed for agents to trip and fall and thus become static obstacles for others to navigate around, but the falling itself is a factor of the forces on the agent, not an individual trigger. Note that while any algorithm can have special events of this nature permitted through the use of triggering mechanisms, each trigger becomes a break from the normal operation of

the algorithm. Each special case adds to the complexity not only from its own existence, but how other agents will need to handle the agents in the "special" condition. Pelechano et al. handled this well by treating fallen agents as static obstacles.

Cognitive modeling aims to provide more "human-like" responses to perceptual input. Cognitive modeling is often considered with respect to high-level thinking such as goal selection [18, 71] but also exists for basic steering and navigation. In the 2010 work of Ondřej et al. [46], synthetic vision is used to grant agents the ability to navigate their environment using human-like anticipation of impending collisions. Agents visualize fields corresponding to the rate of change for other agents' bearings and use these fields to calculate a time-to-intersection. Adjustments to an agent's velocity only takes place when a collision is likely and imminent, much like human behavior. This framework is highly parallelizable and included the use of GPU programming for higher throughput, allowing 200 agents to be simulated in real time at the time of publication. More cognition-oriented features such as personality [15] add to the randomness experienced in a scene as different personalities tune low-level parameters to adjust how an agent moves.

2.2 Crowd Evaluation

To be used as a scientifically valid experimental framework, a crowd simulation needs to be evaluated for correctness. A properly vetted simulation can be used in training simulations requiring crowds of people and provide a higher sense of presence to the subjects. The crowd must behave naturally to not draw undue attention from what the trainer intends to be important. The problem then is how to define natural behavior and quantify deviation from it.

Until recently virtual crowds were validated from a subjective viewpoint. Re-

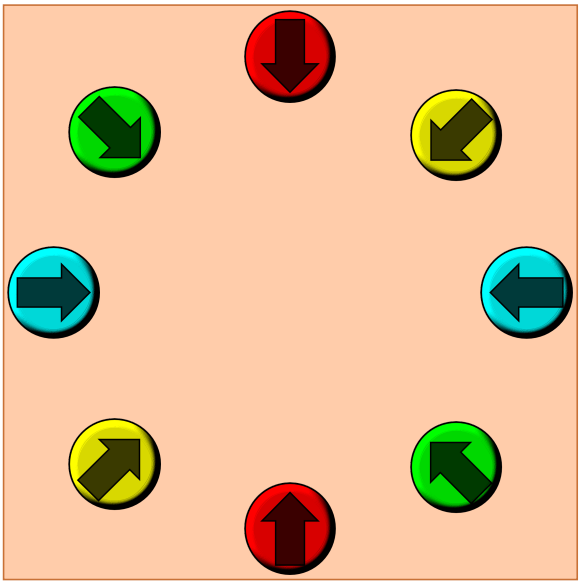


Figure 2.1: A common stress-test scenario for steering algorithms. While it will test the collision-avoidance of an algorithm and force a densely-packed scene, the situation itself is completely artificial. Opposing agents, illustrated here with the same color, have the goal of swapping positions.

searchers and reviewers must decide whether or not a system appears to recreate plausible human behavior in test scenarios. A general "ground truth" is unlikely to exist because of the number of uncontrollable variables to human behavior. In fact, humans do not even always act rationally; we make real mistakes and these mistakes would have to be accounted for by the performance metric used. This high-variance nature of human activity aside, one of the other main problems with subjective analysis of crowd simulations is that many of the most popular test scenarios are artificial examples. A classic scenario is that seen in Figure 2.1. Most subjective evaluation focuses on emergent behaviors as proof of recreating real behavior. Lane formation, the organization of the crowd into polarized lines of agents, is one of the most commonly sought emergent behaviors in crowd navigation literature but in the extreme can create degenerate behavior through strict rules forcing pedestrians to one side of a corridor, similar to highway traffic.

Lerner et al. used a data-driven evaluation algorithm in [35]. A database is constructed from manually tracked video of an intersection with sparse crowding and a walkway with dense crowding. This database is used to compare examples from the real world with agent states in the simulation using database queries. A metric from 0–100 is given to each agent based on how closely it matches normal behavior as defined by that behavior present in the database. Actual queries are formulated based on a density metric. Significantly low ratings can automatically indicate what the authors have deemed "curious" behavior, such as a person walking towards a cluster or odd evasive maneuvers.

The drawbacks of this early quantitative measurement is that anomalous behavior is dictated by the sample video used for comparison without any considerations for why the anomalous behavior exists. Furthermore, only densities are taken into account which is a very transient feature. Random wandering with the right random densities will pass the test. Most of these problems were alleviated in [36], where long-term decision metrics were introduced along with with the addition of proximity and flocking components to the metrics. A drawback of the system is that the flocking behavior requires a flocking video input for analysis; it is not automatically generated. Furthermore, the determination of what is natural and what is not depends only on the trajectories witnessed in the training videos. A similar data-driven approach is used as a component of [66], with the main difference being that input data is pulled using very specific tracking of known persons with known goals.

Some of the latest work has been from Kapadia et al. [31]. Their work takes the quantitative analysis a step further by also describing a state space for the scenarios that steering algorithms can encounter. This space can then be sampled to derive scenario sets to test a steering algorithm in a structured and rigorous fashion. Static obstacles and other agents are generated as a result of the parameters given to the

scenario generator and a run of the simulation is performed. Success or failure, due to agent stalemate or actual collision, will give a representation for the types of scenarios that a given algorithm is competent in handling and where it falls short. This work is beneficial for multiple reasons. First, we now have a way to test algorithms in a structured manner that does not rely solely on creating handmade scenarios to target specific behaviors as previously mentioned with the circle-crossing scenario. We can instead sample the scenario space using whatever strategy best analyzes the algorithm. Second, there is a way to get a clear idea of the conditions that will cause an algorithm to fail. Finally, it can be shown that no one algorithm can universally solve all agent navigation problems. By having a good sense of a failing scenario multiple algorithms can be used to create a more robust system.

2.3 Data-Driven Systems

Data-driven algorithms choose an agent's next action by matching the current environment against examplar scenarios. The data can be used directly by best-match as seen in Section 2.3.1, or machine learning algorithms can be used with the training data to create the final steering behaviors, such as Naïve Bayes in [43] and locally-weighted linear regression in Section 2.3.2. Computer vision has been used to drive agents based on real-world data [44, 45], which provides one of the best "gold standards" for training data. Information theory [68] has been used to analytically manipulate characteristics of agent behavior based on global trends to better tune overall behavior rather than relying on low-level agent-to-agent interactions. The data used for a system can come from the designer or user in lieu of real-world studies. Recently, the idea of a crowd as flowing agents has been used with user-generated navigation fields to give more control over the simulation itself. Rather than using

another algorithm to create a database or annotating video feeds of street corners, Patil et al. [47] use a discretized world with either precalculated or hand-drawn preferences for their agents to follow. Recent work [1] uses discretized pieces of real-world trajectories as the basis for navigation and manipulates these trajectories based on the possibility of future collisions.

2.3.1 Best-Match Search

One of the first data-driven pipelines [34] for steering has been an inspiration for our own. The authors manually tracked videos from an urban setting, and generated examples to populate a database. This database can be very large as each person in view can create one example in each frame of the video. At runtime, the example database is then queried by each agent based on its surrounding influences. A trajectory from the database best matching the situation is then returned and used by the agent.

To best mimic the behavior of a real person, the influences of a real person need to be modeled appropriately. The authors created a region of influence around an agent. Any person and static obstacle in this field is registered as a factor in the resulting trajectory. This brute-force approach creates an interesting, continuous state space, but results in a heavy computational burden for finding the best match and also requires maintaining the full collection of examples at runtime. Finally, as only external influences are considered the internal state of mind for the person cannot be a factor, which is detail that cannot be recovered from a video. We note that no current technology could use such input to derive a computer model, which means this is a problem inherent in the use of real-world data by its very nature. *Ad hoc* decisions and interaction with entities not in view of the video can cause otherwise unreasonable choices to be made when the tracked factors match. Thus a motivat-

ing phone call may be absent in the recreated behavior, causing a sharp trajectory change with no obvious cause. Due to the burden of finding the best-match in a continuous state space, runtimes were not generally favorable for interactive use as 3000 frames of simulation with 8 agents could be processed in 6 minutes, while 40 agents required a full hour for simulation.

2.3.2 Regression on Similar Data

Lee et al. [33] provided an alternative to brute-force matching at about the same time Lerner's group proposed their pipeline. The approach was reproduced and extended by Torrens et al. [65]. These works use multiple samples from a database to construct better results at runtime. Instead of a direct copy-paste trajectory, candidate trajectories are merged via regression into a final steering selection. Most importantly, discrete state spaces were used to handle the otherwise unlimited amount of possible configurations of stimuli. Each agent populates a feature vector to access a kd-tree of examples and extract the nearest neighbors.

Data found in the search is clustered by k-means into 3 clusters to get a high-level feel for the kinds of decisions available to the agent. For example, steering around a wall may have a right and left possibility and these should be separated for the regression step. Clusters are tested for suitability and the least fit is thrown out. A winning cluster is chosen from the remaining two either at random or by strongly favorable fitness to the current environment. This cluster selection is passed to the regression phase. Locally-weighted regression is used to take the data and calibrate the speed and direction of the agent.

2.4 A New Machine-Learned Approach

Currently, some of the most promising reproductions of human activity in crowd simulations have been the results from data-driven engines. This is not surprising as the data itself is an exact representation of human response to comparable stimuli. However, these simulations have been held back by the computational overhead of the very data they require. In particular, the works by Lerner and Torrens in Section 2.3 relied on maintaining a full database of samples, which reduces scalability over time as more information is collected and assimilated into the database. Part of the scalability issue lies in data beginning to contradict other examples, which was observed by Torrens et al. and relieved by the use of clustering. Small sampling of behavior simulated according to [55] was also tested as a basis for generating data. Their pilot use of an artificial algorithm as a source of training data has motivated our own work. Replication of the mechanical behavior of the source algorithm was achieved, but this dissertation extends the concept by using a nearly-optimal space-time algorithm as the source of our own training data. Optimal space-time planning has been used in recent work [37, 39] to produce solutions to complex navigational situations, but is constrained to only a few agents because of the heavy computational overhead.

This dissertation extends this idea of grouping otherwise contradictory data into a key component of the steering pipeline. Another data-driven method seen in [14] focuses on capturing the dynamics of the overall crowd, while we focus on the individual agents. This dissertation also extends the work of Becket [6] who previously showed a single-policy system is insufficient to cover general steering. Becket used online learning techniques to adjust the agents' behavior during the simulation, while we instead try to identify the multitude of policies as well as better for-

malize the concept of a steering policy. We use a two-layer hierarchy as a divide-and-conquer approach to the general steering problem, which is different from how others have used hierarchies to iteratively refine the planning process [20]. Hierarchical machine learning has also seen use for other AI-based problem solving [8, 13] and in robotics [64].

The potential of a data-driven approach in conjunction with the ideas of scenario space in [31] allow for stochastic sampling and coverage analysis of real versus synthetic data. Scenario space also motivates the definition of an algorithm's failure set to characterize the type of scenario the algorithm is incapable of handling. We use the complement of this idea in our work to better focus the machine learning we use for simulation. Similar to Ahn et al. [1] our basic algorithm [10, 11] is not collision-proof and our collision handling is left implicit in the training data itself.

Chapter 3

Steering Contexts

> Our job is to remind us that there are more contexts than the one that we're in—the one that we think is reality.
>
> *Alan Kay*

In this chapter, we define the concept of a steering context. This similarity-based grouping is performed to give a high-level perspective on the agents' current situations. For an agent to properly steer, a policy is required for each context. While these could be handwritten rulesets, achieving generality by identifying contexts and creating their corresponding policies for each is a task which quickly becomes workbound. We use machine learning to offset this burden and automatically generate models to serve as a policy for each context, which is further detailed in Chapter 5.

3.1 Terminology

We must first establish some terminology to serve as a basis for further exposition. A **scenario** is the global configuration of obstacles, agents, and agents' goals

in a virtual environment. The space of all possible scenarios is referred to as scenario space [31] and denoted $\mathbb{S}$. The high-dimensionality of scenario space makes it inherently intractable to exhaustively precompute solutions for all scenarios, thus requiring a more general model of steering behavior in the face of the diverse possible challenges. Each agent in a frame of simulation encounters its own **situation** S based on its local perspective. In essence a situation is a scenario transformed to a local space with respect to a particular agent. Situations which are similar, based on some quantitative metric, are grouped together to form a **steering context** or more succinctly **context**, C. Finally, a **steering policy** or simply **policy** is an approach to finding a steering solution to a situation. Multiple policies may be present in an overall algorithm to account for such things as special cases.

A key difference between our use of the term "context" and that of the related literature is we focus on the context as an egocentric concept. Each agent experiences its own context. Previous works have instead treated context to mean the characteristics of the group or entire population as a whole. While this better allows for a centralized solution to the steering of many agents, such a treatment of contexts does not take into account the fact that not all perspectives are the same in the simulation. In a densely crowded hallway, for instance, one person attempting to walk against a large crowd is experiencing a very different scenario than the others, even though from a macroscopic perspective one could just refer to the hall as densely crowded.

3.2 Decomposition of Scenario Space

We propose breaking the space of all possible scenarios, and by their related definition all possible situations, into more manageable groupings. This is possible by using the context concept to break apart scenario space. For a given feature space $\mathbf{F}^*$,

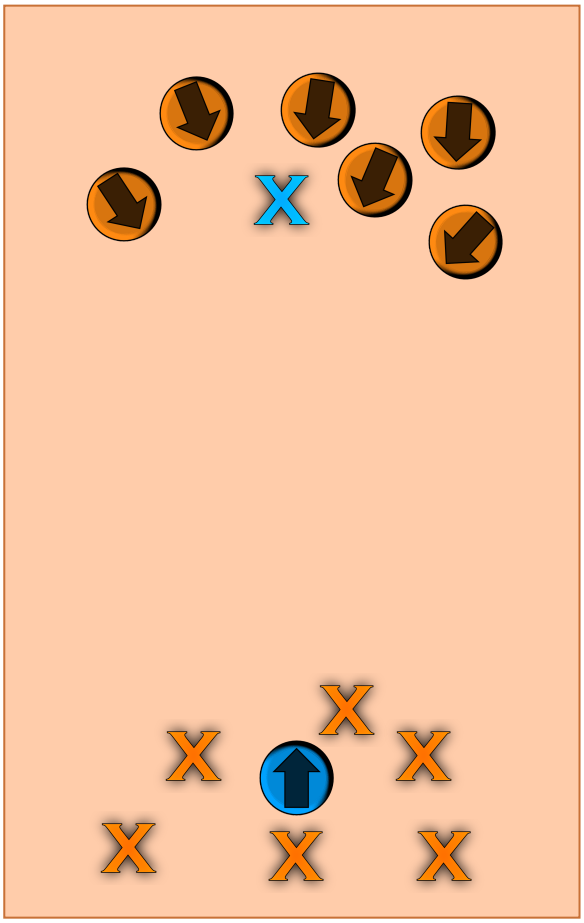


Figure 3.1: An illustrative example of steering contexts and situations. This bird's-eye view of the scene represents the scenario, which consists of one group of agents which must pass a single agent. The blue agent, however, only sees the scene as a situation where it must pass a group of other agents. The orange agents see the situation where they only have one agent moving towards them. Similar situations are thus grouped together into a steering context.

context space $\mathbb{C}$ is a projection of $\mathbb{S}$ onto the coordinate system of $\mathbf{F}^*$. As a result of the projection from $\mathbb{S}$ to $\mathbb{C}$ qualitatively similar scenarios will be mapped relatively closely compared to scenarios showing less similarity. Specific features from $\mathbf{F}^*$ thus become axes for a high-dimensional space, which allows values of the features to be used for classification. While potentially high, the number of features used can be smaller than the total number used for defining all scenarios and thus the overall dimensionality advantageously decreases.

Context space can be split along features' values to form a classification of specific contexts. Steering contexts could also be derived from the features of $\mathbb{S}$ but the "curse of dimensionality" [7] makes the process much more difficult as even similar situations are increasingly sparse as the number of dimensions increases, limiting the usefulness of grouping situations quantitatively. Similar situations should be manageable with a similar approach, motivating our identification and use of a collection of policies to handle subsets of situations rather than confronting the general steering problem with a single-policy approach.

Formally then, an individual context $C_i \subseteq \mathbb{C}$ is defined in Equation 3.1 with respect to the success of steering policy i in handling a group of situations. A policy is successful if it can produce a valid action from the provided action space $\mathbb{A}$ for the situation, which is one where a collision does not occur and the overall situation progresses towards its final frame. This in theory permits some agents to stop moving in some situations provided the result is other agents continue to move, which could be seen in a scenario involving two agents approaching a narrow doorway which requires one to yield to the other.

Thus a simulation can be considered a sequence of situations and actions with some transition function $\delta\,(S, a)$ moving from one situation to the next. Since a situation is the entire scenario transformed to an agent's local perspective, a change in

the overall environment can change the agent's situation even if they are a passive, stationary participant. Two situations can be considered equal if the agents, obstacles, and goal are all in the same location and orientation, and we denote the set of all previous situations as S^{-}. Past situations need to be prevented as they will induce a cycle into the agent's behavior. It is not sufficient to only ensure that $\delta(S, a) \neq S$ as simple oscillation would meet the criteria but not lead to the agent reaching its goal.

$$C_i = \left\{ S \in \mathbb{C} \mid \exists a : \langle \mathbf{f}, a \rangle, \mathbf{f} \in \mathbf{F}^i, a \in \mathbb{A}, S \notin S^{-} \right\} \tag{3.1}$$

A situation is guaranteed membership in at least one context because in the worst case, it could have a special-case policy defined for it. Additionally, the steering policy is an analog for the transition function δ, which allows the objective of policy creation to be framed as one of finding the appropriate function. A main application of machine learning is fitting functions, making the technique especially suitable for automating the derivation of specific policies for each context. Note it is also permissible for a situation to be a valid member of multiple contexts, as several policies may produce valid actions for the same situation.

3.2.1 Computational Impacts

We have used the concept of scenario space to initially frame the concept of steering contexts, but we can also use the contexts to redefine scenario space in terms of a union of sets, seen in Equation 3.2. This alternative definition yields important insight into not only the data-driven process of this dissertation but also the pursuit of any generalized steering algorithm. The general applicability stems from the fact that policies are not specific only to our framework. This means any algorithm

with different approaches, such as activating more costly collision avoidance in the presence of other agents, exists as a collection of policies and thus contexts.

$$\mathbb{S} = \bigcup_i C_i \tag{3.2}$$

The set definition of $\mathbb{S}$ presents contexts as equivalence classes of all possible situations. Since they are equivalence classes, by necessity an equivalence relation must exist. Such a relation exposes another function-based aspect of steering contexts, and again motivates the use of machine learning. By fitting a function for the equivalence relation, we can attempt to classify a situation's context. An equivalence relation should partition scenario space into disjoint subsets, but recall we stated in Section 3.2 that multiple policies could be viable for a situation. This can be trivially remedied by picking an arbitrary candidate context for the scenario to be a member.

Steering algorithms all view the virtual environment in terms of its features in some space, and thus strive to identify and handle contexts. Due to the fundamental set theory nature of situations and contexts two very desirable aspects of formulating a generalized steering algorithm, optimality and coverage, are difficult to confirm. Both of these are $\mathcal{NP}$-Complete set problems [32].

Optimal Contexts

The ultimate challenge to a steering algorithm is to handle any situation in the best possible way. To achieve such a goal, a steering algorithm needs to handle every situation by the policy best suited to it. In this case, the underlying contexts theoretically need to partition scenario space according to optimal contexts. As Equation 3.1 allows for situations to exist in multiple contexts, one would need to select only those contexts optimally solving all situations.

The problem then is one where the input is a collection of subsets for some master set, and task is the selection of those subsets which partition the full set. This is a direct application of the exact cover problem, one of the classic $\mathcal{NP}$-Complete problems.

Number of Contexts

With optimality of context selection out of computational reach, the next analytical question we would want to know is how many contexts are necessary for full coverage of scenario space. Full coverage means that all situations are handled, which as previously mentioned could be done with a situation-per-context approach although not in a practical manner. We only care that all situations are dealt with, not that they are optimally solved.

However, finding the minimum number of contexts, and thus the minimum number of policies, is also $\mathcal{NP}$-Complete. The problem consists of selecting the minimum number of subsets from a collection such that all elements of the full set are represented at least once. This is a direct application of the set cover problem.

3.2.2 Context Approximation

Although steering contexts yield disappointing conclusions for optimality, these same conclusions are applicable to any steering algorithm consisting of multiple policies. A specifically context-cognizant approach to steering still has the advantage of better structure and scalability. Furthermore, by using a suite of data-driven techniques we can strive to approximate the underlying context space. This more purposeful exploration of all possible situations opens the door for more strategic development of algorithms, while also creating a framework which is more modular and extensible by allowing future additions and modifications of some policies without

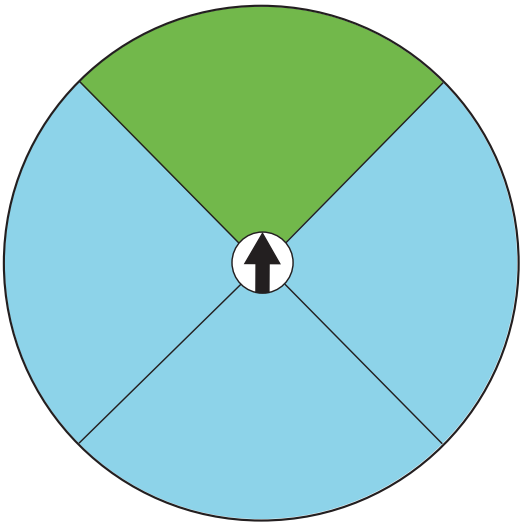


Figure 3.2: $\mathbf{F}^*$ for our set of contexts. The region ahead of the agent checks for the presence of obstacles. All four regions track the density of nearby agents as well as the general flow of traffic.

necessarily affecting the rest.

3.3 Context Identification

We divide the possible scenarios by qualitatively different pedestrian traffic patterns inspired by those observed in real pedestrians. Another strategy would use unsupervised machine learning to cluster the scenarios. Our objective then would be to approximate contexts based on computable separability rather than human-centric qualitative differences. This alternative approach is explored in detail in Chapter 6.

3.3.1 Intuitively Derived Contexts

For our first set of contexts we selected typical traffic patterns seen in pedestrian activity. A feature space visualized in Figure 3.2 was also developed to characterize these patterns[1]. This space's origin is locked on the particular agent currently perceiving the virtual environment. The region around the agent is divided into four sec-

tions, one for each cardinal direction. Furthermore each section tracks three pieces of data: the density of neighbors in that area as well as their general velocity relative to the agent's own movement divided into heading and speed. The northern section also acknowledges the presence of static obstacles in the scene. This gives us a total of 14 dimensions for $\mathbf{F}^*$.

There are four primary types of steering context in this set based on traffic pattern.

`Clear`

Very sparse neighbors, the agent is predominantly free to move.

`Oncoming`

Nearby neighbors are generally moving towards the agent from the forward direction.

`Crossing`

Nearby neighbors are moving perpendicular to the agent's own path. This could be from the agent's left or right side.

`Chaos`

Very dense neighbors with no obvious pattern to their movement with respect to the agent.

Additionally, multiple divisions of `Oncoming` and `Crossing` are defined. These subdivisions take into account the density of the scenario including neighbors traveling along with the agent rather than just those causing potential steering challenges.

`Light`

2–4 agents total in the scenario's environment.

`Medium`

4–7 agents total in the scenario's environment. A "noise" agent not fitting the traffic pattern is permitted to promote generalization.

`Heavy`

7–10 agents total in the scenario's environment. Depending on the total population, 1–2 "noise" agents are permitted.

`Group`

Scenarios where two groups of approximately equal size are navigating the environment, with the agent part of one of the groups.

`Winning-Side`

The mirrored scenarios compared to `Medium` and `Heavy`. The agent is a member of the large group in the environment rather than in the minority.

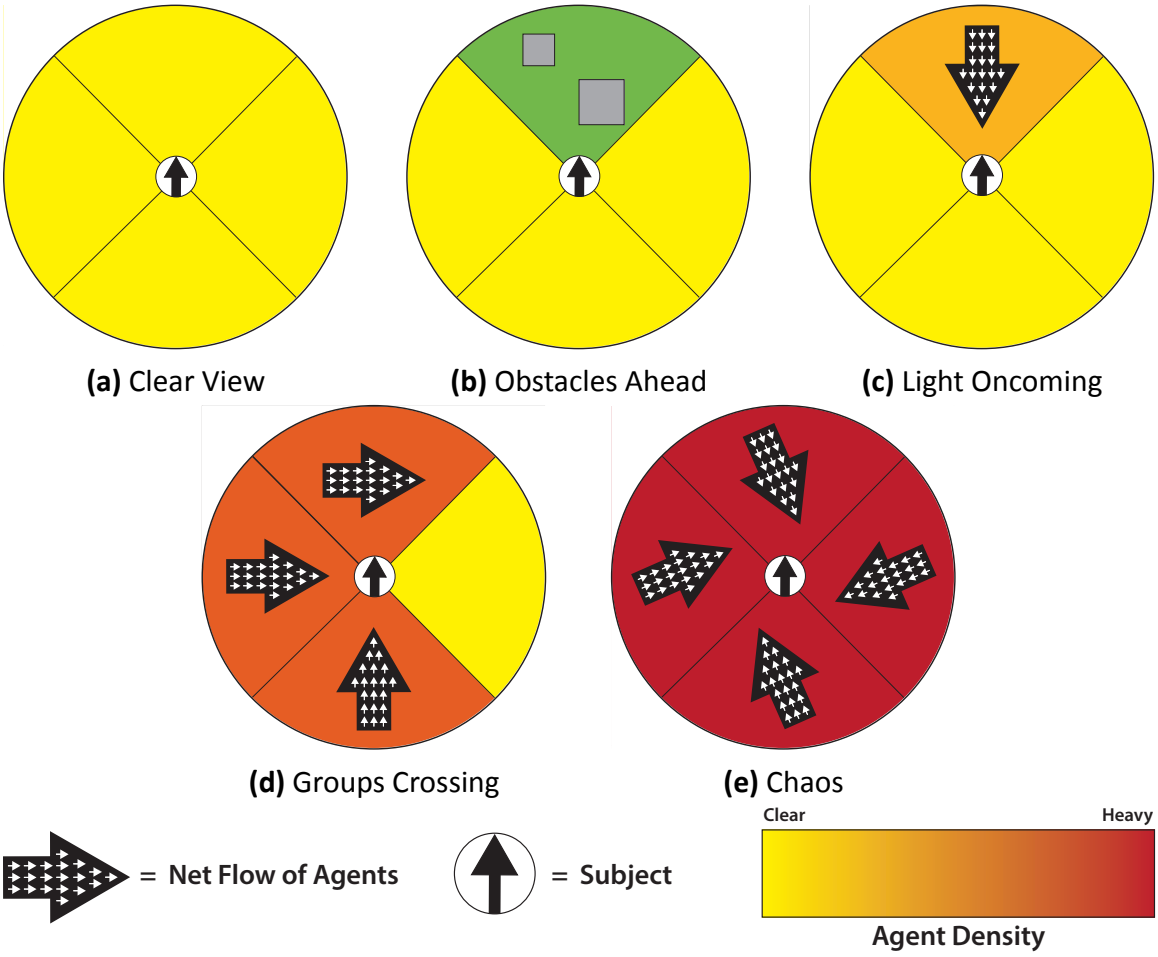
In total, these variations provide us with 12 steering contexts, which are assigned numerical IDs in this dissertation. The full matching of IDs to context definitions is provided in Appendix A. Duplicate contexts with the added presence of static obstacles raise the total to 24. Examples of the contexts from this section and how they are represented in the feature space are provided in Figure 3.3 with a full index in Table 3.1.

Notes

[1]The symmetry of this space is a convenience for prototyping a proof-of-concept. Any other geometric form could be substituted. We do not try to optimize over this structure in this dissertation.

Context ID	Obstacles	North		South		East		West	
		Flow	Density	Flow	Density	Flow	Density	Flow	Density
0	Yes	Neutral	Light	Neutral	Light	Neutral	Light	Neutral	Light
1	Yes	Towards	Light	Neutral	Light	Neutral	Light	Neutral	Light
2	Yes	Towards	Medium	Neutral	Light	Neutral	Light	Neutral	Light
3	Yes	Towards	High	Neutral	Light	Neutral	Light	Neutral	Light
4	Yes	Towards	Medium	Towards	Medium	Neutral	Light	Neutral	Light
5	Yes	Towards	Light	Towards	High	Neutral	Light	Neutral	Light
6	Yes	Neutral	Light	Neutral	Light	Towards|Away	Light	Away|Towards	Light
7	Yes	Neutral	Light	Neutral	Light	Towards|Away	Medium	Away|Towards	Medium
8	Yes	Neutral	Light	Neutral	Light	Away|Towards	High	Away|Towards	High
9	Yes	Neutral	Light	Towards	Medium	Away|Towards	Medium	Away|Towards	Medium
10	Yes	Neutral	Light	Towards	High	Away|Towards	Light	Away|Towards	Light
11	Yes	Towards	High	Towards	High	Towards	High	Towards	High
12	No	Neutral	Light	Neutral	Light	Neutral	Light	Neutral	Light
13	No	Towards	Light	Neutral	Light	Neutral	Light	Neutral	Light
14	No	Towards	Medium	Neutral	Light	Neutral	Light	Neutral	Light
15	No	Towards	High	Neutral	Light	Neutral	Light	Neutral	Light
16	No	Towards	Medium	Towards	Medium	Neutral	Light	Neutral	Light
17	No	Towards	Light	Towards	High	Neutral	Light	Neutral	Light
18	No	Neutral	Light	Neutral	Light	Towards|Away	Light	Away|Towards	Light
19	No	Neutral	Light	Neutral	Light	Towards|Away	Medium	Away|Towards	Medium
20	No	Neutral	Light	Neutral	Light	Away|Towards	High	Away|Towards	High
21	No	Neutral	Light	Towards	Medium	Away|Towards	Medium	Away|Towards	Medium
22	No	Neutral	Light	Towards	High	Away|Towards	Light	Away|Towards	Light
23	No	Towards	High	Towards	High	Towards	High	Towards	High

Table 3.1: Parameters which define the 24 contexts we use to prototype our pipeline.



(f) Symbols Key

Figure 3.3: Examples from our set of contexts. Net flow is represented by the arrow in each region, density of the region is depicted by darker shades of red, and obstacles are gray boxes. Each of these contexts was stochastically generated with overlap in the permissible values for density. Chaos was generated randomly without regard to any structure as seen in the other contexts. We used a total of 24 contexts.

Chapter 4

Synthetic Data

> Ours may be the last generation that can see a difference between real and virtual things.
>
> *Norman I. Badler*

In this chapter we explore some of the sources for training data available to crowd simulation. Real-world data is a commonly used source when using data-driven techniques, but collecting that data has challenges, which we discuss. Regardless of whether or not the pedestrians know they are being observed the subsequent results are likely flawed. In contrast, we use synthetic data in an attempt to mitigate those flaws.

4.1 Nature of Real-World Data

Real-world data is limited in efficacy due to "noise" in observations as well as the complex nature of the system—humans—being observed.

4.1.1 Collection Techniques

While the specifics of real-world observations vary widely, ultimately such experiments fall in one of two categories based on the awareness of the pedestrians that surveillance is occurring. Ultimately these two options available to researchers create a need to compromise between the results' robustness and its accuracy.

Overt Data Collection

Possibly the most direct way to observe pedestrian behavior is to have volunteers "solve" situations presented by an experimenter. The key benefit to being overt with data collection is the essentially complete control researchers have over the circumstances. Scenarios key to the intended purpose of the simulation can be replicated for the participants, complete with obstacles arranged and other participants given specific targets to create the necessary situation. Thus over time a more robust set of data can be compiled with common situations including queuing, groups of pedestrians crossing, and groups of pedestrians passing each other. Additional consideration can be given to the sensors used to collect the data itself such as unique clothing for each individual or other features to enhance tracking of a particular pedestrian over time.

Such an approach to observing pedestrian behavior is not without its limitations. The participants in these experiments are particularly vulnerable to the observer effect. They can become self-conscious about their own behavior, being more careful to exhibit what they think is the expected result. The net result of this effect is a reduced accuracy provided by the observations. There is also a great amount of overhead in adding new data to the set. Volunteers must be found and coordinated, a suitable recording area located, and the scenarios desired must be clearly defined

for the participants. This overhead is not only prohibitive for adding new scenarios, but even repeating previous scenarios. Such repetition is scientifically desired for reducing bias and confirming previous results. While the overhead is unfortunate and inconvenient, the impacts are secondary to the inherent tradeoff of reduced accuracy in favor of robustness.

Covert Data Collection

Pedestrians are not difficult to locate; people walk along streets and through stores all the time. Security cameras already monitor these pedestrians with minimal intrusion and people have acclimated to these recordings taking place. Researchers can gather data from these low-impact views to collect very accurate data. In addition to existing monitoring systems, researchers have set up video surveillance in targeted locations to observe different types of crowds. Since there is no direct intervention on the part of researchers, this technique offers low overhead for the initial data collection, as the camera simply needs to record the location even without the researchers present.

While more accurate in terms of natural responses to stimuli, the lack of control offered by such a covert system reduces the robustness of the resulting data. We are restricted only to behavior which coincidentally takes place, diminishing the likelihood of a variety of observations. In particular those situations which could be more important from a simulation standpoint, such as a building evacuation, may never be witnessed in the time window provided to the researcher.

4.1.2 Processing Data

Regardless of method, the data must be processed into a form suitable for application to crowd steering. Individuals must be identified and tracked, as well as the

factors affecting their actions. Algorithms have been shown [58] that are capable of automating this process but they are imperfect. For example, the algorithms may be unable to differentiate between a person and their reflection in a specular surface, leading to overlapping routes and conflicting data.

Manually tracking of the data is a common solution, but suffers from scalability. When data is collected overtly smaller groups are used, making the task tedious but within the capability of a single person. Busy urban areas, however, lead to more demand on the observer to account for large numbers of individuals as missing any is not only a lost trajectory for the database, but also a missing influence on others' actions. Multiple observers can be used to reduce an individual's workload, but there remains a fundamental disconnect between the data and its origins in a recording.

4.1.3 Noise in the Data

In either paradigm of data collection, there is noise inherent in the data. Extra factors not being monitored by the observers can influence a person's steering decisions, leading to a mismatch between the stimuli being used to later replicate the behavior and that which actually caused it. Loss of focus through distracting thoughts or attention-grabbing environmental factors can cause delays in processing the current situation, such as when two pedestrians nearly—or actually—collide because neither saw the other. Sound is usually not considered and is particularly prevalent during covert data collection. Individuals' personalities, concept of personal space, cultural background and norms, and more can also affect a myriad of variation in steering behavior.

When these factors are not being tallied and accounted for in the eventual data-driven reconstruction of behavior, they become noise in the system. Consequently as these factors contribute more or less, they insert randomness into the results. This is

the equivalent of a signal-to-noise ratio (SNR). The signal consists of the important factors leading to steering behavior, while the noise also affects the results. To improve the SNR of the source data we need to either carefully and thoroughly learn as much about the noise as possible to filter it out, or we can generate data where the noise is absent by algorithmically creating synthetic samples.

4.2 Algorithmic Alternative

In this dissertation, we explore the efficacy of using the trajectories of a crowd *simulation* as a source of synthetic data. The simulation used serves as our **oracle algorithm**, since it can be consulted for the solution to an arbitrary scenario. Using simulation in this manner can help mitigate the issues with real-world recordings because we possess a crowd where we control the initial conditions but avoid introducing an observer effect. New results can be generated as needed with minimal overhead and these results do not require potentially lossy tracking. The use of crowd simulation to create an alternative to real-world data has been used for training computer vision [4], and here we use the same concept to take a slow algorithm and train faster models on that algorithm.

4.2.1 Stochastic Scenario Generation

In order to keep completion time of the oracle practical, we used a relatively small virtual world in our stochastic sampling. An agent is placed in the center of the space, and the virtual environment is divided into five key sections with respect to this centrally placed subject. Four of the regions are seen in Figure 4.1. By identifying these regions as North, South, East, West, and Central to the subject, we can randomly generate scenarios with populations in each region. Additionally, we can set agents'

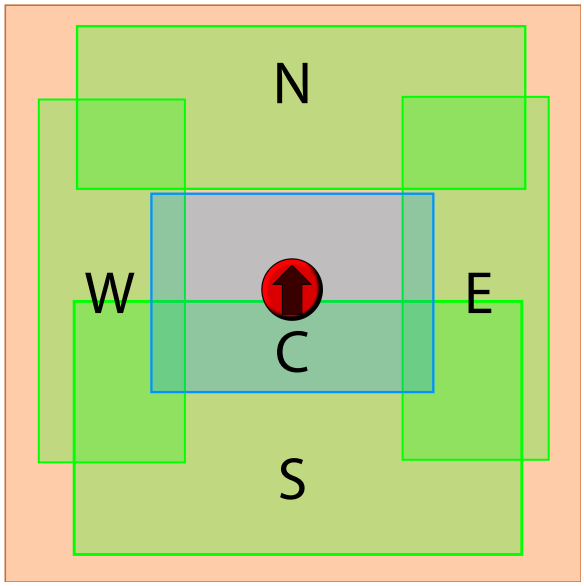


Figure 4.1: A relatively small virtual environment was subdivided into five regions for North, South, East, and West. The "subject" agent, which will be sampled to generate training data, is placed near the center and its goal placed anywhere to the North. Depending on the parameters needed for the simulation, agents were randomly generated in the other regions with goals set as-needed. The Central region was only used to allow the subject to be initially placed with some variation.

goals inside other regions to create likely traffic patterns. For instance, the subject can have the goal to move into the North region, while a group of agents in the North section have the task of navigating to goals in the South. Obstacles can be placed throughout the environment.

As was further explored in Section 3.3.1, the division of the virtual scene into logically separate areas was based on an intuitively-derived set of initial contexts. We wished to use these areas to force random generation according to some overall patterns which we identified as likely steering scenarios for an agent to encounter during simulation. We also introduced some imperfections during the random scenario generation to prevent the resulting data from being too clean. For instance, while generating random scenarios where the bulk of the agents in the scene are moving perpendicular to the subject, we allowed for occasional agents to also be walking

elsewhere in the scene.

Once defined by our stochastic scenario generator, the virtual world's division into sections plays no further role in the oracle or data-driven algorithms we employ in this work. Similarly, it is possible to introduce new scenarios for the oracle to simulate by handcrafting the situation we wish to see. For bulk gathering of data with the widest range of coverage, this randomized approach is ideal and results in far less manual overhead.

4.2.2 Oracle Algorithm

Our oracle algorithm is based on a memory-bounded A* planner with a discrete footstep action space similar to the action space in [61]. We chose a discretized footstep action space so our machine learning can use classifiers instead of being constrained to regression. When the oracle is run on the generated scenarios, each agent uses the memory-bounded A* planner to calculate the optimal path from its current location to the goal. The bound on the memory is raised if a path is not found, as a last resort Iterative Deepening A* (IDA*) is used. The oracle planner's overall algorithm is given in Algorithm 1, and the heuristic used is in Equation 4.1 and is based on the distance to the goal and average expected energy cost to reach that goal.

$$h(\mathbf{p}, \mathbf{g}) = \frac{\|\mathbf{p} - \mathbf{g}\| \cdot \text{energy}_{\text{avg}}}{\text{stride}_{\text{avg}}} \tag{4.1}$$

Each agent has full knowledge only of the obstacles and agents within its bounded field of view. Since other agents may enter or leave this field of view, each agent must monitor its path for new collisions and invoke the planner again if such a problem is found. We chose this limitation on the oracle for a couple of reasons. First, the data would ultimately be used with respect to a set of features, covered in detail in

Algorithm 1: Oracle Planner

Data: Start, goal, low memory bound, max memory bound, memory increment size.
Result: The path from start to goal.

```
1 for i ← memMin to memMax do
2     path ← BoundAStar (start, goal, i)
3     if path.size = 0 then
4         i ← i + memBlock
5     else
6         return path

  // Could not find path with BoundAStar
7 path ← IDAStar (start, goal)
8 return path
```

Section 3.3.1 and Section 5.2.1. The second, and related, reason was that these feature would be inspired by the perceptual capabilities of humans and thus may lead to more realistic responses to stimuli. These limitations are reasonable, as the oracle-based agents would otherwise be capable of unnatural abilities such as veering to avoid neighbors while they should be obscured by an obstacle and thus unknown to the agent.

The simulations using the oracle are recorded for later extraction of training samples. As the oracle does not use any feature spaces, the same oracle recordings can be used to extract data with different feature spaces, allowing for future exploration of such possibilities. We extract a state-action pair $\langle \mathbf{f}, a \rangle$ where $\mathbf{f}$ is a vector from feature space $\mathbf{F}$ and a is the parameters of the agent's current step, and use it as a sample for training.

The oracle essentially serves as a high-fidelity precomputation engine by generating recordings which can be used as needed for later work. While the oracle can be time-consuming to run due to its goal of near-optimality, it is still less challenging than using a corresponding set of volunteers or monitoring a population for rare

events. If the amount of memory demanded for finding a nearly optimal solution proved to be too high we could use IDA*, however in practice this was never required for the scenarios we generated, possibly due to the constrained size of the environment.

4.2.3 Data Collection

We generated 5,550 scenarios for each of our initial 24 steering contexts for a total of 133,200 scenarios. For the purposes of machine learning, further detailed in Chapters 5 and 6, up to 2,500 were used as training data for any given context based on the success and runtime of the oracle algorithm. Approximately 10 seconds of virtual footage was used from each scenario. This resulted in hours of automatically tracked, reusable data with no need to collect and organize volunteers.

Chapter 5

Initial Application

...faith apart from works is dead.

James 2:26, ESV

We used our initial contexts and synthetically generated training data to create crowd simulations and analyze the preliminary results. This implementation of context-sensitive steering served as a proof-of-concept and motivated further improvements.

5.1 Framework

Our framework for the integration of various contexts into a unified steering algorithm is illustrated in Figure 5.1. This framework takes on the form of a pipeline and is predominantly composed of precomputational steps. First the necessary steering contexts must be defined. With these definitions, a targeted sampling of situations can be performed as discussed in Chapter 4. These situations are solved by the oracle algorithm and recorded. These recordings are then sampled with respect to feature spaces for use with machine learning which produces the actual state-action pairs for

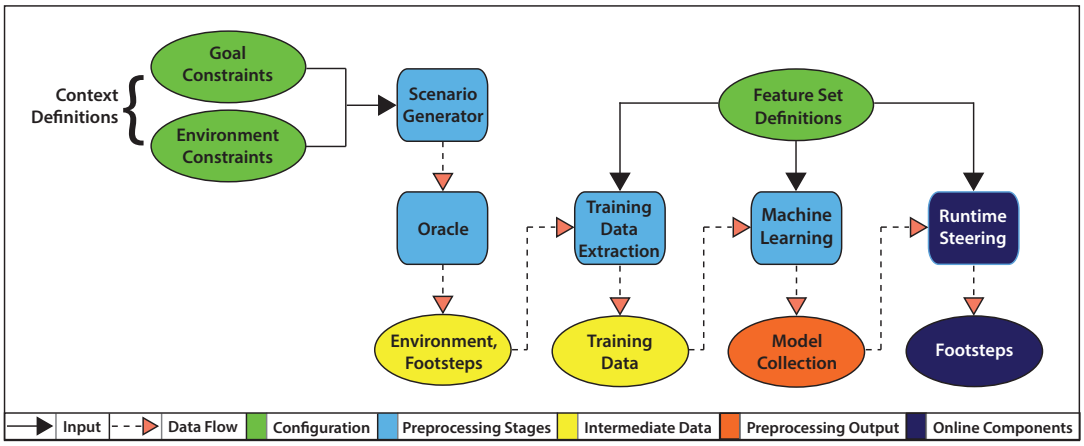


Figure 5.1: Our framework for using steering contexts to develop a machine-learned model for use at runtime using a pipelined approach. The majority of the pipeline is offline processing. A collection of models is trained on data extracted from an oracle algorithm's solution to steering situations, which are stochastically generated. Then each model is a boosted decision tree with its own specialization. The action space consists of footsteps as an advantageous discretization which permits direct control and modeling of human locomotion.

the data-driven technique. Models are fit to the pairs for subsequent use at runtime.

5.2 Initial Machine Learning

Our contexts provide a natural way to form a two-level hierarchy of models. The first level is used to select which context the agent is currently experiencing. This allows the subsequent selection and use of the specialized classifier which represents the policy for that particular context. This is in lieu of fitting a single, monolithic model to all the training data generated by the oracle.

Avoiding the requirement that the learned policy be a monolithic, universal solution has several key benefits. First, the policies can be simpler and thus easier to fit. Second, we avoid the catastrophically high dimensionality common to such approaches, which are held back by all the factors that can influence every potential action. Finally, we do not need to relearn the entire system to assimilate new data.

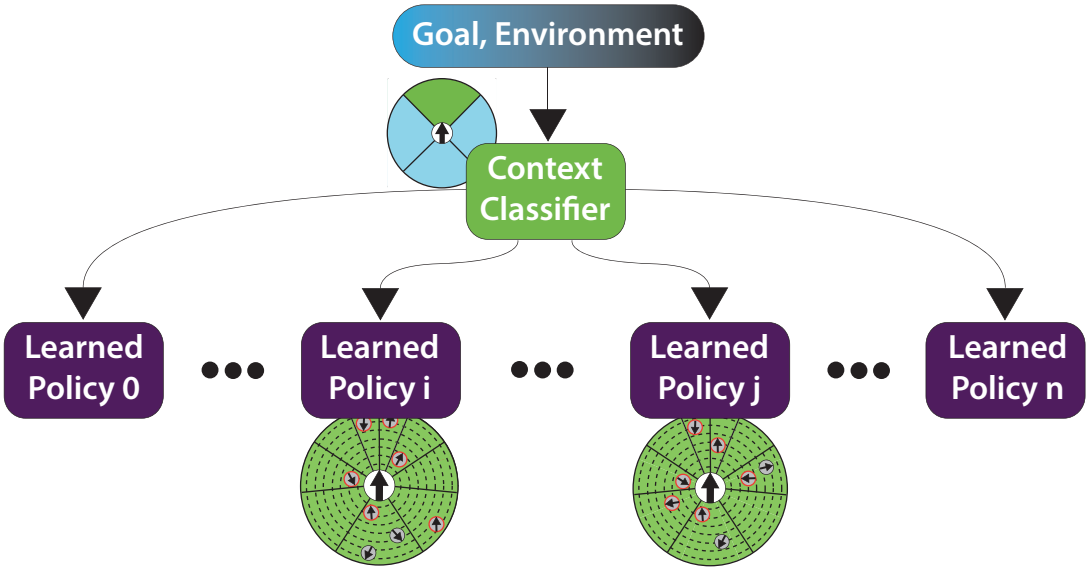


Figure 5.2: The multilevel decision trees used by our models. At runtime the agent gives the model information about its current goal and environment in local-space. This data is used to calculate **f** for each model used. First the context classifier informs the agent of its current context, and the corresponding policy is used to determine the next footstep.

By using one model to select more specialized models, new data requires only the specialized model it belongs to be relearned. Even the creation of a new context only requires the top-level model be recomputed while the other models are still valid and will not be harmed by potentially contradictory data.

This framework is agnostic to the specific learning algorithms used at the different levels of the hierarchy, and different algorithms can even coexist on different levels of the hierarchy if particular contexts are better handled by different models. We have chosen to use two levels of boosted decision trees [52] for our instantiation of the pipeline based on the similar problem domain of [64] that showed success for learning different policies that both classified different types of soccer behavior and could be used to decide the actual action itself.

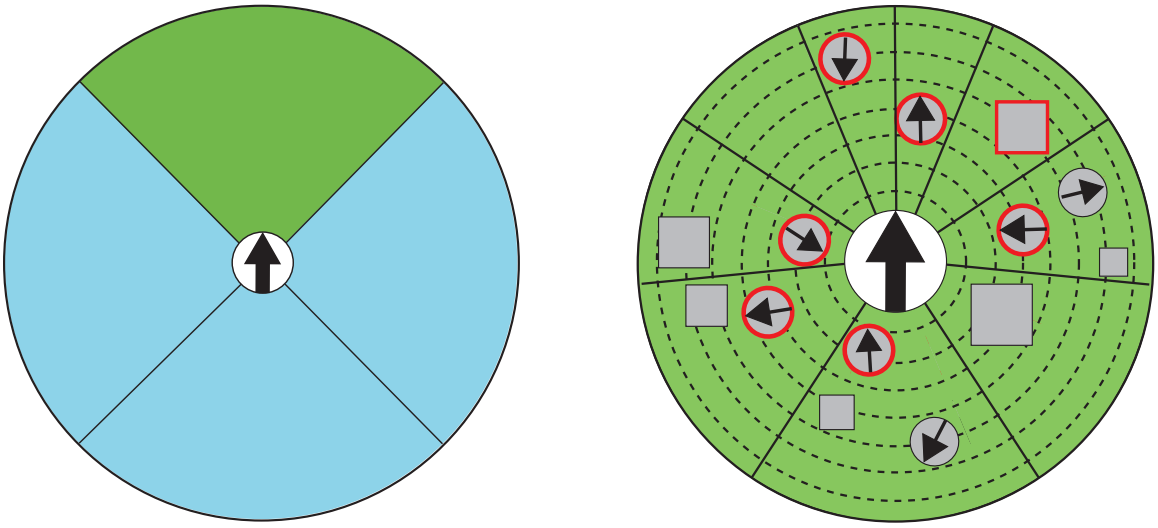
Each of our policies consists of two boosted decision trees; one for each foot. We use a Windows port of the GPL release of the C5.0 decision tree system [56]. We

chose ten trees as the amount of boosting empirically based on cross-validation. In total 2500 scenarios were sampled from each context and each scenario was generated with respect to a central agent, which provided a variable number of steps per scenario. These steps then became the situations representative of the context for the specialized classifier. A context classification sample was only generated for the first five steps of each recording due to the total number of scenarios that were sampled, all of which supplied data to the context classifier.

5.2.1 Feature Spaces

We define two orthogonal features for the area in each cardinal direction about the agent for a total of 8 features, with a ninth feature special to the region ahead of the agent. The components of each area are agent density and the net flow of agents in that area, with the area directly in front of the agent detecting the presence or lack of obstacles. **Agent density** is a rough approximation of overall crowding in the cardinal directions and includes obstacles. **Net flow** is the average velocity direction of agents in a particular area. This helps determine whether or not the general crowd is moving with or against the agent, which requires different care for such things as collision avoidance.

Our feature space for learning specialized policies is based on a circular neighborhood about the agent with discretized wedges that track the nearest agent or obstacle in that region. Our feature spaces can be seen in Figure 5.3 and are in part inspired by the state spaces of [33, 65]. In particular, the context classifier's state space is built of two values for each of the four regions and an additional value denoting the presence of obstacles in front of the agent for a 9-dimensional vector. The specialized feature space is a 29-dimensional vector broken down into three values for each slice: the distance, speed, and orientation of the nearest entity. The distance to the goal and



(a) $\mathbf{F}^*$ (b) $\mathbf{F}$

Figure 5.3: The feature spaces used in our pipeline, where other agents are circles and static obstacles are depicted as boxes. $\mathbf{F}^*$ is used by the context classifier to dynamically choose the best model based on high-level features, while $\mathbf{F}$ is used to choose the agent's next step based on the local neighborhood.

its orientation are the final two values.

A data-driven approach relies on the quality and coverage of its training samples. Real-world data is often used as a source because humans empirically solve any presented steering challenges and we wish to create virtual representations of humans. However, we cannot completely control the steering scenarios or know all the variables in the decision-making process of the people observed. To enforce artificial limitations on the scenarios would impact the integrity of the data through the influences of the observer effect. Second, we have no way of knowing *a priori* whether the data set collected has adequate sample coverage for the situations the agents will need to handle. The problem of this potential incompleteness is compounded by the overhead—or impracticality—of collecting additional data. For these reasons, our pipeline uses synthetic data from which we can be conveniently gather additional samples and know all the influences in advance.

5.2.2 Runtime

At runtime the agent generates feature vectors corresponding to both the context classifier's feature space and the corresponding specialized model's feature space and receives parameters used to derive its next footstep. These parameters include a relative offset and rotational angle to the next step's location, while specifics such as stride length are calculated on-the-fly based on the agent's inherent characteristics. This step is validated and if found to be unfit, a default "emergency action" takes place, wherein the agent immediately stops. This allows the agent to try again after a short cool-down period. This safety net was implemented to account for the worst-case where a returned action is outside of the parameters permitted by the agents' walking such as two steps in a row from the same foot or too wide a turn. The models cannot be expected to be 100% accurate, which is the source of these potential errors. Pseudocode for the agents at runtime is listed in Algorithm 2.

As shown in Section 3.2.1, it is $\mathcal{NP}$-Complete to know if our contexts cover all possible scenarios. Furthermore, decision trees are susceptible to high variance depending on the dataset we generate through our stochastic sampling. This causes uncertainty in the decisions our agents will make. We account for this uncertainty through the use of a confidence threshold defined by the C5.0 algorithm. This rating is roughly defined as the number of correct classifications made by the leaf nodes divided by the total number of classifications made by the same node, making it a static quantity once the tree is learned. If the confidence threshold is not met by the classification the agent stops with the ability to resume as conditions change. This confidence value is not a direct reflection on the technique itself, but is instead heavily affected by pruning the decision trees to yield a more general model.

Note in Algorithm 2 there is no explicit collision detection or avoidance. In our system, runtime collision detection and avoidance is handled implicitly through the

Algorithm 2: Agent Decision at Runtime

Data: The environment with respect to the agent.
Result: The next footstep action.

```
1 fStar ← ObserveEnvironment ()
2 contextID ← ContextClassifier (fStar)
3 f ← ObserveLocalSpace (contextID)
4 action ← Classifier (f,contextID)
5 if action.confidence ≤ threshold then
6 |  action ← StopInPlace
7 return action.step
```

training data itself. This is different from other techniques such as [34] where training samples are used but thorough handling of collisions is required.

5.3 Performance

The following results were generated on a desktop PC with 16GB of RAM, NVIDIA GeForce GTX 680 graphics card, and Intel Core i7 860 quad-core processor supporting eight hardware threads running at 2.8GHz.

5.3.1 Classifier Accuracy

Figure 5.4 plots the error rate for the classifiers used in our experiments with varying amount of training data. Simulations were run using models trained on amounts of data ranging from 100 to 2000 scenarios per context. A separate validation set of 200 scenarios per context was kept back to calculate the error rate of the resulting trees. Error rates were high but did decrease as data size increased, showing improvement in generalization and not simply noise.

Additionally, each context used approximately 12 of the possible step selections

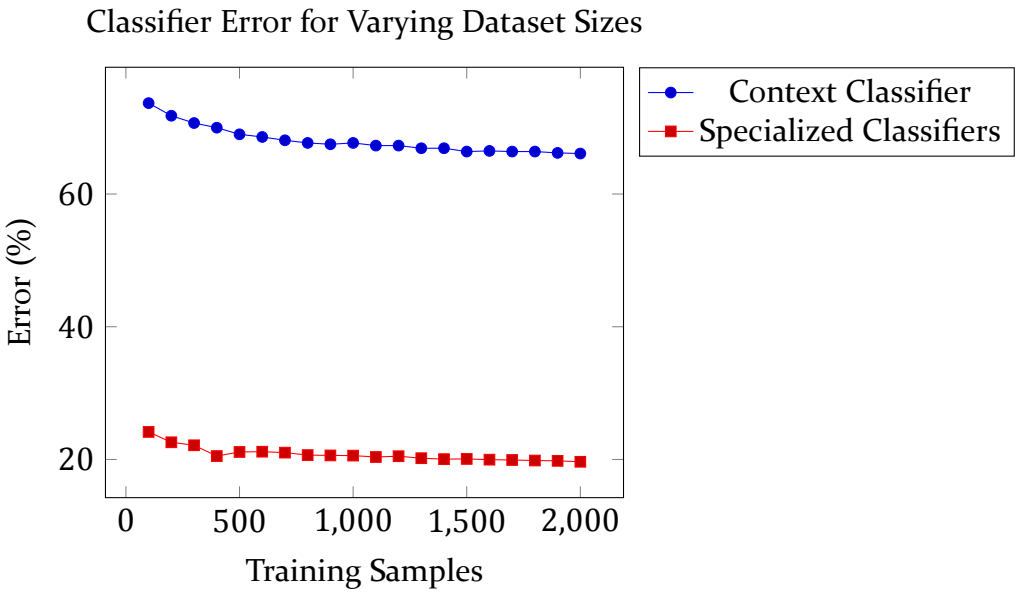


Figure 5.4: Classifier error rates as training data increases in quantity. While the context classifier has a high error rate, a 96% error rate is random chance given the large number of classes to choose from. This shows both generalization and quality which far surpasses random guessing.

which gives a random guess accuracy of only 8%. We far surpass this level of accuracy. Furthermore, random guess accuracy for 24 contexts is 4%, which we also surpassed. The error rate seen in the context classifier is likely a result of how the training data was generated in a noisy manner, for instance some overlap in density between a high density scenario and a medium density scenario exists. For our purposes a medium density is 4 or more nearby agents while 7 or more is high density. A large burden is also placed on the decision trees to distinguish the `Chaos` context from other contexts but this by its nature adds a lot of noise and has no structure, making it difficult to define hyperplanes to separate such scenarios.

5.3.2 Frames per Second

Our initial instantiation of a context-sensitive pipeline is much faster at runtime than the oracle. As seen in Table 5.1, all contexts experienced speedup, especially

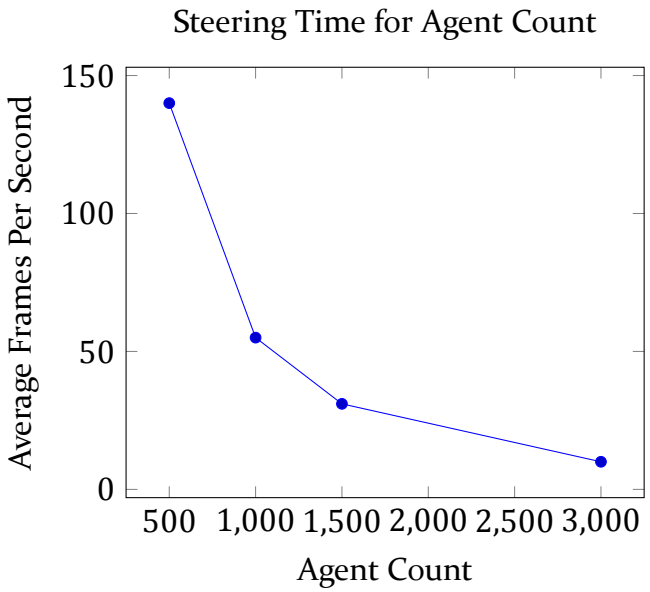


Figure 5.5: Total time taken for computing the steps 1,200 frames of simulation with varying numbers of randomly generated agents. Overhead was mostly incurred from a naïve implementation of agent density measurement which is $O\left(n^2\right)$ where n is the number of agents.

significant for the most challenging scenarios involving obstacles. The `Chaos` context, both with and without obstacles, was the most challenging for the oracle and resulted in skewed performance data due to the number of scenarios which were culled. Our method showed an extremely constant amount of time across the different contexts owing to its dynamic model-swapping.

To test the robustness of our collection of models, we created a large-scale simulation consisting of randomly generated obstacles, agents, and goals, as seen in Figure 5.6. We measured the time to generate the paths for varying numbers of agents to simulate 1,200 frames, with the results given in Figure 5.5. All tests were run using a single-threaded implementation and realtime framerates were experienced at 1,500 agents and interactive framerates of about 10FPS were experienced with as many as 3,000 agents. A major bottleneck in the results come from the generation of feature vectors, which is $O\left(n^2\right)$ in the number of agents. Each agent must check whether or

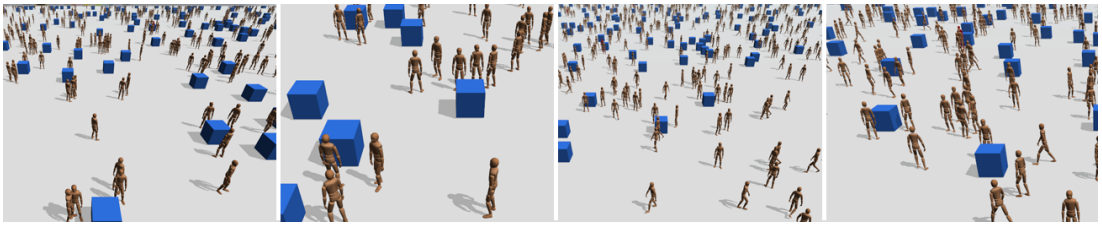


Figure 5.6: Multiple views of a 3,000 agent simulation rendered with the Unity game engine.

not another agent is close enough to be counted in the feature vector. This redundant calculation could be lowered by taking the reciprocity of agent distance into account but this would hurt the prospects of future parallelization.

5.3.3 Qualitative Analysis

Some screenshots of the framework at runtime are given in Figure 5.6, which used far more agents than the oracle could handle. We also ran several medium-scale scenarios that are far beyond the type of scenarios used for training the models to analyze the qualitative performance of the steering while using our data-driven technique. These scenarios were as follows:

Hallway Two opposing groups of 100 agents cross a hallway.

Random 500 randomly placed agents with 696 randomly placed obstacles throughout the environment.

Urban 2500 randomly placed agents in an environment simulating an urban area with obstacles as city blocks.

Recall our virtual agents navigate without an explicit collision avoidance stage to their navigation. Generally, the agents do not collide on the basis that their training samples contain no collisions, and thus they inherently steer around one another. However, as the models are not 100% accurate, collisions are to be expected.

Context	0	1	2	3	4	5	6	7	8	9	10	11
Oracle	0.73	13.84	5.11	15.53	12.35	9.26	1.68	67.27	101.56	19.90	14.71	1.20
Models	0.07	0.07	0.06	0.06	0.06	0.06	0.07	0.06	0.06	0.06	0.06	0.06

(a) Without Obstacles

Context	12	13	14	15	16	17	18	19	20	21	22	23
Oracle	123.95	785.0	1945.24	365.25	565.43	574.52	916.30	462.53	3384.10	577.54	396.79	64.78
Models	0.15	0.15	0.17	0.18	0.17	0.18	0.13	0.13	0.15	0.16	0.17	0.16

(b) With Obstacles

Table 5.1: Total time in seconds of step planning for all contexts to calculate steps over short scenarios. The first 12 are contexts without obstacles and are based on oncoming and cross traffic patterns with varying levels of agent density. Contexts 12 and above have obstacles and agent patterns matching the upper 12.

These tests were run for varying numbers of training scenarios, from 100 to 2000 in increments of 100 and each test was run for 3600 frames. Afterwards, we tabulated the number of collisions and created the graphs in Figure 5.7. The collisions were recorded by severity. Type A collisions have occlusions in the range $(0\%, 10\%]$ at the worst point. These collisions could be registered due to the circular profile of the agents' bounding volume and thus may not be visible when the simulation is rendered. Type B collisions have occlusion in the range $(10\%, 35\%]$ and while more severe than before, could be alleviated with a better anthropomorphic model with torso-rotation. This type of collision is often dealt with in real pedestrians by turning the shoulders to more easily pass one another in cramped conditions. Type C collisions occlude on the range $(35\%, 75\%]$ and are major collisions which require more tuning to the algorithm to avoid. Type D collisions complete the possibilities at $(75\%, 100\%]$ and would most likely need a fully reactive collision avoidance system to prevent.

The results were counterintuitive at first. As training samples grew in quantity, so did collisions and even the severity of the collisions. We hypothesize two main factors behind this increase. First, the oracle algorithm is collision-free. Thus a sort of "event horizon" was established in the training data where no reaction to an agent occurs once the agent is too close to another. This means once two agents are too close, there is no policy that would push them apart, which explains the increased amount of more serious collisions compared to the more minor offenses.

The second factor is that with increased sample counts, the models better attempt the mimicry of the oracle algorithm's behavior. The oracle has the ability to steer agents together in a very tight, close-call manner. While this is good for the oracle and such nearby passing can be accommodated by it, as the training data increases in size and the agents steer more like the oracle, a misstep is more likely to cause

a collision. In essence, more training data made the agents attempt to steer in a more precise manner, but the inherent inaccuracy of any machine learning algorithm simultaneously leads to higher risk. Thus a collision avoidance algorithm is necessary for a data-driven approach to agent steering, although perhaps more sparingly as model accuracy improves.

5.4 Summary of Performance

Our initial hypothesis of improved runtime was confirmed by this experiment. The oracle algorithm was universally slower than the hierarchical model, and in some contexts the difference in speed was orders of magnitude apart, as was shown in Table 5.1. Our data-driven algorithm was also demonstrated to generalize as accuracy improved for all classifiers as training data was increased, and the framework could produce simulations beyond the scope of the training data itself. No training data fed to the decision trees possessed even 20 agents, however agent counts in excess of 1,000 were possible. Given the performance of the oracle on the training scenarios, such high-count simulations were not feasible for comparison.

While the premise was shown to be sound, we also demonstrated the potential for improvement in our use of machine learning. Decision trees are a class of algorithms with high variance in the resulting models. Our accuracy compared to that of random guessing was high, but still lacking. The high variance inherent in our technique helped generalize the complex data we presented for fitting however the diminishing returns evident in Figure 5.4 suggest the samples are too complex—situations likely overlap too much—for the creation of more accurate models. Reasons for this complexity could be inherent in the phenomenon itself or rooted in our initial definition of the contexts. In the following chapter, we use data mining to explore other

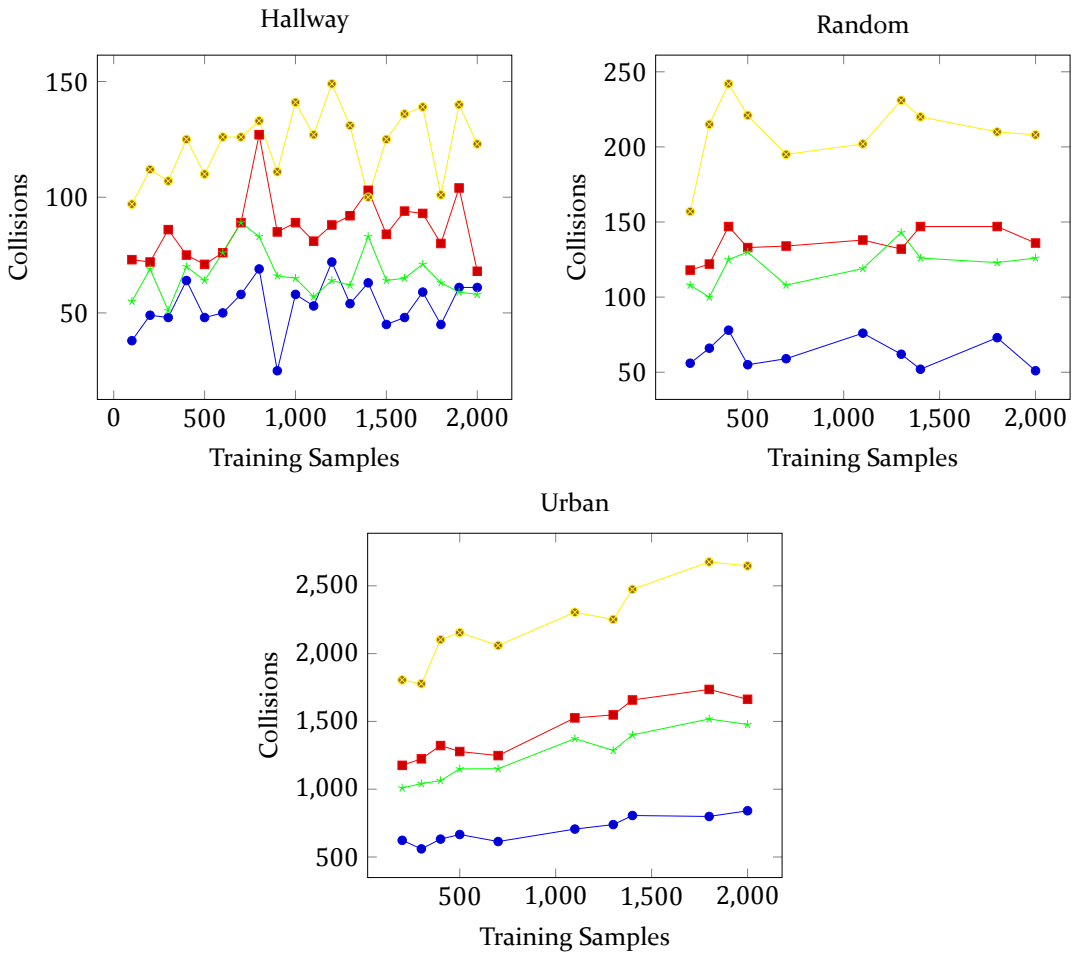


Figure 5.7: Counts for collisions in 3 minute simulations in different test scenarios. Type A collisions are blue, Type B collisions are red, Type C are yellow, and Type D are green. Once collisions occurred, there was little pressure for agents to move apart as the training data was collision-free, thus no samples existed for overlapping agents. Note that while high, per capita an agent in each of these simulations is only likely to encounter around 1-3 collisions with approximately one third of them minor in nature in spite of the lack of any explicit collision avoidance.

possible algorithms to improve this pipeline.

Chapter 6

Refinements Through Data Mining

> "Data! Data! Data!" he cried impatiently. "I can't make bricks without clay."
>
> *Arthur Canon Doyle*, The Adventure of the Copper Breeches

In this chapter, we present data mining as a means to evaluate and integrate possible options for improvement of the machine-learning component of our framework. In Section 6.1 we explain the use of unsupervised learning to better approximate contexts within scenario space. We detail the use of supervised learning for automatically generating a policy for each resulting context in Section 6.2 and conclude the chapter with data mining experiments and results in Section 6.3.

6.1 Context Identification

In Chapter 3 we used intuitively defined contexts as an example of breaking scenario space into qualitatively different types of scenarios. However we also showed

the number of contexts is not truly known and we must instead work with approximations of these steering contexts. Approximations may be refined a couple of different ways: e.g., by improving the precision of the data or by taking new perspective on how it is organized and used. Furthermore, different classes of data separated wholly on the basis of human intuition may not map well to a computationally feasible partition. Such intuitive separations may subsequently give rise to inaccuracies or biases in trained models.

To avoid human bias through intuitively-based separations, unsupervised machine learning techniques can be exploited to search for qualitative differences in data while also focusing on splitting the data in a manner more natural to the computer itself. The result is a better approximation of the potential contexts in our scenarios and a better method for identifying those contexts at runtime.

6.1.1 Unsupervised Learning Algorithms

Unsupervised machine learning algorithms, also called clustering algorithms, use a set of samples as input and group them together based on similarities of their various features. Unlike their counterpart, the supervised learning algorithms, these clustering techniques can operate on unlabeled data with the goal of labeling said data. In essence, rather than attempting to fit rules to discern between classes of data, unsupervised learning tries to derive what the likely classes of data could be.

This *modus operandi* is a natural fit to our goal of content approximation. We have a large set of steering scenarios and we know some are more similar than others, but we do not know with certainty how many types of scenario there are nor do we know the characteristics which define these types. Thus clustering algorithms' goal of identifying related scenarios and in the process deriving values which suggest some scenarios are more related than others will give us an automated manner

of computing an approximation for our contexts. There are several clustering algorithms which we consider as candidates for our work and we selected three due to their different approaches to determining the number of clusters. K-means [38] is given the number of clusters, X-means [51] takes a top-down approach, and canopy clustering [42] uses a bottom-up technique.

K-means is one of the best known and most used clustering algorithms. This algorithm uses an initial guess at the feature values which identify the different groups of data. These values form the centroids of each cluster. Samples are then assigned to a cluster based on nearest-neighbor proximity. The centroids are moved to the median of the clusters and the process is repeated until the centroids' positions reach convergence. The theory behind this algorithm is that the centroids will be pulled into position by data which is close together. K-means suffers from the problem of its random initial guess, local minima, and the user's estimation of the number of clusters. The random initial guesses can be improved with better seeding of the initial centroids [5] and we can use cross-validation to analyze several cluster counts and their relative merit.

X-means is a clustering algorithm which attempts to extend the concepts from K-means while avoiding its shortcomings. Rather than supplying the number of centroids as one of its parameters, X-means accepts a range of possible cluster counts. The iterative process discussed with K-means is extended to include refining the number of clusters by considering the results of splitting each current centroid into two children. These new clusters are initially separated an even distance from the original center along a randomly chosen vector. Ideally this will cause the child clusters to identify distinct groups of data rather than unnecessarily divide samples. X-means is not without its potential weaknesses. The success of the centroid division depends on the relative density of samples that are related versus those which are

not. Local minima can still exist with the randomized aspect of the new cluster generation. We can again use cross-validation to compare results with those of other algorithms.

In contrast with the prior two algorithms, canopy clustering was developed primarily as a way to accelerate the learning process for large data sets, cluster counts, and number of dimensions. Canopies serve as a rough pass on the training data using a similarity approximation and samples may belong to multiple canopies. Common membership in a canopy does not guarantee samples will belong in the same cluster, but lack of a common canopy indicates the samples cannot be in the same cluster. A second pass through the data computes an exact similarity between samples and assigns them to their actual clusters with a greatly reduced overall cost. By using heuristics in the initial creation of canopies, the number of clusters can be estimated and cross-validation used to compare possible models for the best result.

6.1.2 Principal Component Analysis

The 13-dimensional feature space used for context selection is not unreasonably high in and of itself but as the number of dimensions increases, samples move geometrically further apart and clustering becomes more difficult. This is because datapoints which are similar are harder to distinguish from those which are dissimilar based on Euclidean distance in a high-dimensional space. Thus it is to our advantage to consider reducing the number of features. Principal Component Analysis (PCA)[28] is a technique used to project the data onto a lower-dimensional space while also preserving much of the important variation in the dataset. The features after applying PCA are linear combinations of the original features so all original values of a datapoint can influence the values of the projected data. The success of this reduction is dependent on the original data which may or may not be well-suited to

projection.

6.1.3 Evaluating Cluster Results

Unlike with supervised learning algorithms, we cannot compute an accuracy metric when clustering unlabeled data. Rather than comparing to a "ground truth" or set of known classes, we must evaluate the clusters produced by unsupervised learning algorithms using metrics of relative likely quality. One such metric is the likelihood of the clusters with respect to the data provided, for which a formula is given in Equation 6.1 where $\mathcal{D}$ is the complete dataset, $\boldsymbol{\theta}$ represents the model—in this case cluster centroids—fit to $\mathcal{D}$, and $p\left(\mathbf{x}_i|\boldsymbol{\theta}\right)$ is the likelihood of the model with respect to the particular sample $\mathbf{x}_i$. An option for comparing the clusters is to compare the measures of likelihood resulting from each tested algorithm.

$$L = p\left(\mathcal{D}|\boldsymbol{\theta}\right) = \prod_{i=1}^{n} p\left(\mathbf{x}_i|\boldsymbol{\theta}\right) \tag{6.1}$$

Without knowing the number of clusters *a priori*, we must try different quantities. Cross-validation can be used to estimate the number of clusters, but a drawback to using likelihood as our metric is that the likelihood value tends to increase as the number of clusters increases. This is a phenomenon related to overfitting when training a model. Each cluster can be considered an additional, independent parameter for the algorithm to tune to the data. Additional parameters result in additional degrees of freedom and yield higher likelihoods. This is analogous to supervised machine learning's memorization problem. With memorization a model is fit to the training data with extreme accuracy, but later suffers a high error rate when given a separate test set as the model fit parameters too tightly to the specific data rather than identifying a trend. Since we do not have a separate test set, we need a more

robust metric than just likelihood.

The Akaike Information Criterion (AIC)[2] uses the likelihood as well as the number of parameters used in a model to produce a better relative metric. Equation 6.2 defines the AIC using the natural logarithm of the likelihood as well as $k_{\boldsymbol{\theta}}$, the number of parameters used by model $\boldsymbol{\theta}$. For our purposes, $k_{\boldsymbol{\theta}}$ is the number of clusters used by the algorithm. Akaike defines this criterion based on information theory such that the AIC is a unitless estimation of how much information is potentially lost by using a given model. In any form of machine learning, models are fit to data produced by some process—known or unknown—and by necessity some information is lost. This is manifest as the error rate of a learned model against a test set. In the absence of a known process, AIC provides a statistical foothold on quality while still being wary of overfitting. Theoretically, the best model out of a candidate pool is that which minimizes the probability of losing information relative to the other models.

$$\mathrm{AIC} = 2k_{\boldsymbol{\theta}} - 2\ln(L) \tag{6.2}$$

Since we are dealing with probabilities and the relative quality of various potential contexts, it is useful that AIC provides a probability-based comparison of two values. Given some model $\boldsymbol{\theta}_{\min}$ with $\mathrm{AIC}_{\min}$, another model $\boldsymbol{\theta}_i$ has the relative probability of actually minimizing information loss according to Equation 6.3:

$$\exp\left(\frac{\mathrm{AIC}_{\min} - \mathrm{AIC}_i}{2}\right) \tag{6.3}$$

We use the AIC metric to evaluate the candidate clustering algorithms due to its more balanced approach of comparing models with varying parameters.

6.2 Generating Policies for Contexts

In addition to seeking a better set of contexts, we also need to explore other options for creating a policy for a particular context. We consider several algorithms as potential replacements of the C5.0 algorithm used previously and take full advantage of our model hierarchy by looking for an algorithm better-suited to each context, rather than using the same decision tree technique for all policies.

6.2.1 Application of a Mixture of Experts

Our prototype in Chapter 5 consisted of a two-level hierarchy of decision trees. The top level of the hierarchy chose the most appropriate model for the agent's particular context. A major benefit of this departure from a single, monolithic model is the ability to use different models for different policies. Our approach also allows for the use of different algorithms for the different models. Rather than simply choosing different parameter values, the nature of the models themselves is open for customization. This concept of allowing smaller models to serve as specialists for narrowed problem domains is known as a mixture of experts[27], and is defined in Equation 6.4.

$$p\left(t|\mathbf{x}\right) = \sum_{k=1}^{K} \boldsymbol{\pi}_k\left(\mathbf{x}\right) p_k\left(t|\mathbf{x}\right) \tag{6.4}$$

In our use of this mixture of experts concept, $p_k\left(t|\mathbf{x}\right)$ gives the result of a specific policy, and we leverage our definition of contexts from Equation 3.1 to define $\boldsymbol{\pi}$. By using multiple experts, the result from each expert for the same sample can be combined into a single final result. Due to the binary nature of our $\boldsymbol{\pi}$ and our context classifications' result of only a single context for a particular $\mathbf{x}$, we do not need to

consult every policy.

$$\boldsymbol{\pi}_k(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in C_k \\ 0 & \text{if } \mathbf{x} \notin C_k \end{cases} \tag{6.5}$$

Thus we can consider our hierarchical machine learning to be a mixture of experts, and each policy serves as an expert. Since the model and underlying algorithm are customizable for each policy, other aspects of the data-driven process can be tailored as well. In this dissertation we concentrate on the choice of algorithms, but another avenue for exploration would be creating and selecting different feature spaces for the different contexts.

6.2.2 Feature Space

The models in our prototype application used discretized feature spaces. This was due to a limitation of the C5.0 algorithm used to generate all models. C5.0 can accept both continuous and discrete values for features, but handles each very differently. Nodes created for the continuous features may only be divided into three ranges according to two split values. Discrete values, conversely, allow the algorithm to consider various subsets as potential splits. This increases the branching factor and flexibility during the training process. In the short-term the use of discrete values improved the error rate but ultimately caused a loss of information as otherwise distinct values were forced into bins. These bins did not necessarily reflect the distribution of the data which could increase the loss of information as bins could be devoted to seldom-used values.

Since we were no longer constrained to the C5.0 algorithm, we changed the feature space to use continuous values. We used a filter on the data for tests on the

effectiveness of machine learning techniques that required discrete values. The filter generated bins based on a statistical view of the full data set to reduce the impact of lowering the precision. Since the filter created bins for each data set separately, each policy could have a different set of bins. While not as effective as a fully customized feature space for each context, this did allow some additional flexibility available only possible through the use of a hierarchy of models.

6.2.3 Action Space

We initially viewed the lack of collisions in the training data as a strength of the oracle's output. However, inaccuracies inherent in the use of machine learning led to collisions under the data-driven algorithm. The collisions exposed a gap in coverage for the policies, as no conditions comparable to the situation existed in the training data. In essence, the oracle's lack of collisions meant the data-driven agents could find themselves in a fundamentally novel situation which the agents could not "know" is improper and thus should be avoided. Fixing this problem required either the oracle be modified to permit collisions, or the data-driven models be made such that the wrong step is never selected. Only one of these options is possible, so the oracle was revised such that an agent may opt to collide with another but only as a last resort, enforced with a high penalty to the energy cost.

6.2.4 Classification Algorithms Used

We used the clustering results from Section 6.1 to form new contexts. The new contexts required new policies, so we repartitioned the footstep sample data according to the new contexts. We considered other algorithms than C5.0 for generating the policies and ran experiments to determine which algorithm should be responsi-

ble for generating which policy. In particular we considered some other decision tree algorithms, ruleset algorithms, Bayesian algorithms, and margin-based classifiers.

Various algorithms exist for creating decision trees. In addition to the one already used in our preliminary experiments, there are also the ID3 [52] and C4.5 [53] algorithms. The algorithms are similar and developed by the same person, J. R. Quinlan. The feature used for splitting the data at a given level of the decision tree is selected based on entropy and information gain. The algorithms differ in how the split is determined and what forms of data can be used for features. The strengths of these algorithms are their ability to fit to data with high variance as well as their need to only store the information for nodes rather than samples themselves.

Similar to decision trees, rulesets can also be created from training data. The internal structure of a decision tree can be converted into a ruleset, with each node's split information becoming a rule. In a sense, this type of algorithm subsumes decision trees. Rules can take on a more general form though and allow for more combinations and special cases than might be considered with techniques such as ID3. PART [17] is an example of learning rules based on techniques similar to a decision tree, in this case C4.5. An alternative is ripple-down learning [19] which takes a different approach to deriving rules. This method treats the most prominent class of data as the default classification. Rules serving as special cases or exceptions to this default rule are then created to accommodate the other possible classes found in the training data.

In contrast to the above, Bayesian statistics have been used to create classifiers. Naïve Bayes selects a class of data based on that which is most probable based on training samples. Bayesian networks [48] expand on the probability-based treatment of classification by adding the potential for dependencies to the mix. The strengths of these approaches are in their success with higher-dimensional data. Naïve Bayes in

particular is often used for text classification applications which can have dimensions in the tens of thousands.

Support vector machines (SVMs) [12] focus the most on geometric relations between classes and features. An SVM fit to training data can provide the smallest memory footprint at runtime. This is due to the need to only keep specific samples which serve as the boundaries between classes. SVMs can also use the kernel trick to find separating planes in transformed spaces, which can help divide classes in otherwise difficult datasets.

6.2.5 Evaluation Metrics

Our application consists of classes with very different representation in the samples, which complicates how we should measure the quality of our learning results. A simple measure of accuracy can be found by tallying the number of correct classifications. Accuracy then only accounts for the number of errors, not their nature. More thorough analysis is possible by further analyzing the types of incorrect classifications. In binary classification, we can consider two types of errors. False positives, or Type I errors, occur when a classified instance actually belongs to the other class. False negatives, or Type II errors, occur when a classified instance is a member of the class but not recognized as such by the model. This distinction of errors means a model may be incorrect by too heavily favoring a class, causing Type I errors, or by insufficiently acknowledging a class reflected by a high Type II error rate.

Recall and precision are two statistical measures which can be used to give more insight into the performance of the model by quantifying the manner in which the model is accurate. For instance, recall measures the ability to identify positives relative to the number of true positives. Higher recall then indicates higher probability of identifying the class given the sample does in fact belong to the class. Precision,

on the other hand, measures how likely the model is to be correct when it indicates a positive result. Precision increases when fewer false positives are returned. Neither recall nor precision is a complete picture on its own. High recall can be attained with low precision and vice-versa, though in both cases overall accuracy should be reduced. Rather than compare two measurements indirectly, recall and precision are combined into what is known as the F-measure which is the harmonic mean of the two quantities and defined in Equation 6.6.

$$F = 2\left(\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\right) \tag{6.6}$$

The F-measure summarizes accuracy of supervised learning algorithms well but focuses entirely on one part of the classification results, namely that of positively identifying a class. While the F-measure is potentially better than simple accuracy, both recall and precision can be artificially high when there is a lack of balance between positive and negative instances in the data. In particular, a class which dominates the dataset can be favored and result in a high F-measure. If the difference in sample count is high enough, a model could simply always return the more common classification and receive high numbers. The Matthews Correlation Coefficient (MCC)[41] computes a value which takes this imbalance into account. The MCC is formulated in Equation 6.7 and essentially considers the model's effectiveness for not only the original problem but its inverse as well. Thus if a model's high precision and recall is only an artifact of having comparatively few negative samples, the MCC will be reduced when the model fails to adequately identify those negative samples.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)\,(TP + FN)\,(TN + FP)\,(TN + FN)}} \tag{6.7}$$

Our policies are generated for a multiclass application, rather than binary classi-

fication as described above. However, each specific class can be analyzed using these metrics by considering success with the problem "class i" and "not class i." The results for all classes can be averaged to provide an overall metric for the success of the model. We use a weighted average when combining the individual class results to compensate for unbalanced class sizes.

6.3 Experiments and Results

We used the same training data from the oracle algorithm to explore the possibilities for both unsupervised and supervised learning algorithms. These experiments were all conducted using the `Weka` data mining system [22]. Clustering experiments were conducted first and the results deemed the best were used to define new contexts. Since these new contexts need new policies, further experiments were conducted to evaluate which algorithm should be used for each policy.

We used ten trial runs of each algorithm tested with each dataset. Different trial runs had different randomized ordering of the data. We also used 10-fold, stratified cross-validation in each trial to check the expected generalization of the algorithms. Stratification slightly alters the typical cross-validation scheme by actively matching the distribution of classes in the validation fold and the other 9 folds. As some of our classes are very rare, this is necessary to ensure some classes are not left out of the validation phase.

6.3.1 Clustering Contexts

We explored several options for unsupervised learning algorithms so we could define steering contexts based on a computational sense of similarity rather than our intuitive sense of similar scenarios. Creating clusters also gave us the opportunity

to correct for some problems with invalid class labeling. Some of the error in the context classifier in Chapter 5 likely came from polluted data caused by an entire oracle simulation being attributed to the same context. However the oracle's agents could move to new positions which would not match the original criteria defining the context, such as a sample from a crowded scenario after the agents have successfully passed one another.

For these trials, we predivided the data based on the presence or lack of obstacles. This decision was due to the ease of making the distinction before consulting the model hierarchy, much like we already distinguish between left and right step selections for the specialized classifiers themselves. Our decision was further justified because the clear divide between samples with and without obstacles made the single feature completely dominant with preliminary clustering experiments universally showing two clusters based solely on the obstacles.

One additional control available to us is the distance metric used to determine how close different sampled points are to one another. The standard metric is the L_2 norm, or Euclidean distance between two points. For our application the relative importance of each feature is not known, so they may not have a geometric relationship. Thus we also analyzed algorithms with the L_1 norm, commonly referred to as the Manhattan distance. This distance formulation adds the absolute value of the differences in all dimensions, rather than the square root of the summed squared differences.

Original Feature Space

We performed 10 runs of each clustering algorithms. A run consisted of a randomized ordering of the context samples and execution of 10-fold cross-validation. The results of these clustering experiments are given in Table 6.1 with average values

Table 6.1: Results of unsupervised learning algorithms on the original data generated by the oracle. K-means and X-means were tested with both Euclidean (L_2) and Manhattan (L_1) distance metrics. A † indicates the lowest AIC and a ‡ indicates results within statistical significance of the lowest value.

Algorithm	No Obstacles			With Obstacles		
	Clusters	Log-Likelihood	AIC	Clusters	Log-Likelihood	AIC
L_2 K-means	2	-10.83(0.73)	25.67(1.46)	2	-10.82(0.85)	25.65(1.71)
	3	-9.23(0.69)	24.45(1.37)	3	-9.10(0.65)	24.20(1.31)‡
	4	-7.75(0.64)	23.51(1.28)‡	4	-7.74(0.60)	23.47(1.21)‡
	5	-6.48(0.56)	22.95(1.12)‡	5	-6.64(0.56)	23.27(1.12)†
	6	-8.70(1.17)	22.62(0.97)†	6	-9.01(0.84)	23.37(1.04)‡
	7	-5.31(0.48)	22.87(0.92)‡	7	-5.68(0.52)	23.84(0.97)‡
	8	-4.43(0.46)	23.43(0.90)	8	-4.92(0.48)	24.48(0.95)
	9	-3.72(0.45)	24.25(0.89)	9	-4.24(0.48)	25.40(0.83)
	10	-2.58(0.43)	25.16(0.86)	10	-3.21(0.40)	26.43(0.81)
L_1 K-means	2	-11.72(0.40)	27.43(0.79)	2	-12.02(0.30)	28.04(1.93)
	3	-10.15(0.96)	26.29(1.93)	3	-10.18(0.77)	26.36(1.55)
	4	-8.70(1.17)	25.40(2.33)	4	-9.01(0.84)	26.02(1.68)
	5	-7.57(0.94)	25.14(1.88)	5	-8.00(0.83)	26.01(1.66)
	6	-5.31(0.48)	25.06(1.85)	6	-5.68(0.52)	26.03(1.47)
	7	-4.43(0.46)	25.37(1.71)	7	-4.92(0.48)	26.47(1.35)
	8	-3.72(0.45)	25.78(1.61)	8	-4.24(0.48)	26.85(1.39)
	9	-3.13(0.45)	26.45(1.63)	9	-3.70(0.41)	27.26(1.30)
	10	-3.67(0.79)	27.34(1.58)	10	-4.25(0.69)	28.50(1.39)
L_2 X-means	9.98(0.14)	-2.91(0.54)	25.78(1.13)	9.99(0.10)	-3.46(0.44)	26.85(0.75)
L_1 X-means	9.97(0.17)	-2.33(0.37)	24.64(0.75)	10(0.00)	-2.90(0.36)	25.79(0.72)
Canopy	19.18(1.6)	-0.43(0.74)	39.21(3.02)	18.71(1.66)	-1.16(0.71)	39.73(2.86)

and standard deviations. Note that smaller AIC values depict higher potential quality of the resulting clusters due to a decreased probability of information loss. The measures were compared for statistical-significance using a paired T-test and 0.05 two-tailed confidence interval. The results with the lowest AIC values were selected as potential candidates for clustered contexts.

PCA Feature Space

Our previous tests of unsupervised learning included different distance metrics to try to help mitigate the impact of the number of dimensions. We also ran experiments using PCA on the features first, which made a slight reduction to the overall number of dimensions and provided for more abstract features. We repeated our experiments with this lower-dimensional projection of the data.

Based on the results in Table 6.2, using PCA to try to reduce the dimensionality of the data was not beneficial to the clustering process. The number of clusters was suggested to be only 2 with statistical significance, which intuitively seems low compared to previous results and the diversity in simulations created as samples for the oracle. Additionally, the AIC values for this small value of clusters are much higher than those seen in Table 6.1, suggesting information is likely lost by using PCA. While some information is expected to be lost with the projection to a lower-dimensional space, the goal of PCA is to preserve the important variance which does not appear to have happened in our experiments.

6.3.2 Policies for Clustered Contexts

The footstep feature vectors were regrouped based on their corresponding context feature vector's cluster membership. Thus step data was no longer per-scenario but directly tied to a context definition, which prevented further errors from mislabeled

Table 6.2: Results of unsupervised clustering algorithms on the original data after dimensionality reduction with PCA. A †indicates the lowest AIC.

Algorithm	No Obstacles			With Obstacles		
	Clusters	Log-Likelihood	AIC	Clusters	Log-Likelihood	AIC
L_2 K-means	2	-12.53(0.12)	29.05(0.24)†	2	-12.52(0.13)	29.04(0.25)†
	3	-12.26(0.06)	30.53(0.13)	3	-12.28(0.04)	30.56(0.09)
	4	-12.07(0.08)	32.14(0.16)	4	-12.08(0.08)	32.15(0.16)
	5	-11.89(0.09)	33.77(0.18)	5	-11.89(0.11)	33.78(0.22)
	6	-11.69(0.10)	35.37(0.19)	6	-11.72(0.11)	35.44(0.22)
	7	-11.49(0.10)	36.99(0.21)	7	-11.55(0.12)	37.11(0.23)
	8	-11.30(0.10)	38.60(0.21)	8	-11.37(0.12)	38.73(0.25)
	9	-11.14(0.11)	40.27(0.21)	9	-11.17(0.15)	40.34(0.31)
	10	-10.96(0.11)	41.93(0.23)	10	-11.01(0.14)	42.03(0.29)
L_1 K-means	2	-12.57(0.04)	29.14(0.09)	2	-12.53(0.06)	29.06(0.13)
	3	-12.33(0.05)	30.66(0.10)	3	-12.35(0.15)	30.70(0.29)
	4	-12.15(0.06)	32.29(0.12)	4	-12.17(0.11)	32.33(0.22)
	5	-11.93(0.09)	33.86(0.17)	5	-11.94(0.10)	33.87(0.19)
	6	-11.74(0.10)	35.48(0.21)	6	-11.68(0.11)	35.37(0.23)
	7	-11.50(0.12)	37.01(0.24)	7	-11.46(0.13)	36.91(0.25)
	8	-11.34(0.12)	38.69(0.24)	8	-11.25(0.15)	38.50(0.29)
	9	-11.15(0.13)	40.30(0.25)	9	-11.04(0.17)	40.09(0.35)
	10	-10.96(0.14)	41.93(0.27)	10	-10.81(0.18)	41.62(0.36)
L_2 X-means	7.57(0.84)	-11.43(0.15)	37.99(1.50)	7.80(0.86)	-11.36(0.22)	38.32(1.43)
L_1 X-means	7.62(1.05)	-11.41(0.21)	38.05(1.78)	7.66(0.89)	-11.36(0.24)	38.04(1.41)
Canopy	10.63(1.28)	-11.92(0.09)	54.09(2.49)	9.72(1.24)	12.05(0.09)	43.53(2.40)

contexts. We tested C4.5, ID3, Naïve Bayes (NB), Bayesian network (BN), Ripple-down ruleset learning (Ridor), SVM, and PART algorithms in the `Weka` software suite.

Raw accuracy for each algorithm with each subset of data is given in Table 6.3. As discussed previously, raw accuracy is not necessarily a good discriminator of quality in machine-learned models, and the weighted F-measure is given in Table 6.4 to give a better analysis of the recall and precision produced by each algorithm's models. Since we have unbalanced representation of step selections in our training data, we ultimately used the weighted MCC in Table 6.5 for most decisions regarding which models to use for which policies.

Table 6.3: Raw error rates for each algorithm using each dataset with standard deviation to two significant digits. †indicates the lowest error rate for a particular context, and ‡indicates values which are within statistical significance of the lowest value.

Dataset	C4.5	ID3	NB	Ridor	BN	SVM	PART
cluster 0 left	0.30(0.12)‡	0.34(0.16)‡	20.51(2.71)	0.31(0.15)‡	1.88(0.41)	16.37(0.77)	0.28(0.16)†
cluster 0 right	0.27(0.07)†	0.30(0.10)‡	39.50(0.71)	0.33(0.08)	4.11(0.36)	12.07(0.62)	0.33(0.09)‡
cluster 1 left	2.22(0.12)†	2.44(0.18)	19.79(1.50)	2.43(0.16)	3.09(0.21)	18.08(0.38)	2.47(0.18)
cluster 1 right	2.05(0.15)†	2.04(0.18)‡	31.72(1.84)	2.37(0.18)	3.01(0.19)	16.01(0.48)	2.34(0.19)
cluster 2 left	0.83(0.14)†	1.01(0.20)	13.04(0.66)	0.87(0.16)	1.53(0.27)	17.09(0.80)	0.95(0.18)‡
cluster 2 right	0.68(0.10)†	0.76(0.11)	25.95(0.70)	0.78(0.12)	2.54(0.31)	10.41(0.52)	0.79(0.14)
cluster 3 left	1.00(0.12)†	1.09(0.19)‡	36.06(1.52)	1.11(0.15)	7.64(0.62)	14.43(0.51)	1.11(0.15)
cluster 3 right	0.99(0.12)‡	0.92(0.17)†	33.40(1.54)	1.15(0.17)	3.01(0.37)	15.14(0.64)	1.10(0.17)
cluster 4 left	0.16(0.06)‡	0.15(0.07)‡	19.76(1.61)	0.16(0.06)‡	1.22(0.23)	11.46(0.57)	0.15(0.06)†
cluster 4 right	0.30(0.07)†	0.38(0.11)	55.10(2.46)	0.35(0.09)	2.74(0.31)	10.52(0.68)	0.36(0.10)
cluster 5 left	0.12(0.04)†	0.14(0.05)‡	19.51(3.25)	0.14(0.05)‡	1.72(0.36)	11.96(0.47)	0.15(0.05)‡
cluster 5 right	0.22(0.06)†	0.28(0.08)	31.79(1.70)	0.26(0.07)	3.94(0.46)	10.56(0.51)	0.23(0.08)‡
cluster 6 left	17.15(0.70)†	20.00(0.91)	42.50(1.80)	18.67(1.07)	18.18(0.77)	39.33(0.47)	21.40(1.08)
cluster 6 right	14.50(0.53)†	16.49(0.63)	39.44(3.49)	16.41(1.11)	16.93(0.53)	28.09(0.41)	17.46(0.62)
cluster 7 left	14.75(0.37)†	16.23(0.54)	48.00(0.82)	16.66(0.57)	16.92(0.42)	30.46(0.08)	16.70(0.41)
cluster 7 right	16.40(0.49)‡	16.12(0.60)†	53.91(14.18)	18.70(1.35)	19.80(0.46)	36.65(0.62)	18.47(0.56)
cluster 8 left	15.819(0.51)‡	15.58(0.49)†	60.69(3.89)	17.20(0.75)	17.57(0.55)	33.44(0.08)	16.78(0.49)
cluster 8 right	13.60(0.33)	13.25(0.45)†	42.31(1.82)	14.89(0.63)	16.39(0.42)	32.36(0.41)	15.05(0.44)
cluster 9 left	18.77(0.54)†	24.34(0.82)	34.50(0.98)	22.09(1.26)	20.23(0.54)	33.94(0.05)	25.13(0.70)
cluster 9 right	19.19(0.57)†	22.50(0.84)	38.94(1.86)	21.86(1.28)	19.86(0.60)	35.95(0.30)	24.53(0.93)
cluster 10 left	17.35(0.56)†	20.31(0.72)	34.54(0.92)	19.48(1.22)	18.39(0.67)	34.47(0.06)	22.21(0.71)
cluster 10 right	16.73(0.50)†	20.11(0.79)	40.36(4.16)	18.55(0.94)	18.42(0.58)	36.25(0.48)	21.27(0.64)

Table 6.4: Weighted F-measure for each algorithm using each dataset with standard deviation to two significant digits where the metric is weighted for each action by its frequency in the training data. †indicates the lowest error rate for a particular context, and due to the consistency of performance may be repeated for contexts where the standard deviation is too low for our significant digits.

Dataset	C4.5	ID3	NB	Ridor	BN	SVM	PART
cluster 0 left	1.00(0.00)†	1.00(0.00)†	0.84(0.02)	1.00(0.00)†	0.99(0.00)	0.83(0.01)	1.00(0.00)†
cluster 0 right	1.00(0.00)†	1.00(0.00)†	0.58(0.01)	1.00(0.00)†	0.98(0.00)	0.88(0.01)	1.00(0.00)†
cluster 1 left	0.97(0.00)	0.98(0.00)†	0.81(0.01)	0.97(0.00)	0.97(0.00)	0.79(0.00)	0.97(0.00)
cluster 1 right	0.98(0.00)†	0.98(0.00)†	0.72(0.01)	0.97(0.00)	0.97(0.00)	0.83(0.00)	0.98(0.00)†
cluster 2 left	0.99(0.00)†	0.99(0.00)†	0.88(0.01)	0.99(0.00)†	0.99(0.00)†	0.82(0.01)	0.99(0.00)†
cluster 2 right	0.99(0.00)†	0.99(0.00)†	0.74(0.01)	0.99(0.00)†	0.98(0.00)	0.89(0.01)	0.99(0.00)†
cluster 3 left	0.99(0.00)†	0.99(0.00)†	0.69(0.01)	0.99(0.00)†	0.95(0.00)	0.85(0.01)	0.99(0.00)†
cluster 3 right	0.99(0.00)†	0.99(0.00)†	0.65(0.01)	0.99(0.00)†	0.98(0.00)	0.85(0.01)	0.99(0.00)†
cluster 4 left	1.00(0.00)†	1.00(0.00)†	0.83(0.01)	1.00(0.00)†	0.99(0.00)	0.88(0.01)	1.00(0.00)†
cluster 4 right	1.00(0.00)†	1.00(0.00)†	0.48(0.02)	1.00(0.00)†	0.98(0.00)	0.89(0.01)	1.00(0.00)†
cluster 5 left	1.00(0.00)†	1.00(0.00)†	0.84(0.02)	1.00(0.00)†	0.99(0.00)	0.87(0.01)	1.00(0.00)†
cluster 5 right	1.00(0.00)†	1.00(0.00)†	0.70(0.02)	1.00(0.00)†	0.97(0.00)	0.89(0.01)	1.00(0.00)†
cluster 6 left	0.81(0.01)†	0.79(0.01)	0.57(0.01)	0.80(0.01)	0.80(0.01)	0.49(0.01)	0.78(0.01)
cluster 6 right	0.84(0.01)†	0.82(0.01)	0.62(0.02)	0.82(0.01)	0.81(0.01)	0.64(0.01)	0.82(0.01)
cluster 7 left	0.83(0.00)†	0.82(0.01)	0.55(0.01)	0.82(0.00)	0.81(0.00)	0.57(0.00)	0.82(0.00)
cluster 7 right	0.82(0.01)†	0.82(0.01)†	0.50(0.12)	0.80(0.01)	0.78(0.00)	0.57(0.01)	0.81(0.01)
cluster 8 left	0.82(0.01)	0.83(0.01)†	0.44(0.03)	0.81(0.01)	0.81(0.01)	0.53(0.00)	0.82(0.01)
cluster 8 right	0.84(0.00)	0.85(0.00)†	0.63(0.01)	0.83(0.01)	0.81(0.00)	0.63(0.00)	0.83(0.00)
cluster 9 left	0.78(0.01)†	0.74(0.01)	0.64(0.01)	0.76(0.01)	0.77(0.01)	0.53(0.00)	0.74(0.01)
cluster 9 right	0.78(0.01)†	0.76(0.01)	0.61(0.01)	0.76(0.01)	0.77(0.01)	0.52(0.01)	0.75(0.01)
cluster 10 left	0.81(0.01)†	0.78(0.01)	0.65(0.01)	0.79(0.01)	0.80(0.01)	0.52(0.00)	0.77(0.01)
cluster 10 right	0.81(0.01)†	0.79(0.01)	0.61(0.02)	0.80(0.01)	0.80(0.01)	0.55(0.01)	0.78(0.01)

Table 6.5: Weighted Matthews Correlation Coefficient for each algorithm using each dataset with standard deviation to two significant digits where the metric is weighted for each action by its frequency in the training data. †indicates the lowest error rate for a particular context, and due to the consistency of performance may be repeated for contexts where the standard deviation is too low for our significant digits. ‡indicates values which are within statistical significance of the lowest value.

Dataset	C4.5	ID3	NB	Ridor	BN	SVM	PART
cluster 0 left	0.99(0.00)†	0.99(0.00)†	0.61(0.03)	0.99(0.00)†	0.97(0.01)	0.54(0.02)	0.99(0.00)†
cluster 0 right	0.99(0.00)†	0.99(0.00)†	0.31(0.01)	0.99(0.00)†	0.96(0.00)	0.75(0.01)	0.99(0.00)†
cluster 1 left	0.94(0.00)†	0.94(0.00)†	0.59(0.02)	0.94(0.00)†	0.93(0.01)	0.42(0.01)	0.94(0.00)†
cluster 1 right	0.96(0.00)†	0.96(0.00)†	0.51(0.02)	0.95(0.00)	0.94(0.00)	0.63(0.01)	0.95(0.00)
cluster 2 left	0.98(0.00)†	0.98(0.00)†	0.75(0.01)	0.98(0.00)†	0.97(0.00)	0.58(0.02)	0.98(0.00)†
cluster 2 right	0.98(0.00)†	0.98(0.00)†	0.34(0.02)	0.98(0.00)†	0.95(0.01)	0.72(0.01)	0.98(0.00)†
cluster 3 left	0.97(0.00)†	0.97(0.01)†	0.41(0.02)	0.97(0.00)†	0.86(0.01)	0.55(0.02)	0.97(0.00)†
cluster 3 right	0.98(0.00)†	0.98(0.00)†	0.32(0.02)	0.98(0.00)†	0.96(0.00)	0.68(0.01)	0.98(0.00)†
cluster 4 left	1.00(0.00)†	1.00(0.00)†	0.58(0.03)	1.00(0.00)†	0.97(0.01)	0.62(0.02)	1.00(0.00)†
cluster 4 right	0.99(0.00)†	0.99(0.00)†	0.31(0.02)	0.99(0.00)†	0.96(0.00)	0.76(0.02)	0.99(0.00)†
cluster 5 left	1.00(0.00)†	1.00(0.00)†	0.62(0.03)	1.00(0.00)†	0.97(0.01)	0.65(0.01)	1.00(0.00)†
cluster 5 right	1.00(0.00)†	1.00(0.00)†	0.47(0.02)	1.00(0.00)†	0.95(0.01)	0.79(0.01)	1.00(0.00)†
cluster 6 left	0.69(0.01)†	0.65(0.02)	0.34(0.02)	0.67(0.02)	0.68(0.01)	0.15(0.02)	0.63(0.02)
cluster 6 right	0.71(0.01)†	0.67(0.01)	0.30(0.02)	0.68(0.02)	0.66(0.01)	0.32(0.01)	0.66(0.01)
cluster 7 left	0.66(0.01)†	0.64(0.01)	0.25(0.01)	0.64(0.01)	0.63(0.01)	0.07(0.01)	0.64(0.01)
cluster 7 right	0.74(0.01)†	0.73(0.01)	0.33(0.09)	0.71(0.01)	0.67(0.01)	0.35(0.01)	0.71(0.01)
cluster 8 left	0.68(0.01)‡	0.67(0.01)†	0.21(0.02)	0.66(0.01)	0.65(0.01)	0.06(0.01)	0.66(0.01)
cluster 8 right	0.77(0.01)†	0.77(0.01)†	0.47(0.01)	0.76(0.01)	0.73(0.01)	0.42(0.01)	0.76(0.01)
cluster 9 left	0.61(0.01)†	0.52(0.02)	0.33(0.02)	0.56(0.02)	0.59(0.01)	0.02(0.02)	0.52(0.01)
cluster 9 right	0.64(0.01)†	0.58(0.02)	0.33(0.02)	0.60(0.02)	0.62(0.01)	0.13(0.02)	0.57(0.02)
cluster 10 left	0.66(0.01)†	0.60(0.02)	0.38(0.01)	0.63(0.02)	0.65(0.01)	0.04(0.01)	0.58(0.01)
cluster 10 right	0.71(0.01)†	0.65(0.01)	0.41(0.03)	0.68(0.01)	0.68(0.01)	0.28(0.01)	0.65(0.01)

Chapter 7

Improved Application

> Strive for continuous improvement,
> instead of perfection.
>
> *Kim Collins*

In this chapter, we use the results of our data-mining experiments to make a second iteration of simulations. We then use the updated hierarchical model to explore the limitations of our pipeline and data-driven crowd simulation.

7.1 Oracle Improvement

Originally we planned to use the same training data from Chapter 5 without any change to the oracle itself. However, an upgrade in hardware required running the oracle again for accurate comparisons of performance. Since we needed to run the oracle's scenarios again, we took the opportunity to try to improve it and potentially acquire better training data. In particular we adjusted two behaviors in the oracle's implementation.

7.1.1 Collisions

The oracle's lack of collisions were previously viewed as a strength of the algorithm itself. The reasoning was that if an agent had no training that might involve colliding with another, the situation would be less likely to present itself. As seen in the preliminary results, however, other factors can lead to agents colliding with each other. What began as a strength was revealed as also having a weakness associated with it; once the agents experienced a collision, nothing in their training motivated them to separate. Rather than completely dismissing steps with collisions, we attach a high energy penalty to make it extremely unfavorable. Agents should only collide in rare instances when the only other option is complete deadlock.

7.1.2 Action Space

While implementing the high energy penalty, we discovered and fixed several weaknesses in the original planner code regarding the selection of a stop action. Agents were effectively prevented from considering a temporary stop as a viable part of a solution in all but the rarest of instances. To stop, an exact sequence of steps had to be accepted by the planner without deviation. Although we refined the approach, the nature of IDA* is such that stopping must bear an energy penalty disproportionate to its actual energy cost. Without this additional penalty, the planner must consider arbitrary stops of arbitrary length at any point along an agent's potential path. We empirically set this penalty to allow stopping to occur but not lead to unworkable runtimes. As a result, agents do not stop as often as might be encountered in real pedestrians, but the action is better represented in the data.

Table 7.1: These are the feature values for each centroid produced by clustering all samples of scenarios without obstacles.

Region	Feature	Cluster0	Cluster1	Cluster2	Cluster3	Cluster4	Cluster5
North	Speed	0.0083	0.6429	1.0145	0.6489	0.0276	0.9875
	Theta	-0.0016	-0.1719	-0.4009	-0.3656	0.0068	-0.3133
	Density	0.1368	1.1682	2.3913	1.3586	0.4579	2.4844
East	Speed	0.9805	0.7940	1.0166	1.0160	0.0174	0.0043
	Theta	-0.6722	-0.2732	-0.7727	-0.7187	0.0117	0.0010
	Density	2.5267	1.5932	2.2646	2.0931	0.3592	0.0799
South	Speed	0.2703	1.0216	0.3915	0.0010	0.1360	0.0558
	Theta	0.0017	-0.0220	-0.0139	0.0027	0.0109	0.0008
	Density	0.8205	2.1157	0.7241	0.0226	0.3916	0.1404
West	Speed	0.0078	1.0212	0.0009	1.0173	0.3184	0.3555
	Theta	0.0140	0.2868	0.0027	0.6064	0.2893	0.3074
	Density	0.1586	2.0230	0.0345	1.9874	0.8738	0.8962

7.2 Classification Improvements

In addition to the changes made to the oracle algorithm, we used the results from our experiments in the previous chapter to modify our use of classification and the model hierarchy.

7.2.1 Context Classifier

Based on results from Section 6.3.1, we used k-means clustering with 6 clusters for scenarios lacking obstacles and 5 clusters for scenarios with obstacles. Consequently we reduced the total number of contexts by just over half the original count. The centroids for contexts derived for the absence of obstacles are given in Table 7.1 and were fit with a log-likelihood of -7.27. This is slightly better than the projected log-likelihood in Table 6.1 and we believe the difference is from the extra data available when not using cross-validation. Table 7.2 gives the centroid information for the contexts with obstacles, which were fit with a log-likelihood of -7.91, which was lower than projected.

Table 7.2: These are the feature values for each centroid produced by clustering all samples of scenarios with obstacles.

Region	Feature	Cluster6	Cluster7	Cluster8	Cluster9	Cluster10
North	**Speed**	0.9991	0.9932	0.0161	1.0111	0.0044
	Theta	-0.4125	-0.5498	-0.0027	-0.3053	0.0060
	Density	2.2113	2.1903	0.2277	1.7770	0.0616
East	**Speed**	0.9220	0.3095	0.4648	0.7623	0.6866
	Theta	-0.6500	-0.1964	-0.3361	-0.2328	-0.1967
	Density	2.0640	0.6455	1.2180	1.4979	1.5315
South	**Speed**	0.4095	0.0030	0.0079	1.0068	1.0020
	Theta	0.0129	0.0038	0.0136	0.0050	0.0182
	Density	0.8007	0.0329	0.0945	1.8522	2.6555
West	**Speed**	0.0025	0.5981	0.3938	1.0069	0.6800
	Theta	0.0060	0.3592	0.3296	0.2925	0.2086
	Density	0.0533	1.3343	0.9371	1.9954	1.4753

For runtime identification of an agent's context we switched from using a supervised learning model, such as a decision tree, to using the centroids found by the clustering algorithm with a nearest-neighbor search. This change allows for lower memory overhead as only the centroids must be stored rather than a full model and also logically matches well to our use of the clusters; given a set of clusters, a single new sample would have been attributed to the nearest centroid. Any other method of runtime context identification would have only added an unnecessary level of indirection.

7.2.2 Specialized Classifiers

Based on the results observed in our machine learning experiments in Section 6.3.2, we selected a learning algorithm for each cluster's policy. Recall the data was split based on which foot is being used, for a total of 22 models. Four particular algorithms stood out as the best candidates for these policies: ripple-down rules, PART rules, C4.5 decision trees, and ID3 decision trees. Often these algorithms were

Table 7.3: Each cluster's designated policy was selected based on the combined performance metrics of accuracy, F-measure, and MCC.

Cluster	Left Foot	Right Foot
cluster0	Ripple-Down	C4.5
cluster1	C4.5	ID3
cluster2	Ripple-Down	Ripple-Down
cluster3	C4.5	C4.5
cluster4	Ripple-Down	Ripple-Down
cluster5	Ripple-Down	PART
cluster6	C4.5	C4.5
cluster7	C4.5	C4.5
cluster8	ID3	ID3
cluster9	C4.5	C4.5
cluster10	C4.5	C4.5

in tight contention for "best" algorithm and results for some of the metrics would be within statistical significance of each other. Final selections were made by using the results of all three performance metrics and determining which technique performed the best on more metrics than the others.

The algorithm selections for each policy are provided in Table 7.3. Ripple-down learning was the most commonly used algorithm for scenarios without obstacles, where the oracle's paths were predominantly straight as turning generally relied on course-corrections and more subtle movements of that nature. The introduction of obstacles into scenarios forced the oracle to use more frequent and pronounced turns to steer around objects. As a result, the high variance characteristic of decision trees appears better suited to handle this additional complexity.

7.2.3 Model Interface

Since the `Weka` suite was used to perform the data mining experiments, we used the `Weka` implementations of the algorithms for our runtime models. The toolkit provides for the export of models for subsequent runtime use which streamlined the

process of putting new models in use within our pipeline. The tradeoff, however, lies in the Java implementation of the `Weka` codebase. Because Java runs on a virtual machine, it is generally known to run less efficiently on hardware compared to a C++ implementation. While the gap in performance has been shrinking, this must be taken into account when viewing performance results.

We extended the interface to our models to use the Java Native Interface (JNI) distributed with Java. This technology allows for intercommunication between C++ and Java code. For our purposes, the proper technique requires the instantiation and loading of the necessary Java classes into a dynamically-linked library (DLL) form of the Java Virtual Machine (JVM). Thus, Java code must be run through the JVM intermediary, which adds overhead due to the use of interprocess communication and access model of the virtual machine itself. As Java class method calls and return values are the performance bottleneck, we implemented our JNI wrapper to minimize these calls and instead perform as much computation on either side of the interface as possible.

7.2.4 Emergency Stop Action

In our first prototype application, there was no collision detection or avoidance and we instead used the `C5.0` classification's provided confidence rating as a trigger to the agent if it should execute a stopping action. The confidence rating was discovered to be based on precomputation embedded in the leaf of the tree at the time of training. Since the rating was not calculated based on a dynamic comparison of the current feature vector and the decision being offered, we looked for a different and more effective approach for better stopping behaviors.

Rather than use a prediction based on the feature vector, we elected to focus on the results of the policy's chosen action. The agent is permitted to use the action cho-

sen by the policy with two exceptions. If the action violates the inverted pendulum model used by the agent for locomotion or if the action will lead to an immediate collision, the action is rejected. In place of the policy's action, the agent is forced into a stopping action. Once in this stopping action, the agent will become stationary until it determines a starting action is valid. The agent resumes motion and also resumes the use of policies for determining future actions.

7.3 Performance

The following results were generated on a desktop PC with 32GB of RAM, NVIDIA GeForce GTX TITAN graphics card, and Intel Core i7 3930K six-core processor supporting twelve hardware threads running at 3.2GHz. The same test scenarios were used to compare runtimes between the oracle and data-driven algorithms. Table 7.4 gives the runtimes and shows that the new models run slower than the the original models, but faster than the oracle itself. This was expected due to the additional overhead of using JNI.

7.3.1 Frames per Second

To check the overall scalability of our revised technique, we replicated our prior experiments of scenarios with randomly generated obstacles and agents. The results are shown in Figure 7.1 and demonstrates a linear growth with respect to the number of agents. The graph further shows our implementation can support generating steps for nearly 2,000 agents in realtime. An example view of the randomized scenario is provided in Figure 7.2.

The average decision time per step was also calculated and given in Figure 7.3. The average decision time is not constant but also grows linearly with the number

Table 7.4: Average time in seconds of step planning for 50 test scenarios from each original context, to three significant digits. Each test scenario ran for 25 simulated seconds.

(a) Without Obstacles

Context	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Oracle	0.0321	0.357	2.53	23.6	7.13	3.37	0.117	0.909	65.7	5.97	6.16	0.441
C5.0	0.0308	0.0757	0.133	0.229	0.178	0.155	0.0744	0.140	0.214	0.188	0.156	0.0884
Clusters	0.153	0.243	0.324	0.437	0.368	0.338	0.245	0.328	0.430	0.372	0.332	0.231

(b) With Obstacles

Context	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Oracle	5.28	111	179	1890	53.3	97.4	514	644	620	337	1750	20.9
C5.0	0.0974	0.197	0.371	0.599	0.487	0.408	0.153	0.308	0.450	0.413	0.356	0.196
Clusters	0.206	0.326	0.453	0.661	0.562	0.485	0.302	0.440	0.627	0.550	0.481	0.309

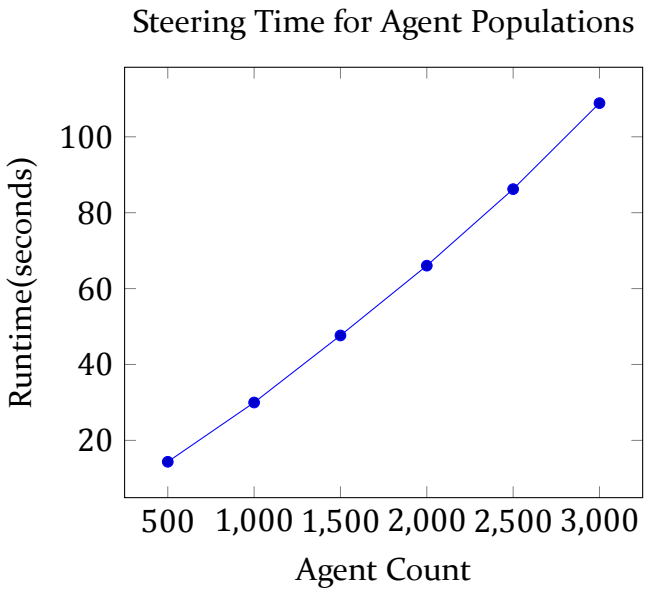


Figure 7.1: Total time taken for computing the steps of a simulation 1,200 frames long for varying numbers of agents with randomly generated obstacles and an overall small area.

of agents. This time could be reduced with a more optimized implementation, such as the use of natively-coded algorithms in lieu of JNI. These results strongly indicate that parallel threads could boost the supported agent count, likely with a nearly scalar improvement.

7.3.2 Policy Use

Since the policies are not 100% accurate, incorrect steps can still be selected by the policies at runtime. This motivated our inclusion of a the "emergency stop" action. For large scenarios, we cannot use the oracle algorithm to generate an expected or ground-truth series of steps due to the prohibitive amount of time required to run the simulations. However, we can still measure the performance of the hierarchical model by checking how often the stopping action is invoked. We checked this rate of use for the random scenario used in the previous section, an urban-based scenario

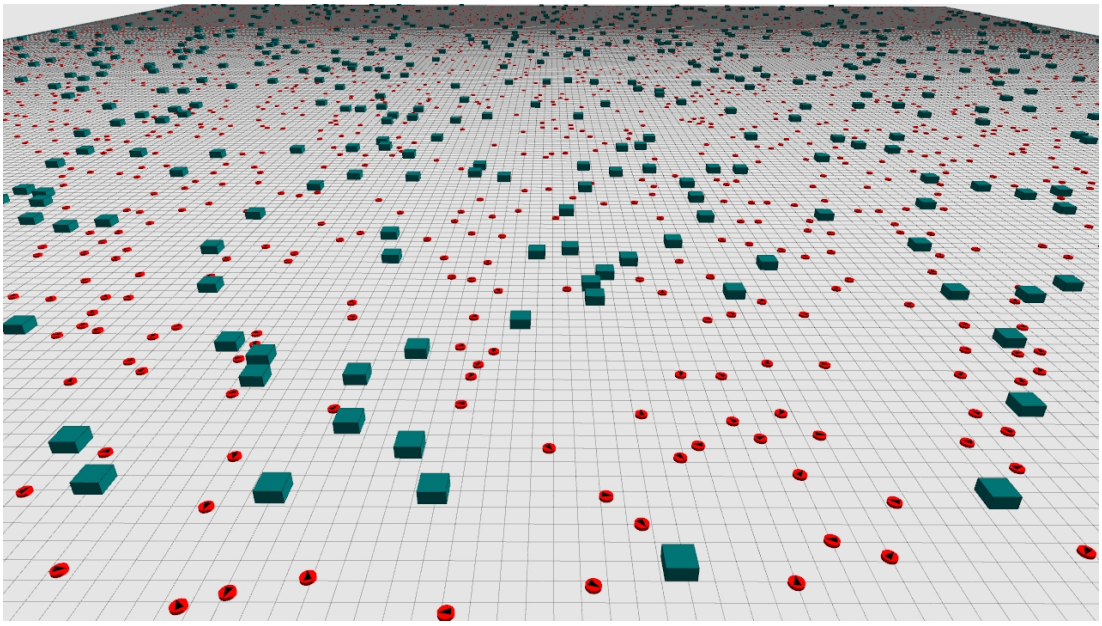


Figure 7.2: One view of the randomized scenario used for much of our performance testing.

provided in Figure 7.4, and a hallway scenario illustrated in Figure 7.5.

As shown in Figure 7.6, the percentage of policy-based steps used by agents stays in the upper 80% range even at larger populations as seen in the random and urban scenarios. The percentage is also high in situations with relatively crowded populations, such as the hallway scenario. Although the rejection of the policy-based step may help stop collisions and actions inconsistent with the inverted pendulum model, it is important to note that the step ultimately used by an agent to resume motion is not tested for ideal progress towards the agent's goal. Thus agent progress towards goals and actions to circumvent collisions and obstacles come from the policies.

7.3.3 Qualitative Analysis

In general, the data-driven agents follow paths with similar characteristics to those seen with oracle-driven agents. Since the planner used by the oracle uses a heuristic based on the expected energy cost per step, paths are predominantly

Figure 7.3: The average time taken per-agent to decide their next step action for various population sizes.

straight. Differences arise with turning radii when the agents must turn, as the oracle has the ability to plan ahead rather than just use a single step. Thus the oracle can better fine-tune the turning angle.

The use of a single turning angle for a given circumstance, rather than being able to plan ahead, results in degenerative behavior, an example of which is given in Figure 7.8. This behavior is particularly witnessed when agents must turn to reorient

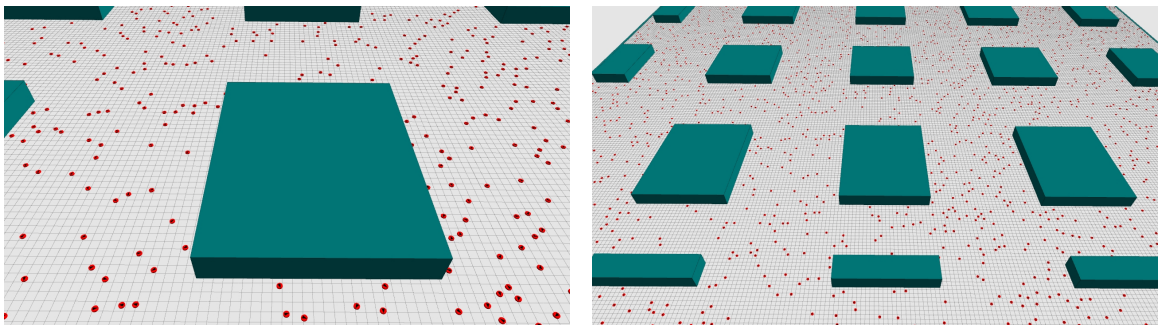


Figure 7.4: Two views of the urban-based scenario used for testing the frequency with which the data-driven agents resort to the emergency stop action.

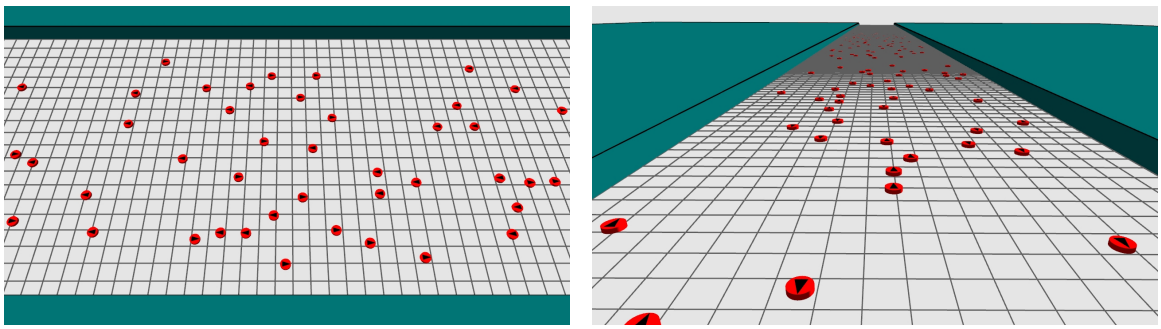


Figure 7.5: Two views of the hallway scenario used for testing the frequency with which the data-driven agents resort to the emergency stop action in a more confined space.

to a goal. Depending on the distance to the goal, turning radius, and availability of other stimuli, an agent may settle into an orbit around the goal. This happens for the same reason objects can maintain steady orbits around gravitational bodies in astronomy. The forces at play are rotationally invariant, so the same reaction occurs. Similarly, after a small distance traveled while turning, an agent observes the same state as before, which invokes the same step. This is a limitation of only being capable of planning a single step at a time and represents a limit reached with our technique's abilities.

The hallway scenario we tested also demonstrated lane-forming emergent behavior. As the two groups move to opposing sides of the hall, the agents within each group tend towards their right sides, letting the groups pass one another with fewer problems involving collisions. This is particularly interesting to witness as the training data for the hierarchical model does not specify such organized behavior, and also this is more consistent with human behavior than that often seen with crowd simulations. Crowd simulations often show lane-forming as many narrow lanes, not a bisection of the passage into two lanes as seen in Figure 7.7.

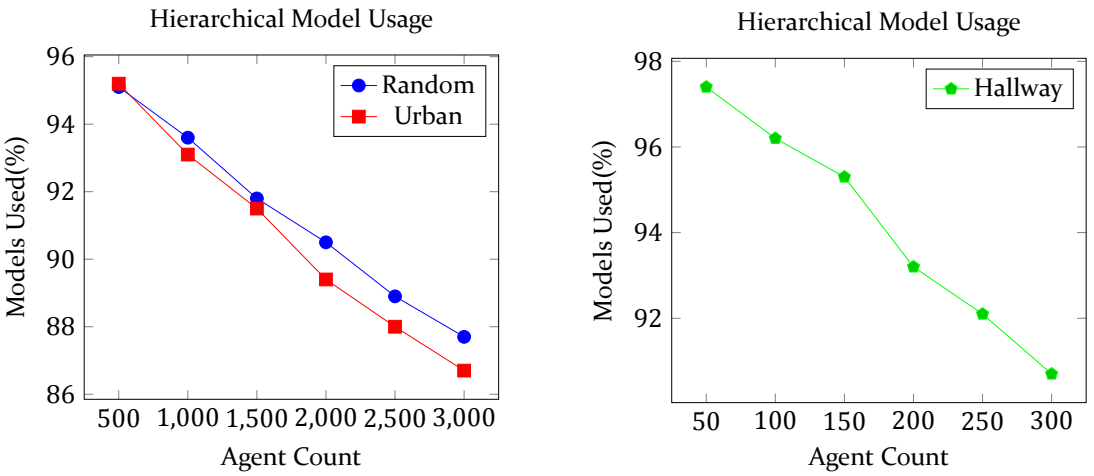


Figure 7.6: These two graphs indicate the hierarchical model usage for agents using our data-driven technique. The left graph is for higher-population simulations in more open areas while the right graph is for a different scenario which could not support the same agent counts.

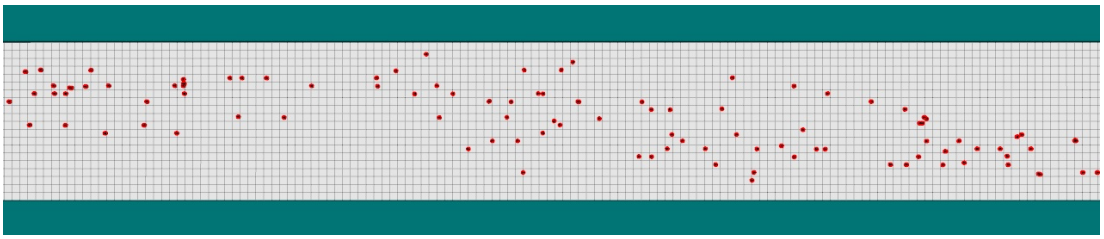


Figure 7.7: A view of an example hallway simulation where the AI produced lane-forming behavior. Rather than forming narrow lanes to squeeze past one another, the agents *en masse* move to their right sides, creating larger trends more consistent with common practice in the real world.

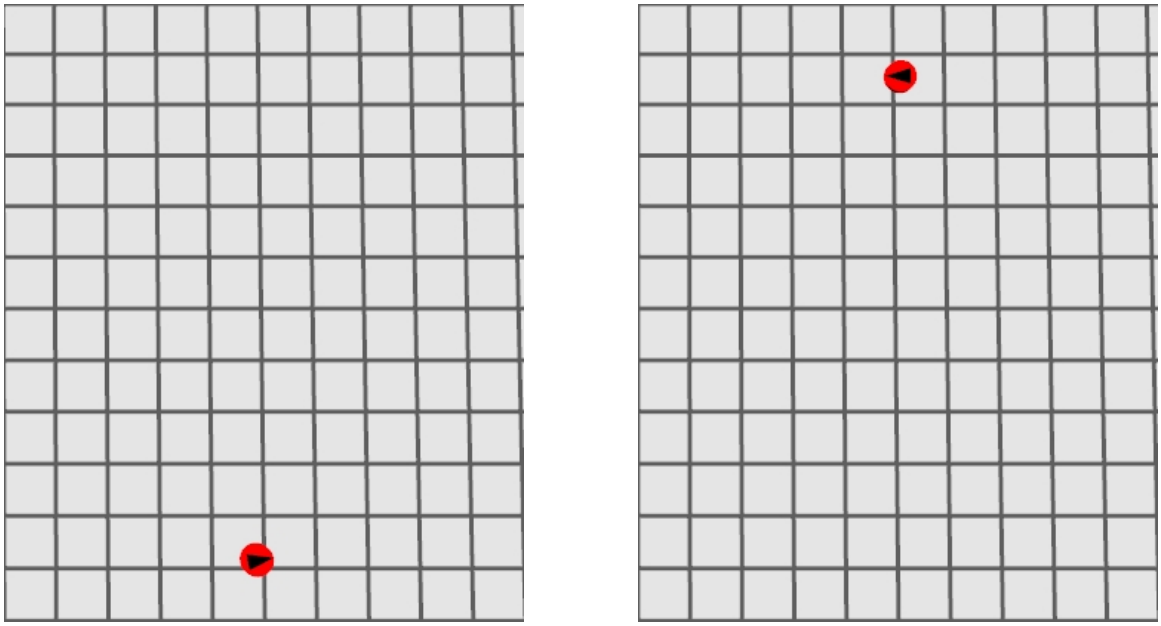


Figure 7.8: This is an example of an agent which will not reach its goal. The turning radius for the agent's selected step is such that the agent's local view of the scene is rotationally invariant, leading to the agent repeating the same decision.

Chapter 8

Conclusions and Future Work

> I'd now like to digress from my prepared remarks to say, "I'm done."
>
> *Bender*, Futurama

We have described a technique for systematically breaking down the problem of agent steering in crowd simulations, while also demonstrating the efficacy of computer-generated training data for pedestrian paths. In this chapter, we conclude with a discussion of the limitations of our system and potential avenues for future work.

8.1 Conclusions

This dissertation has defined steering contexts, a new view on the space of possible scenarios an agent may encounter as it steers through its virtual world. These contexts provide insight into the task of creating a robust, general-purpose steering controller suitable for any situation. Unless the controller can be independently proven to consist of a single context, the algorithm will shatter scenario space into subsets which must be handled by separate policies. This creates an uncertainty of

coverage that is by its nature $\mathcal{NP}$-Complete. To our knowledge no realtime algorithm is unaffected by this discovery because as discussed in Section 3.2.1 different behaviors triggered by different conditions, such as collision avoidance, create contexts and thus require multiple policies.

We have also proposed a pipeline for constructing a steering algorithm that is both context-sensitive and scalable with respect to the circumstances the algorithm can handle. Through the use of a multiplicity of policies fit to steering contexts, machine learned models can be combined for more structured and principled coverage of the space of possible scenarios than would otherwise be possible by a single-model approach. We used an oracle algorithm to generate high quality, on-demand training data which can be used for new contexts without the overhead, uncertainty, or uneven coverage. The overhead of organizing volunteers and subsequently tracking their movements would have been large for a single scenario, let alone the vast number of scenarios we used. We could not have perfectly controlled the parameters of the scenarios as was possible with the virtual participants who could not possess motives or the behavior-changing awareness which comes with being observed.

Our training data was then broken into contexts based on intuition and policies fit for each context using machine learning. To even further remove the unscientific role of human intuition in partitioning the data into contexts, we demonstrated the derivation of a set of contexts formed by applying a clustering algorithm to the training data set. The contexts found through unsupervised machine learning were very different from those we defined by intuition, as evidenced by the values of the centroids.

This data-driven technique has shown a massive increase in efficiency as realtime simulation was achieved with far higher population counts than the oracle algorithm could handle. The oracle's calculations would exceed the amount of time required for

realtime simulation for scenarios with as few as 10 agents. By comparison, the data-driven simulations of the same scenarios were nearly always faster by 1–3 orders of magnitude and could support thousands of agents in realtime. Furthermore, training on this data resulted in relatively small numbers of collisions, many of them minor. The clustered contexts also resulted in paths similar to those generated by the oracle algorithm and some emergent behavior such as lane-forming which were not targeted by the training data.

8.1.1 Strengths

The context-sensitive approach detailed in this dissertation has several strengths compared to the current literature. Our technique has low memory overhead compared to other data-driven work which relies on searching for either the best match in a database of samples or finding the k-nearest neighbors in an arbitrarily large collection of data. We achieve this lower overhead by storing fit models rather than the data itself; in the case of the context classifier we reduce hundreds of thousands of samples to 11 points, each containing a 12-dimensional vector. Other models we used vary in how much space is saved, but none of the machine learning algorithms we used requires maintaining the whole collection of training data, which also led to memory consumption growing slower than the raw data set itself. The resulting collection of policies is also shared by all agents in the simulation, making the memory overhead a constant independent of the size of the population.

Another strength is that the runtime performance of footstep selection is very consistent. This is because models are quick to execute with only slight variation in runtime depending on how "deep" into the learned structure the computation must progress to reach a decision. The depth of the model is constant once learned which gives an upper bound on time required to attain a decision, which cannot be said for

algorithms involving loops or recursion. Perception of the environment does experience a linear increase in time as population increases, but it is a slow growth and is a possible avenue for future optimization. Improving the efficiency of the code evaluating the policies or constructing the feature vectors would have significant impacts on performance. Either of these are strong candidates for out-of-core processing on the GPU since each agent could perform these tasks independently.

8.1.2 Limitations

The data-driven work in this dissertation is not without its limitations. Many of these limitations are ultimately rooted in the same key issue shared by all data-driven techniques. Agents' paths are generated piecemeal rather than in totality. Construction of a complete long-term solution from partial short-term results led to several problems.

One such problem is concerned directly with model inaccuracy. If we envision each selected step as being a component of an overall ideal plan, even one step in the wrong direction will change the entire path. The next observation of the environment will not take place from the same expected perspective, which can lead to a different selection in the next step, and so on. Thus even with extremely high accuracy, there exists a butterfly effect of consequences for an inaccurate selection. Inaccurate steps also cause secondary ripples through the simulation, as the selections will alter the agent's location with respect to what other agents would have expected for their own plans. While our models could exceed 90% accuracy which is far higher than random chance, this still represents an average of 1 false step for every 10. The full extent to which these suboptimal steps impact the simulations requires further analysis because though it is true the emergency stop action is evidence of their negative influence, we do not know how often these deviations from an ideal solution can be

corrected by subsequent data-driven actions.

Another problem with data-driven crowd simulation comes from the global versus local nature of step selection. The oracle algorithm used in this work planned a long series of steps for an agent to execute. By using data-driven techniques to decide a single step at a time, we are treating footstep selection as a Markovian process, meaning the past values for the agents' states are not a factor for the current decision. Some steps were used rarely by agents in our training data. While it is possible they were used for extremely rare circumstances in the virtual environment, it is also possible they were used because they helped form very specific paths as the oracle's cost function was minimized. If the steps are sensitive to the path being used, one could argue against the Markovian assumption, which leads to a fundamental shortcoming in data-driven techniques for agent steering. This is because proper modeling of the step selections would require accumulating this past state which requires either allowing the features to grow over time without bound, or discarding the past state which causes the fidelity of the state space to degrade with time.

8.1.3 Suggested Uses

As it currently exists, we can foresee several application areas for the development pipeline presented in this dissertation. First, the modular aspect of the pipeline and hierarchical nature of the policies makes the pipeline a robust framework for future data-driven crowd simulation efforts. This work allows for the specific focus on steering contexts as a way to generalize from considering special cases and instead to thinking of categories of cases with the intent to increase the realism and/or scenario coverage of data-driven crowd simulation.

Another potential application area is in anomaly detection [9]. The limitations expressed above can be lessened if an agent is locked to a runtime source of path

information. In monitoring pedestrians, for instance, an agent can be bound to a particular pedestrian. The pedestrian's position and orientation can then provide the long-term information missing from the data-driven technique. Mismatched actions of agents compared to real-world entities could then be used to determine how far observations are from "expected" results. At its simplest form anomalies could be found by assuming an agent should not be wrong more often than its models' error rates, with the butterfly effect of incorrect steps mitigated by synchronizing to the agent's real-world counterpart.

Finally, the data-driven algorithm presented here could be useful for short-term agent activity, perhaps while amortizing the calculation of a higher-quality plan. Due to the effects of errors on the overall simulation, long-term simulations requiring coordination would be unsuitable application areas for this work. Evacuation scenarios, for instance, would not be a natural fit to our technique at this time. Instead the collection of policies can be used to help agents progress to their goals while more elaborate algorithms can plan further ahead and correct future actions for previous suboptimal data-driven decisions.

8.2 Future Work

This dissertation provides a context-sensitive approach to developing policies for agent steering and also opens areas for further exploration. Particular focus could be made on creating a better oracle algorithm to serve as the underlying basis for our data. Additionally there is the possibility of new ways to derive collections of contexts rather than intuition or unsupervised learning. Creating sets of contexts for simulations with different purposes would extend the paradigm an additional level and acknowledge, for example, the difference between an evacuation and ordinary

pedestrian traffic.

8.2.1 Further Oracle Improvements

The oracle algorithm used for generating synthetic data was used for its theoretical ability to solve any steering problem. As the current oracle is based on IDA* a tradeoff is required between the completeness of the oracle and the time necessary to reach the solution. A better oracle could exist because of this compromise and there are three components of the oracle which could be changed while searching for an improved algorithm.

Heuristic Function Alternatives

Recall our heuristic function, reproduced in Equation 8.1. Minimizing the heuristic value involves minimizing the number of steps to reach the goal state. Generally this implies favoring the lowest energy cost but as we discovered previously there is a problem regarding situations when agents should temporarily stop. This is advantageous for allowing other agents to pass before continuing forward. As we have discussed in Chapter 7, a temporary stopping action is difficult to justify given this heuristic function.

$$h(\mathbf{p}, \mathbf{g}) = \frac{\|\mathbf{p} - \mathbf{g}\| \cdot \text{energy}_{\text{avg}}}{\text{stride}_{\text{avg}}} \tag{8.1}$$

We theorize a more realistic heuristic function would be one which represents both the energy cost and the expected time remaining to reach the goal. Thus stopping to yield the right-of-way to another agent can incur a more natural penalty to cost since the agent must wait while not expending energy, however the energy cost from a more indirect route which does not include pausing can help constrain the

search. With such a heuristic, an agent could discover that the overall fastest way to reach its goal is to wait, rather than always be in motion. An analogy would be the meter of a taxicab, where the fare increases for both waiting and distance traveled. Thus standing still can be the better option when too much extra distance is added to avoid the wait, which would encourage the agent to more appropriately use its stationary action.

Trajectory Planning

We chose to use footsteps as the action space of the oracle because classification-based learning could be used for generating the policies. These learning algorithms often have higher variance which allow for fitting more complex models. An alternative worth further investigation is to use the trajectories from footstep sequences as the results of the policies. These trajectories are continuous, rather than discrete, which allows for the expansion of machine learning experiments to consider regression. Regression models are more adaptive than classification as several results can be combined to a customized result depending on the output from the model. At runtime, these trajectories would be selected and the agent can find steps to walk along the chosen path.

8.2.2 Failure-Based Context Generation

This dissertation has explored two strategies for identifying steering contexts: differences based on intuition and differences identified by clustering algorithms. In both cases the differences were decided *a priori* and were not reinforced using any information from the success or failure of the resulting policies. Future work could identify those scenarios for which the policies fail and use those scenarios to define new contexts. The process could then be repeated to find new failure cases

and merge contexts when one policy can do the work of two. Our context-sensitive crowd steering could adapt from a pipeline based on clustering, to one bootstrapped by clustering and procedurally refined with minimal human input.

8.2.3 Purpose-Dependent Context Sets

All of the simulations used for this work were pedestrian simulations where the specific goals and manner in which the agent reaches that goal is not crucial to the overall result. In an evacuation or similar safety study, these details are often very important as are the agents' roles in the scenario. For example, some agents may be designated as leaders or be more erratic in their behavior to represent panic. Features for these added details would need to be added to extend this pipeline into these application areas. Additionally, the new features would motivate the creation of purpose-dependent context sets. Since this process fits within the overall paradigm of this dissertation, it may also be possible to link together the purpose-based contexts at a higher level and allow agents to transition between categories at runtime as well.

Appendix A

Original Context ID Numbers

For brevity, this dissertation uses IDs to refer to the intuitively defined contexts. The following is a full enumeration of these contexts with respect to traffic patterns. Recall that the first twelve IDs are repeated for scenarios which also have static obstacles present. Thus contexts 12–23 are the same as 0–11 with respect to traffic patterns.

Context 0: Clear. The agent has an insignificant number of neighbors nearby, if any.

Context 1: Light oncoming. The agent has a small number of neighbors nearby that are walking against its velocity.

Context 2: Medium oncoming. The agent has a more significant number of neighbors which are walking against its velocity.

Context 3: Heavy oncoming. The agent has a large number of neighbors, 8 or more, which are walking against its velocity.

Context 4: Group oncoming. The agent is a member of a group walking at opposing velocity to another group of agents.

Context 5: Winning-side oncoming. The agent is a member of a group walking at opposing velocity to only one or two neighbors.

Context 6: Light crossing. The agent has a small number of neighbors nearby which are walking perpendicular to its velocity.

Context 7: Medium crossing. The agent has a more significant number of neighbors which are walking perpendicular to its velocity.

Context 8: Heavy crossing. The agent has a large number of neighbors, 8 or more, which are walking perpendicular to its velocity.

Context 9: Group crossing. The agent is a member of a group walking perpendicular to to another group of agents.

Context 10: Winning-side crossing. The agent is a member of a group walking perpendicular to only one or two neighbors.

Context 11: Chaos. The agent has a significant number of neighbors nearby whose relative velocities do not form a coherent pattern.

Appendix B

Details of the Final Oracle Planner

Due to the oracle algorithm's role as provider of all synthetic data used in this dissertation, choices made during implementation have a potentially large effect on all subsequent results. This appendix serves as an account of the changes made to the algorithm and observations of the generated behavior.

B.1 Basis

At its core, the oracle planner is derived from the IDA* planning algorithm. This algorithm represents one of the greatest speed-for-memory tradeoffs in computing as it will, under certain circumstances[1], provide an optimal solution while minimizing memory consumption. These characteristics are made possible by limiting the total cost of possible solutions during the search and only increasing these limitations if no solution is found.

Essentially, the technique converts an optimization problem into a decision problem. Rather than directly ask, "What is the lowest cost of a solution," we can iteratively ask, "Is there a solution with x cost" with increasing values for x. IDA* answers

this question by performing a bounded search using the heuristic function to predict if such a solution *could* exist. The algorithm keeps track of the minimum exceeding cost during the search, and if it is determined that no viable solution exists with the current cost limitation, the limitation is increased to the minimum exceeding cost and the search restarts with this new boundary.

The constraint placed on cost restricts how far down a suboptimal solution the search can progress before ending. Unlike with A*, only one possible solution is tracked at a time as the rising ceiling should ultimately only allow an optimal path to the goal and disqualifies suboptimal paths. The restarting of the search is what provides previously disqualified paths a new chance under a new cost limitation. However, restarting the search also delivers a steep blow to performance because the computation must be repeated[2].

B.2 Domain-Specific Considerations

In this dissertation's application, we used planning for identifying footsteps in multiagent simulations which lead to goal locations. The branching factor was high, as 18 possible steps were considered as the next potential step. While it was possible that not all steps were valid at any given time due to neighboring agents, obstacles, or the inverted pendulum model used for step mechanics, the full branching factor of 18 still leads to many possible paths, which favors the use of IDA* as the memory requirement for a single solution remains relatively small compared to keeping so many potential solutions for future expansion.

Although the oracle's use as an offline universal algorithm specifically permits for long computation times, we still had real-world time constraints on how long we could afford to wait for results. Thus the repetitive computation is not *prima facie*

proof of IDA*'s poor fit to the application, but rather the concern is a matter of how many times the computation is repeated. The number of restarts depends on how many times the cost limit may need to be raised, which in turn depends on how much the cost limit is raised on a particular iteration. For our application, the difference in cost can be very small, less than 1. In our initial empirical analyses of the oracle's runtime, differences in cost limit between iterations were as low as 0.0005.

The final blow to the use of pure IDA* was proving the heuristic function was not, as originally intended, admissible. The heuristic function is an estimate of the energy cost to the goal and based on the number of steps to cover a straight-line trajectory to the goal and the average energy cost per step. While often an underestimation, as a straight-line trajectory typically includes obstacles that must be steered around, the average energy per step is a problem. It is empirically estimated and thus has uncertainty involved, but worse as an average it is immediately implied that there are steps with less energy. Their inclusion would create a solution with less energy than the heuristic.

B.3 Algorithm Alterations

Since the branching factor of our problem space is too high for A* to serve as our oracle, the time penalty is poorly defined and empirically excessive for IDA*, and the heuristic function already strips the algorithm of its optimality guarantee, changes were required of the oracle. We decided to use a hybrid of the two algorithms based on memory bounding.

Memory was constrained by placing a limit on the number of nodes which could be expanded by A*. Since we could not guarantee optimality, the search could be continued to find other solutions in the same memory-bound and the best solution

retained. If a solution is found during a particular memory bound, it is returned by the algorithm. If no solution is found, the memory bound is raised. If the memory bound exceeds the memory available by the computer, IDA* could be used as a last resort. The algorithm could, if not practically then theoretically, find the solution to any solvable steering challenge.

The time was mitigated by the modifications back towards A* and also through the constraints on the actual problems presented to the algorithm. Since the number of steps to reach a goal does depend on the distance to the goal, scenarios given as input to the planner were kept short, about the width of 20 agents shoulder-to-shoulder. In more general use, waypoints would need to be generated as milestones to the goal and allow for piecemeal planning.

Notes

[1]The heuristic function must be admissible, which means it cannot overestimate the cost of a solution.

[2]This is also the memory tradeoff seen in dynamic programming.

Bibliography

[1] Ahn, J., Wang, N., Thalmann, D., and Boulic, R. Within-crowd immersive evaluation of collision avoidance behaviors. *Proc. 11th ACM SIGGRAPH Int. Conf. Virtual-Reality Contin. its Appl. Ind. - VRCAI '12* (2012), 231.

[2] Akaike, H. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on 19*, 6 (Dec 1974), 716–723.

[3] Allbeck, J. M. Carosa: A tool for authoring npcs. In *MIG* (2010), R. Boulic, Y. Chrysanthou, and T. Komura, Eds., vol. 6459 of *Lecture Notes in Computer Science*, Springer, pp. 182–193.

[4] Andrade, E. L., and Fisher, R. B. Simulation of crowd problems for computer vision. In *Work. Crowd Simul.* (2005).

[5] Arthur, D., and Vassilvitskii, S. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2007), SODA '07, Society for Industrial and Applied Mathematics, pp. 1027–1035.

[6] Becket, W. M. *Reinforcement learning of reactive navigation for computer animation of simulated agents*. PhD thesis, University of Pennsylvania, 1997.

[7] Bellman, R. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[8] Bentivegna, D. C., Atkeson, C. G., and Cheng, G. Learning From Observation and Practice Using Behavioral Primitives: Marble Maze. *2004 AAAI Fall Symp. Ser. / Work. Notes Real-Life Reinf. Learn.* (2004).

[9] Boatright, C. D., Kapadia, M., Shapira, J. M., and Badler, N. I. Pedestrian Anomaly Detection using Context-Sensitive Crowd Simulation. In *First International Workshop on Pattern Recognition and Crowd Analysis* (2012).

[10] Boatright, C. D., Kapadia, M., Shapira, J. M., and Badler, N. I. Context-sensitive data-driven crowd simulation. In *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry* (New York, NY, USA, 2013), VRCAI '13, ACM, pp. 51–56.

[11] Boatright, C. D., Kapadia, M., Shapira, J. M., and Badler, N. I. Generating a multiplicity of policies for agent steering in crowd simulation. *Computer Animation and Virtual Worlds* (2014), n/a–n/a.

[12] Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (New York, NY, USA, 1992), COLT '92, ACM, pp. 144–152.

[13] Chernova, S., and Veloso, M. Confidence-Based Policy Learning from Demonstration Using Gaussian Mixture Models. In *Proc. 6th Int. Jt. Conf. Auton. Agents Multiagent Syst.* (New York, New York, USA, 2007), ACM, pp. 223:1–233:8.

[14] Courty, N. Data-driven animation of crowds. In *Int. Conf. Comput. Vision/Computer Graph. Collab. Tech.* (2007), pp. 377–388.

[15] Durupinar, F., Pelechano, N., Allbeck, J., Güdükbay, U., Badler, N. I., and Gudukbay, U. How the Ocean Personality Model Affects the Perception of Crowds. *IEEE Comput. Graph. Appl. 31*, 3 (2011), 22–31.

[16] Fiorini, P., and Shiller, Z. Motion planning in dynamic environments using velocity obstacles. *Int. J. Rob. Res. 17*, 7 (1998), 760–772.

[17] Frank, E., and Witten, I. H. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning* (San Francisco, CA, USA, 1998), ICML '98, Morgan Kaufmann Publishers Inc., pp. 144–151.

[18] Funge, J., Tu, X., and Terzopoulos, D. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proc. 26th Annu. Conf. Comput. Graph. Interact. Tech.* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 29–38.

[19] Gaines, B. R., and Compton, P. Induction of ripple-down rules applied to modeling large databases. *J. Intell. Inf. Syst. 5*, 3 (Nov. 1995), 211–228.

[20] Guo, D., Wang, C., and Wang, X. A hierarchical pedestrians motion planning model for heterogeneous crowds simulation. In *2009 Int. Conf. Inf. Autom.* (June 2009), Ieee, pp. 1363–1367.

[21] Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., and Dubey, P. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *Proc. 2009 ACM SIGGRAPH/Eurographics Symp. Comput. Animat.* (2009), ACM, pp. 177–187.

[22] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The WEKA data mining software: an update. *SIGKDD Explor. 11*, 1 (2009), 10–18.

[23] Heïgeas, L., Luciani, A., Thollot, J., and Castagné, N. A physically-based particle model of emergent crowd behaviors. In *Graphicon* (2003), Citeseer, pp. 5–10.

[24] Helbing, D., and Molnar, P. Social force model for pedestrian dynamics. *Phys. Rev. E 51*, 5 (1995), 4282.

[25] Hughes, R. L. A continuum theory for the flow of pedestrians. *Transp. Res. Part B Methodol. 36*, 6 (July 2002), 507–535.

[26] Hughes, R. L. The Flow of Human Crowds. *Annu. Rev. Fluid Mech. 35*, 1 (Jan. 2003), 169–182.

[27] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural Comput. 3*, 1 (Mar. 1991), 79–87.

[28] Jolliffe, I. T. *Principal Component Analysis*, second ed. Springer, 2002.

[29] Kapadia, M., Singh, S., Hewlett, W., and Faloutsos, P. Egocentric affordance fields in pedestrian steering. *Proc. 2009 1*, 212 (2009), 215–224.

[30] Kapadia, M., Singh, S., Hewlett, W., Reinman, G., and Faloutsos, P. Parallelized Egocentric Fields for Autonomous Navigation. *Vis. Comput.* (2012), 1–19.

[31] Kapadia, M., Wang, M., Singh, S., Reinman, G., and Faloutsos, P. Scenario space: Characterizing coverage, quality, and failure of steering algorithms. In

Proc. 2011 ACM SIGGRAPH/Eurographics Symp. Comput. Animat. (2011), ACM, pp. 53–62.

[32] Karp, R. M. Reducibility among combinatorial problems. *Complex. Comput. Comput.* (1972), 85–103.

[33] Lee, K. H. K., Choi, M. G. M., Hong, Q., and Lee, J. Group behavior from video: a data-driven approach to crowd simulation. In *Proc. 2007 ACM SIGGRAPH/Eurographics Symp. Comput. Animat.* (2007), vol. 1, Eurographics Association, pp. 109–118.

[34] Lerner, A., Chrysanthou, Y., and Lischinski, D. Crowds by Example. *Comput. Graph. Forum 26*, 3 (Sept. 2007), 655–664.

[35] Lerner, A., Chrysanthou, Y., Shamir, A., and Cohen-Or, D. Data Driven Evaluation of Crowds. In *Proc. 2nd Int. Work. Motion Games* (2009), Springer, pp. 75–83.

[36] Lerner, A., Chrysanthou, Y., Shamir, A., and Cohen-Or, D. Context-Dependent Crowd Evaluation. *Comput. Graph. Forum 29*, 7 (2010), 2197–2206.

[37] Levine, S., Lee, Y., Koltun, V., and Popović, Z. Space-time planning with parameterized locomotion controllers. *ACM Trans. Graph. 30*, 3 (May 2011), 1–11.

[38] Lloyd, S. Least squares quantization in pcm. *Information Theory, IEEE Transactions on 28*, 2 (Mar 1982), 129–137.

[39] Lopez, T., Lamarche, F., and Li, T. Space-?time planning in changing environments: using dynamic objects for accessibility. *Comput. Animat. Virtual ...*, March (2012), 87–99.

[40] Loscos, C., Marchal, D., and Meyer, A. Intuitive crowd behavior in dense urban environments using local laws. *Proc. Theory Pract. Comput. Graph. 2003.* (2003), 122–129.

[41] Matthews, B. W. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure 405*, 2 (1975), 442–451.

[42] McCallum, A., Nigam, K., and Ungar, L. H. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2000), KDD '00, ACM, pp. 169–178.

[43] Metoyer, R. A., and Hodgins, J. K. Reactive pedestrian path following from examples. In *16th Int. Conf. Comput. Animat. Soc. Agents* (Nov. 2003), vol. 20, pp. 149–156.

[44] Musse, S., Jung, C., and Braun, A. Simulating the motion of virtual agents based on examples. *ACM/EG Symp. Comput. Animat.* (2006), 2006–2006.

[45] Musse, S. R., Jung, C. R., Jacques Jr., J. C. S., and Braun, A. Using computer vision to simulate the motion of virtual agents. *Comput. Animat. Virtual Worlds 18* (2007), 83–93.

[46] Ondřej, J., Pettré, J., Olivier, A. A.-H. H., and Donikian, S. A synthetic-vision based steering approach for crowd simulation. *ACM Trans. Graph. 29*, 4 (July 2010), 123.

[47] Patil, S., Berg, J. V. D., Curtis, S., Lin, M. C., and Manocha, D. Directing crowd simulations using navigation fields. *IEEE Trans. Vis. Comput. Graph. 17*, 2 (Feb. 2011), 244–54.

[48] Pearl, J. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society* (1985), pp. 329–334.

[49] Pelechano, N., Allbeck, J., and Badler, N. *Virtual Crowds: Methods, Simulation, and Control*. Morgan & Claypool, 2008.

[50] Pelechano, N., Allbeck, J. M., and Badler, N. I. Controlling individual agents in high-density crowd simulation. In *Proc. 2007 ACM SIGGRAPH Eurographics Symp. Comput. Animat.* (2007), D. Metaxas and J. Popovic, Eds., vol. 1 of *SCA '07*, Eurographics Association, p. 108.

[51] Pelleg, D., and Moore, A. W. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning* (San Francisco, CA, USA, 2000), ICML '00, Morgan Kaufmann Publishers Inc., pp. 727–734.

[52] Quinlan, J. R. Induction of decision trees. *Mach. Learn. 1*, 1 (Mar. 1986), 81–106.

[53] Quinlan, J. R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[54] Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 25–34.

[55] Reynolds, C. W. Steering behaviors for autonomous characters. In *Game Dev. Conf.* (1999), vol. 1999, Citeseer, pp. 763–782.

[56] RuleQuest. See5/c5.0. `http://rulequest.com`, 2014.

[57] Shao, W., and Terzopoulos, D. Autonomous pedestrians. *Graph. Models 69*, 5-6 (Sept. 2007), 246–274.

[58] Singh, G. *Categorising the Abnormal Behaviour from an Indoor Overhead Camera*. PhD thesis, VIT University, 2010.

[59] Singh, S., Kapadia, M., Faloutsos, P., and Reinman, G. SteerBench: a benchmark suite for evaluating steering behaviors. *Comput. Animat. Virtual Worlds 20* (2009), 533–548.

[60] Singh, S., Kapadia, M., Hewlett, B., Reinman, G., and Faloutsos, P. A modular framework for adaptive agent-based steering. In *ACM SIGGRAPH Symp. Interact. 3D Graph. Games* (2011), vol. 1, pp. 141–150.

[61] Singh, S., Kapadia, M., Reinman, G., and Faloutsos, P. Footstep navigation for dynamic crowds. *Comput. Animat. Virtual Worlds 22*, April (2011), 151–158.

[62] Sunshine-Hill, B., and Badler, N. Perceptually Realistic Behavior through Alibi Generation. In *Proc. 6th Artif. Intell. Interact. Digit. Entertain. Conf.* (2010).

[63] Thalmann, D., and Musse, S. R. *Crowd Simulation*, 2nd ed. No. May. Springer, 2012.

[64] Thawonmas, R., Hirayama, J., and Takeda, F. RoboCup Agent Learning from Observations with Hierarchical Multiple Decision Trees. *PRIMA2002* (2002).

[65] Torrens, P., Li, X., and Griffin, W. a. Building Agent-Based Walking Models by Machine-Learning on Diverse Databases of Space-Time Trajectory Samples. *Trans. GIS 15* (July 2011), 67–94.

[66] Torrens, P. M., Nara, A., Li, X., Zhu, H., Griffin, W. a., and Brown, S. B. An extensible simulation environment and movement metrics for testing walking behavior in agent-based models. *Comput. Environ. Urban Syst.* (Aug. 2011).

[67] Treuille, A., Cooper, S., and Popović, Z. Continuum crowds. *ACM Trans. Graph. 25*, 3 (July 2006), 1160.

[68] Turkay, C., Koc, E., and Balcisoy, S. Integrating Information Theory in Agent-Based Crowd Simulation Behavior Models. *Comput. J. 54*, 11 (Feb. 2011), 1810–1820.

[69] van den Berg, J., and Manocha, D. Reciprocal Velocity Obstacles for real-time multi-agent navigation. *2008 IEEE Int. Conf. Robot. Autom.* (May 2008), 1928–1935.

[70] Yersin, B., Maïm, J., Morini, F., and Thalmann, D. Real-time crowd motion planning. *Vis. Comput. 24*, 10 (Aug. 2008), 859–870.

[71] Yu, Q. *A Decision Network Framework for the Behavioral Animation of Virtual Humans*. PhD thesis, University of Toronto, 2007.

[72] Zhou, S., Chen, D., Cai, W., Luo, L., Yoke, M., Low, H., Tian, F., Tay, V. S.-H., Ong, D. W. S., and Hamilton, B. D. Crowd modeling and simulation technologies. *ACM Trans. Model. Comput. Simul. 20*, 4 (2010), 20:1–20:35.