Masters Thesis

Optimizing Simulated Crowd Behaviour

Glen Berseth

July 16, 2014

Contents

Abstract

In the context of crowd simulation, there is a diverse set of algorithms that model *steering*, the ability of an agent to navigate between spatial locations, while avoiding static and dynamic obstacles. The *performance* of steering approaches, both in terms of quality of results and computational efficiency, depends on internal parameters that are manually tuned to satisfy application-specific requirements. This work investigates the effect that these parameters have on an algorithm's performance. Using three representative steering algorithms and a set of established performance criteria, we perform a number of large scale optimization experiments that optimize an algorithm's parameters for a range of objectives.

For example, our method automatically finds optimal parameters to minimize turbulence at bottlenecks, reduce building evacuation times, produce emergent patterns, and increase the computational efficiency of an algorithm. Our study includes a statistical analysis of the correlations between algorithmic parameters, and performance criteria. We also propose using the *pareto-optimal front* as an efficient way of modelling optimal relationships between multiple objectives, and demonstrate its effectiveness by estimating optimal parameters for interactively defined combinations of the associated objectives. The proposed methodologies are general and can be applied to any steering algorithm using any set of performance criteria.

Chapter 1

Introduction

Simulating groups of autonomous virtual humans (agents) in complex, dynamic environments is an important issue for many practical applications. A key aspect of autonomous agents is their ability to navigate (steer) from one location to another in their environment, while avoiding collisions with static as well as dynamic obstacles. The requirements of a steering approach differ significantly between applications and application domains. For example, computer games are generally concerned with minimizing computational overhead, and often trade off quality for efficiency, while evacuation studies often aim to generate plausible crowd behaviour that minimizes evacuation times while maintaining order.

There is no definitive solution to the steering problem. Most of the established methods are designed for specific classes of situations (scenarios), and make different trade-offs between quality and efficiency. The fine balance between these often competing performance criteria is governed by algorithm specific parameters that are exposed to the user. Some of these parameters have intuitive direct effects. For example, the radius of a *comfort zone* affects how close agents may come to each other, while the *neighbour horizon* limits the distance from an agent within which other agents are considered during steering. This significantly influences both the predictive power and computational efficiency of the associated method. However, even when the parameters are fairly intuitive, their combined effect, or their effect on the macroscopic behaviour of a large crowd, is not always easy to predict. For this reason, the inverse question is particularly interesting. Given a pattern of behaviour, a performance criterion (metric) or a trade-off between performance metrics, can we automatically select the parameter values of a steering algorithm that will produce the desired effect? This is a timely and important question, and the main focus of our work.

We begin our study look at the independent effects of parameters. Using simple a equidistant parameter sampling strategy we analyze the effects individual parameters have on different performance measures. We perform additional correlation analysis over the parameters and the metrics to identify parameters that effect different performance measures the most. It can easily be shown that these methods are incapable of finding a globally optimal solution but the finds can be used in a number of way. For example Using the correlation analysis to find s subset of the parameters for a steering algorithm that are most important and limit the number of parameters used while optimizing.

We present a methodology for automatically fitting the parameters of a steering algorithm to minimize *any combination* of performance metrics across *any set* of environment benchmarks in a general, model-independent fashion. Using our approach, a steering algorithm can be optimized for the following: success; quality with respect to distance, time, or energy consumption of an agent; computational performance; similarity to ground truth; user-defined custom metrics; or, a weighted combination of any of the above. Optimizing an algorithm's parameters across a representative set of challenging scenarios provides a parameter set that generalizes to many situations. A steering approach may also be fitted to a specific benchmark (e.g., a game level), or a benchmark category (e.g., evacuations) to hone its performance for a particular application.

We demonstrate our proposed methodology using three established agent-based algorithms: (1) **ORCA**: a predictive technique that uses reciprocal velocity obstacles for collision avoidance [van den Berg *et al.*, 2011], (2) **PPR**: a hybrid approach that uses rules to combine reactions, predictions, and planning [Singh *et al.*, 2011], and (3) **SF**: a variant of the social forces method for crowd simulation [Helbing *et al.*, 2000]. We thoroughly study these algorithms and compute their optimal parameter configurations for different metric combinations on a representative scenario set of local agent interactions and large-scale benchmarks. For example, our method automatically finds optimal parameters to minimize turbulence at bottlenecks, reduce building evacuation times, produce emergent patterns, and increase the computational efficiency of an algorithm, in one case by a factor of two. Cross-validation shows that, on average, optimal parameter values generalize across scenarios that were not part of the test set. Our study includes an in-depth statistical analysis of correlations between algorithmic parameters and performance criteria, however, because of space limitations the complete analysis can be found in the supplemental material.

We also study the interesting and challenging problem of dynamically tuning the parameters of an algorithm to support interactively defined combinations of objectives. For most practical cases, it is not feasible to solve this problem in real-time every time the combination changes. To address this issue we precompute optimal trade-offs between the objectives in the form of a discrete approximation of the *pareto-optimal front*. During runtime, our method efficiently estimates the parameters of the algorithm that optimally support a new combination of the objectives.

1.1 Contributions

1. We propose a statistical framework that can be used to identify the relationship between a steering algorithm's parameters and a set of quality and performance objectives.

2. An analysis of the effects parameter changes have on a number of different steering algorithms

3. A set of optimal parameter settings for each of the steering algorithms for each of the objectives used.

4. A model-independent solution that automatically fits a steering algorithm's parameters to maximize its performance, and we demonstrate its effectiveness with a use-case analysis of many popular crowd simulation techniques.

5. A general method to produce a *pareto-optimal front* between a number of objectives that can be used to form a dynamic blending function between objectives.

1.2 Outline

The rest of this work is organized in a number of chapters. Chapter 2 describes of the recent work in the area of crowd simulation and related work on optimization methods used in the field of computer animation. In Chapter 3 we outline the software foundation used in this research. The methodology and mathematical formulation needed for optimization is found in Chapter 4. Chapter 5 contains results and discussion from early experiments and studies on independent parameters. Then in Chapter 6 we start work on multi-variet optimization and how well genetic algorithms can optimize the behaviour of crowds. Next, in Chapter 7 we move into multi-objective optimization and *pareto-optimal front* estimation were we also blend between sample points on the *pareto-optimal front*. Chapter 8 is a collection of additional results and use cases for the created framework.

Chapter 2

Related Work

Since the seminal work of [Reynolds, 1987; Reynolds, 1999], crowd simulation has been studied from many different perspectives. We refer the readers to comprehensive surveys [Pelechano *et al.*, 2008; Huerre *et al.*, 2010; Thalmann and Musse, 2013] and present a broad review below.

2.1 Steering Techniques

Centralized techniques [Milazzo *et al.*, 1998; Hoogendoorn, 2003; Henderson, 1971; Lovas, 1994; Treuille *et al.*, 2006] model the characteristics of the crowd flow rather than individual pedestrians. Such models are of value in computing macroscopic simulations involving thousands of agents (e.g. stadium evacuation scenarios, urban simulations etc). However, these approaches are unable to model specific agent-agent interactions which are crucial in a microscopic view of crowd simulations that are prevalent in today's games.

Continuum-based techniques [Treuille *et al.*, 2006; Narain *et al.*, 2009] model the characteristics of the crowd flow to simulate macroscopic crowd phenomena. Particle-based approaches [Reynolds, 1987; Reynolds, 1999] model agents as particles and simulate crowds using basic particle dynamics. The social force model [Helbing *et al.*, 2005; Pelechano *et al.*, 2007] simulates forces such as repulsion, attraction, friction and dissipation for each agent to simulate pedestrians. Rule-based approaches [Lamarche and Donikian, 2004; Sud *et al.*, 2007] use various conditions and heuristics to identify the exact situation of an agent. Egocentric techniques [Kapadia *et al.*, 2009; Kapadia *et al.*, 2012] model a local variable-resolution perception of the simulation. Data-driven methods [Lee *et al.*, 2007; Lerner *et al.*, 2007; Ju *et al.*, 2010; Boatright *et al.*, 2013] use existing video or motion capture data to derive steering choices that are then used in virtual worlds, and recent work [Ondřej *et al.*, 2010] demonstrates a synthetic vision-based approach to steering. The works of [Paris *et al.*, 2007; van den Berg *et al.*, 2011] use predictions to steer in environments populated with dynamic threats.

Commercial and open-source software [Regelous, ; Mononen, 2009; Axel Buendia, 2002; Singh *et al.*, 2009b] provide complete steering and navigation solutions using variations of the aforementioned techniques.

2.2 Crowd Evaluation

There has been a growing recent trend to use statistical analysis in the evaluation and analysis of crowd simulations. The work by Lerner *et al.* [2010] adopts a data-driven approach to evaluating crowds by measuring its similarity to real world data. Singh *et al.* [2009a] proposes a compact suite of manually defined test cases that represent different steering challenges and a rich set of derived metrics that provide an empirical measure of the performance of an algorithm. Recent extensions [Kapadia *et al.*, 2011a] propose a representative sampling of challenging scenarios that agents encounter in crowds to compute the coverage of the algorithm, and the quality of the simulations produced. Density measures [Lerner *et al.*, 2010] and fundamental diagram-based comparisons [Seyfried *et al.*, 2010] use aggregate metrics for quantifying similarity. The work in [Guy *et al.*, 2012; Pettré *et al.*, 2009] measures the ability of a steering algorithm to emulate the behaviour of a real crowd dataset by measuring its divergence from ground truth. [Musse *et al.*, 2012] presents a histogram-based technique to quantify the global flow characteristics of crowds. Perceptual studies rely on human factors experiments to measure the variety in appearance and motion [McDonnell *et al.*, 2008], or perceptual fidelity of relaxing collisions [Kulpa *et al.*, 2011] in crowds.

2.3 Parameter Optimization

Parameter fitting is widely used in visual effects [Bruckner and Moller, 2010] to automate the tuning of model parameters to meet certain user-defined criteria. The resulting optimization problems tend to involve non-convex, and high-dimensional spaces. For these problems evolutionary strategies are preferred because they generally have less parameters to tune and do not require the computation of derivatives. Such techniques have been successfully demonstrated on a diverse set of application domains [Ha *et al.*, 2013; Wang *et al.*, 2010]. By selecting the right set of parameters, researchers have shown improvements in a steering algorithm's ability to match recorded crowd data [Johansson *et al.*, 2007; Pettré *et al.*, 2009; Pellegrini *et al.*, 2009; Davidich and Koester, 2011; Lemercier *et al.*, 2012].

2.4 Concurrent Work

Concurrent work [Wolinski *et al.*, 2014] explores parameter estimation of steering algorithms to match reference data for specific scenarios. Our method is not tied to ground truth, and can be used to optimize quantitative metrics such as the computational performance of the algorithm. Additionally, we leverage the use of different test sets including small-scale interactions and high-density crowds, to obtain optimal parameter values that *generalize* across the space of possible scenarios. To offset the computational burden of optimizing an algorithm for different criteria, we propose a method to precompute the mapping between an algorithm's parameters and objective weights, thus allowing us to dynamically adapt the crowd behaviour at real-time rates.

Although prior work has entertained the notion of parameter tuning in certain specific cases, a methodology to identify the mapping between a steering algorithm's parameters and performance objectives has not been performed yet. Such a study is an important and timely next step, and it is the main focus of this paper

Chapter 3

Outline of the Framework

We built this framework off of a pre exhisting crowd simulator called SteerSuite. We briefly describe what SteerSuite is and the modifications that were done to the system in order to perform this research.

3.1 SteerSuite

SteerSuite is a modular framework that is used to simulate and evaluate steering algorithms. There exist few libraries that can be used to prototype and experiment with steer algorithms. The software is designed to make it easier to develop, test and analyze steering algorithms. The system includes a number of example scenarios, many steering algorithms and *SteerBench* [Singh *et al.*, 2009a], which can be used to profile the performance and behaviour of steering algorithms.

3.1.1 Steering Algorithms

Steering algorithms, or dynamic navigation algorithms are used to control the locomotion decisions of agents during a simulation. The navigation problem is complex because of the static and dynamic obstacles that exist in the environment. There are many methods that attempt to conquer this problem domain.

Every steering algorithm has a number of parameters that can be changed by the user. Changing the parameters of a steering algorithm results in exhibiting a different behaviour. for demonstration purposes, we use the following established algorithms that model different steering approaches.

1. **PPR**. [Singh *et al.*, 2011] presents a hybrid framework that combines reaction, prediction and planning. It is an example of a rule-based method for agent based steering and has 38 independent parameters. For example, *avoidance-turn-rate* defines the turning rate adjustment speed in proportion to the typical speed and *query-radius* controls the radius around an agent that **PPR** uses to predict collisions with other objects and agents.

2. **ORCA**. [van den Berg *et al.*, 2011] is a very popular method that uses optimal reciprocal collision avoidance to efficiently steer agents in large-scale crowds. A subset of its independent parameters are: *max-neighbors*, the maximum number of nearby agents that an agent will take into consideration when making steering choices; *max-speed*, the maximum speed that an agent may travel with; and *time-horizon*, the minimal time for which an agent's computed velocity is safe with respect to other agents.

3. **SF**. [Helbing *et al.*, 2000] uses hypothetical social forces for resolving collisions between interacting agents in dense crowds. In addition to general

parameters similar with the other methods, each social force model has associated parameters that govern its relative influence.

3.1.2 What is a Scenarios

A scenario is an initial configuration of a simulation. It can be thought of as the initial positions of the obstacles and agents in the simulation and additional information on the agents settings. Settings such as desired velocity and target location. The space of possible scenarios is considered to be infinite. A basic scenario can be seen in Figure 3.1.

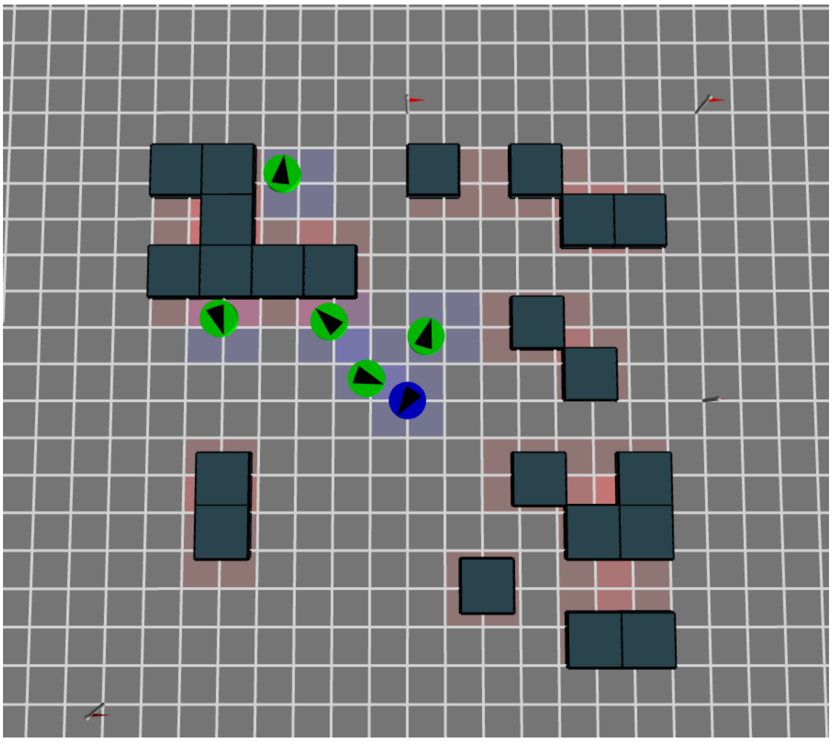


Figure 3.1: A basic scenario in SteerSuite with 6 agents. The goals of the agents are marked by the small flags and the dark green blocks are static obstacles.

3.1.3 Scenario Module

In crowd simulation it is common to evaluate steering algorithms over a small set of possible benchmarks. The scenario module is designed to have the capability to generate massive amounts of benchmarks. This method was first used in [Kapadia *et al.*, 2011b], where the total space of possible scenarios was considered and a formulation of a representative set of all of the scenario space was created. The scenario module is used significantly in this work as a primary means to generate and execute large amounts of scenarios.

3.2 SteerStats

SteerStats acts as a wrapper for SteerSuite in order to make the processes of calling and collecting the statistical information a single function. When using the scenario module all of the statistics for the simulation are recorded and can be accessed by SteerStats. The wrapper accepts many arguments that are passed to the scenario module when running SteerSuite that indicate, amongst many other things, the kinds of data to be collected from the simulation and the type of simulation to be executed.

3.2.1 SteerSuite Interface

This wrapping is not a direct Python wrapping of the C++ library. Instead, the wrapping calls the executable using a mechanism similar to a system call, passing all of the relevant simulation parameters to the executable program. The wrapping of SteerSuite is done in two parts. The first part is primarily designed to read, parse and organize the simulation data that is recorded by the scenario module. The second part is used to control the execution of SteerSuite and the arguments passed. In addition, various forms of parallelization are supported to allow for more efficient collection of data and execution of SteerSuite on multi-core systems.

3.2.2 SteerStats Database

The SteerStats framework also supports integration with the postgreSQL database[1]. This integration provides a number of useful features:

1. Facilitates full data recording
2. Allows for easier analysis/data mining
3. Gives a structured organization to the data

Schema

The schema used to organize the data in the data base is rather standard. A brief digram of the schema is presented in Figure 3.2. The organization uses join tables between the different tables to organize various types of data logged by the system. Each testcase is stored as a *test* in the database. Many tests can be associated with a single *test set*, which also stores the simulation configuration. If desired a video and binary recording of the simulation can also be associated with a testcase. Last, there are a number of tables for each steering algorithm that store information on the algorithms settings during simulation.

[1]http://www.postgresql.org/

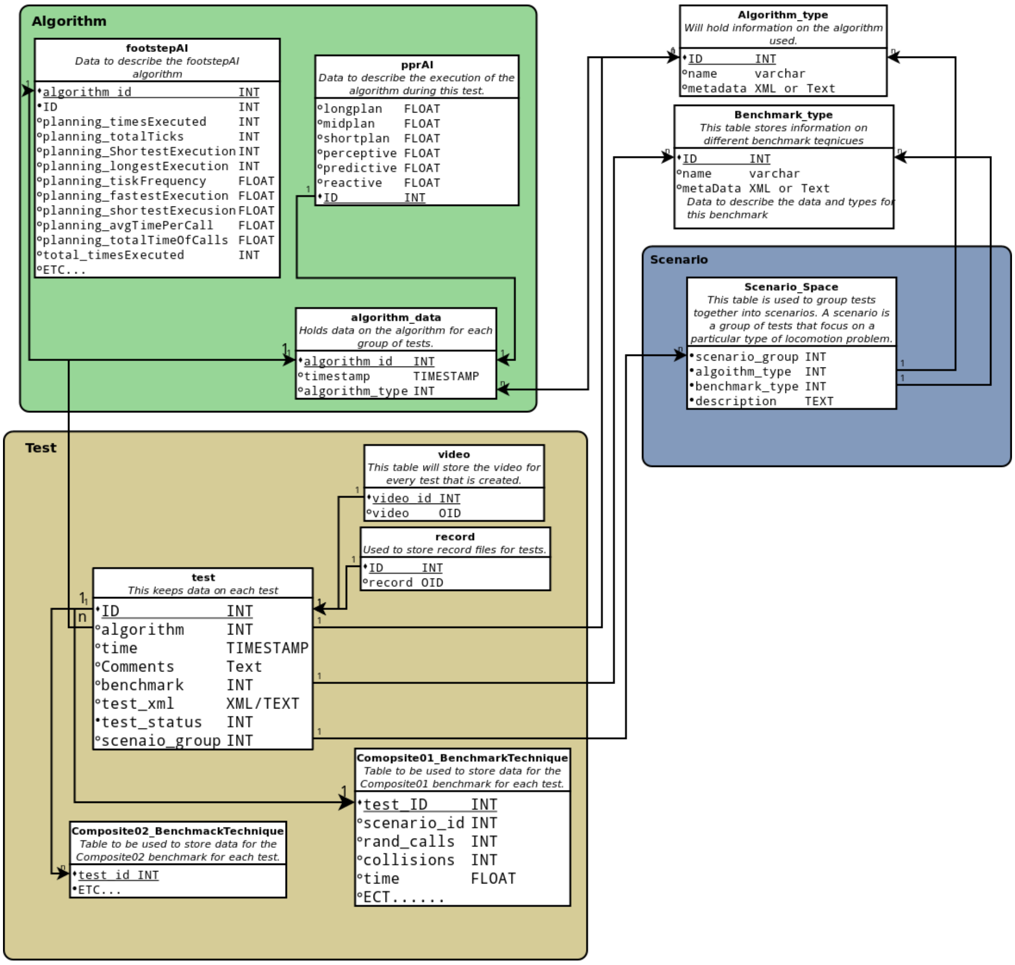


Figure 3.2: SteerStats database schema. For brevity, not all of the columns in the tables are listed in this figure.

3.2.3 Conclusion

This framework has been designed for ease of use. The goal is to make a call to SteerStats a single function which can be later be used by different methods to analyze the performance of steering algorithms.

Chapter 4

Preparation And Formulation

In this chapter we present a framework for analyzing the effects of parameters $\mathbf{v} \in \mathbb{V}$ of an algorithm, $A_{\mathbf{v}}$. The next sections describe the elements involved in this framework.

In order for this analysis to be more statically significant of the steering algorithms, a number of test sets are created. These test sets will be used across the steering algorithms for analysis and optimization. These test sets are also supposed to be the best known samplings of the scenario space so that the analysis and optimization results can be considered more general. We also define a number of objectives that are used to measure different elements of the steering algorithms performance and a weighted combination of these objectives.

4.1 Generating Test Sets

We employ different benchmark sets including local agent interactions and high-density crowds to find the optimal values of an algorithm's parameters that generalize across the wide range of situations that agents encounter in crowds. Note that certain performance metrics may have more meaning for specific test sets. For example, computational efficiency is more meaningful for situations that involve sufficiently large numbers of agents.

Large Scale Set. $\mathcal{S}$ contains most of the large-scale benchmarks in Table 4.1 that define large environments with many agents. $\mathcal{S}^v$ is a set of similar but different large-scale benchmarks that will be used to validate the results of parameter optimization on previously unseen cases (cross-validation).

Benchmark	# Agents	Description
`Random`	1000	Random agents in open space.
`Forest`	500	Random agents in a forest.
`Urban`	500	Random agents in an urban environment.
`Hallway`	200	Bi-directional traffic in a hallway.
`Free Tickets`	200	Random agents to same goal, then disperse.
`Bottleneck`	1000	Tight bottleneck.
`Bottleneck evac`	200	Evacuation through a narrow door.
`Concentric circle`	250	circle with target on opposite side.
`Concentric circle`	500	circle with target on opposite side.
`Hallway`	400	4-way directional traffic.

Table 4.1: Large scale benchmarks. The bottom three scenario are part of $\mathcal{S}^v$. All are designed to stress the steering algorithms computational efficiency.

Representative Set. The representative scenario set, $\mathcal{R}$, includes 5000 samples of a wide range of local interactions. It is designed to include challenging local scenarios and to exclude trivial or invalid cases. We construct it in a fashion similar to [Kapadia *et al.*, 2011a], following these general guidelines: (a) The reference agent is placed near the center of the scenario, (b) agent targets are placed at the environment boundary, and (c) non-reference agents are distributed at locations that maximize the likelihood that their static paths will intersect the reference agent's static path to its target. We use the same method to generate another set of the same size, $\mathcal{R}^v$, for cross-validation. We use the representative set because it provides the best sampling of the full space of possible scenarios. Therefore, optimizing for the representative set should give good results in general for any scenario.

Combined Test Set. The union of the large scale set, $\mathcal{S}$, and the representative set, $\mathcal{R}$, $\mathcal{T} = \mathcal{S} \cup \mathcal{R}$ is the main test set that we use for algorithm analysis and parameter fitting in a statistically significant general fashion. Here we use statistical significance to contrast against common practice in crowd simulation where results are demonstrated on a very limited number of test cases.

Combined Validation Set. Similarly, the combined cross-validation set is $\mathcal{T}^v = \mathcal{S}^v \cup \mathcal{R}^v$.

Custom Scenario Set. A user can specify a subset of scenarios in $\mathcal{T}$ or even design custom benchmarks to focus the parameter fitting on application-specific requirements. Random permutations in the environment configuration and agent placement can generate multiple samples of a custom benchmark category. For example, one can create a set of test cases that capture two-way traffic in orthogonally crossing hallways as is common in large buildings.

Ground Truth Test Set. There are few publicly available data sets of recorded crowd motion which can be used to test a steering algorithm's ability to match real world data. We use a ground truth test set $\mathcal{G}$, published by

[Seyfried *et al.*, 2010], for our experiments.

4.2 Performance Measures

Given an appropriate test set, we want to compute normalized quantities (metrics) that characterize important aspects of a steering algorithm's performance. Recently a number of intuitive performance metrics have been proposed that include: (a) the fraction of scenarios that an algorithm is unable to solve in a representative set of scenarios, (b) quality measures with respect to distance travelled, total time taken, or energy consumption of an agent, (c) computational performance of the algorithm, and (d) statistical similarity with respect to ground truth. The specific metrics that we use in our experiments are briefly described below. For more details see [Kapadia *et al.*, 2011a; Berseth *et al.*, 2013; Guy *et al.*, 2010; Guy *et al.*, 2012]. One can also define custom metrics to meet application-specific requirements.

Failure Rate. The coverage $c(A_\mathbf{v})$ of a steering algorithm $A_\mathbf{v}$ over a test set $\mathcal{T}$ is the ratio of scenarios that it successfully completes in $\mathcal{T}$. An algorithm successfully completes a particular scenario if the reference agent reaches its goal without any collisions and the total number of collisions among non-reference agents is less than the number of agents in the scenario. The failure rate is the complement of coverage $d(A_\mathbf{v}) = 1 - c(A_\mathbf{v})$. It captures the ratio between the number of scenarios not successfully solved and the total number of scenarios in $\mathcal{T}$, and it has obvious bounds in $\mathcal{T}$.

Distance Quality. For a single small scale scenario s we define the distance quality metric $q^{\mathrm{d}}(A_\mathbf{v})$ of an algorithm $A_\mathbf{v}$ as the complement of the ratio between the length of an ideal optimal path o_s^d, and the length of the path that the reference agent followed, a_s^d:

$$q^{\mathrm{d}}(A_{\mathbf{v}}) = 1 - \frac{o_s^d}{a_s^d}. \tag{4.1}$$

The ideal optimal path is the shortest static path from the agent's initial position to its goal after line-of-sight smoothing [Pinter, 2001]. If the algorithm does not successfully complete the scenario then the associated distance quality metric is set to the worst-case value of 1. For a large-scale scenario we compute $q^{\mathrm{d}}(A_{\mathbf{v}})$ as the average over all agents, and for a set of scenarios, we computed it as the average over the set.

Time Quality. Similarly, $q^{\mathrm{t}}(A_{\mathbf{v}})$ characterizes how much longer the reference agent took to reach its goal compared to an ideal optimal time. The ideal optimal time for a single scenario corresponds to the agent reaching its goal when moving with its desired velocity along the ideal optimal path. Defined as:

$$q^{\mathrm{t}}(A_{\mathbf{v}}) = 1 - \frac{o_s^t}{a_s^t}, \tag{4.2}$$

where a_s^t is the time it took the agent to reach its goal in the scenario s. If the algorithm does not successfully complete the scenario then the metric is set to the worst-case value of 1. For large scale scenarios this metric represents the average over all agents, and for a set of test cases the average over the set.

PLE Quality. The principal of least effort characterizes the energy expenditure of a reference agent over a path traveled [Guy *et al.*, 2010] as follows:

$$p^e = m \int_{t_{start}}^{t_{end}} (e_s + e_w)|\mathbf{v}|^2 dt, \tag{4.3}$$

where e_s and e_w are commonly used energy terms for the average person [Whittle, 2007], and the mass, m is set to 1 in our experiments. The PLE quality metric, $q^{\mathrm{e}}(A_{\mathbf{v}})$, is computed similar to the other metrics as follows:

$$q^{\mathrm{e}}(A_{\mathbf{v}}) = 1 - \frac{o^e}{a^e}, \tag{4.4}$$

where $o^e_s = 2 \cdot \text{optimal-path-length} \times (e_s + e_w)$ is the ideal optimal effort and a^e the actual effort of the agent. If the algorithm does not successfully complete the scenario the metric is set to the worst case value of 1. For many agents and/or test cases the metric is computed in the average sense.

Computational Efficiency. The computational efficiency $e(A_{\mathbf{v}})$ metric is the average CPU time consumed by all agents in all scenarios in a test set $\mathcal{S}$. Unlike the above normalized metrics, it is not straightforward to provide an ideal upper bound for $e(A_{\mathbf{v}})$. To provide a basis for normalization, we assume that 10% of all computational resources are allocated to the steering algorithm. Hence, the maximum time allocated to a steering algorithm every frame is n_{des}^{-1} seconds for a desired framerate of n_{des} fps. For every scenario s, the maximum time t^s_{max} allocated to every steering agent per frame is $0.1 \cdot (N \cdot n_{\mathrm{des}})^{-1}$ seconds, where N is the number of agents in s. Let t^s_{avg} be the average time spent per frame for all agents to reach a steering decision. The average computational efficiency e over a test set $\mathcal{S}$ is computed as follows:

$$e(A_{\mathbf{v}}) = 1 - \frac{\sum_{s \in \mathcal{S}} e_{\mathrm{s}}(A_{\mathbf{v}})}{|\mathcal{S}|}, \quad e_{\mathrm{s}}(A_{\mathbf{v}}) = \frac{t^{\mathrm{s}}_{\mathrm{max}}}{t^{\mathrm{s}}_{\mathrm{avg}}} \tag{4.5}$$

where $e_{\mathrm{s}}(A_{\mathbf{v}})$ is the efficiency of $A_{\mathbf{v}}$ for a particular scenario s, and $|\mathcal{S}|$ is the cardinality of the test set $\mathcal{S}$.

The desired framerate, n_{des}, provides an ideal upper bound for efficiency, analogous to the ideal upper bounds of the other metrics, and allows us to define a normalized efficiency metric. Normalized metrics can be combined more intuitively into optimization objectives in the forthcoming analysis. Alternatively, we could set the desired framerate to a very high value for all algorithms and attend to scaling issues later.

Similarity to Ground Truth. In addition to quantitatively characterizing the performance of a steering algorithm, we can also measure its ability to match ground truth. We compute a simulation-to-data similarity measure $g(A_{\mathbf{v}},\mathcal{G})$ [Guy *et al.*, 2012] which computes the entropy measurement of the prediction errors of algorithm $A_{\mathbf{v}}$ relative to a given example dataset, such as the test set $\mathcal{G}$ defined in Section 4.1.

4.3 Weighted Multi-Objective Optimization

Given a set of performance metrics such as the ones defined in Section 4.2, $\mathbb{M} = \langle d, q^{\mathrm{d}}, q^{\mathrm{t}}, q^{\mathrm{e}}, e\rangle$, we can define an objective function as a weighted combination of these metrics:

$$f(A_{\mathbf{v}}, \mathbf{w}) = \sum_{m_i \in \mathbb{M}} w_i \cdot m_i, \tag{4.6}$$

where $\mathbf{w} = \{w_i\}$ contains the weights which determine the relative influence of each individual metric. By choosing different sets of metrics and associated relative weights, we can define custom objectives. For a steering algorithm $A_{\mathbf{v}}$ with internal parameters $\mathbf{v} \in \mathbb{V}$, a set of test cases, and a desired objective $f(A_{\mathbf{v}}, \mathbf{w})$, our goal is to find the optimal parameter values $\mathbf{v}^*_{\mathbf{w}}$ that minimize the objective over the test set. This can be formulated as a minimization problem:

$$\mathbf{v}^*_{\mathbf{w}} = \arg\min_{\mathbf{v} \in \mathbb{V}} f(A_{\mathbf{v}}, \mathbf{w}). \tag{4.7}$$

This is generally a non-linear and non-convex optimization problem for the independent parameters, $\mathbf{v} \in \mathbb{V}$.

4.4 Conclusion

We have formulated a number of objective metrics that can be used to compare the behaviour of steering algorithms. These objectives have been formulated in a novel way that normalizes the result in order to make optimization smoother.

Chapter 5

Parameter Analysis

In this chapter the results of initial experiments using the framework are discussed. Starting with a uni-variate optimization process. After which correlations between the objectives and steering algorithm parameters, then between objectives and other objectives, are explored.

5.1 Uni-Variate Optimization

This section describes a prefatory analysis we performed to understand the effect of the independent parameters on an algorithm's performance, and serves as a precursor to the multi-variate analysis reported in the work in Chapter 6. By varying each parameter in isolation and studying its effects on the performance criteria, we can answer questions such as: Which parameters are important? What are the bad values we need to avoid? Are the default values good?

For reference, we first compute the deficiency $d(A_{\mathbf{v}})$, distance quality $q^{\mathrm{d}}(A_{\mathbf{v}})$, and efficiency $e(A_{\mathbf{v}})$ metrics for the test set, $\mathcal{T}$, for the **PPR** algorithm using

Algorithm	**v**	$d(A_{\mathbf{v}})$	$q^{\mathrm{d}}(A_{\mathbf{v}})$	$e(A_{\mathbf{v}})$	$f(A_{\mathbf{v}})$
PPR	`DEF`	0.39	0.49	0.96	0.61
	`UNI`	0.25	0.25	0.95	0.46

Table 5.1: Comparison of $d(A_{\mathbf{v}})$,$q^{\mathrm{d}}(A_{\mathbf{v}})$, $e_{\mathrm{s}}(A_{\mathbf{v}})$, and $f(A_{\mathbf{v}})$ which is the equally weighted combination of the 3 metrics for the **PPR** steering algorithm using: (a) `DEF`: default parameter values and (b) `UNI`: best parameter values obtained using uni-variate analysis.

their default parameters, provided in Table 5.1. For default parameter settings, **PPR** can respectively solve 62% of the sampled scenarios.

To study the effect of each parameter in isolation, we sample each parameter of the steering algorithm independently in a bounded interval taking 20 uniformly distributed samples. The parameter bounds are chosen separately for each parameter based on intuition, physical interpretation of the parameter, or default values provided by the algorithm's creators. Table 10.1 enumerates the bounds of the parameters for **PPR**.

We find that the deficiency of **PPR** is sensitive to 23 of its 38 parameters. For each parameter we can also identify its optimal value and make trade offs. Table 5.1 shows the maximum improvement in the value of the performance metrics that we can achieve using this analysis (labelled UNI in the table) compared to value of the metrics that correspond to the default values (labelled DEF in the table).

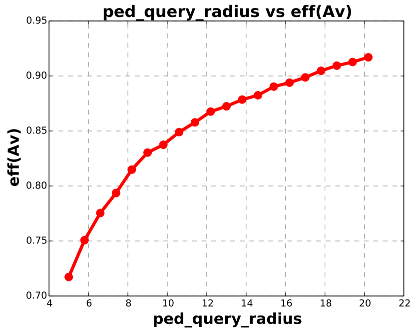
The deficiency for **PPR**, $d(A_{\mathbf{v}}^{ppr})$, decreases to 25% by selecting the optimal values of ped-avoid-rate and typical speed. The quality with respect to distance travelled for **PPR**, $q^{\mathrm{d}}(A_{\mathbf{v}}^{ppr})$, decreases to 0.25 for the optimal values of ped-avoid-rate and typical speed.

Efficiency is an important issue for steering algorithms. We can see that, as expected, the e of **PPR** decreases with the query radius. However, the more interesting observation comes from Table 5.1, where we can see that the e metric for the **PPR** algorithm improves when we use the appropriate parameter values from this analysis.

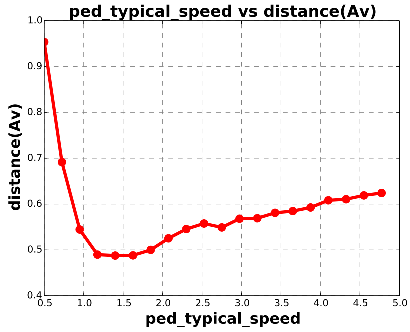
Optimizing for a weighted combination of all three metrics also yields interesting results. We observe that ped-avoid-rate $= 0.55$ produces optimal results in the **PPR** algorithm for an equal proportion of the 3 $d(A_{\mathbf{v}}^{ppr})$, $q^{\mathrm{d}}(A_{\mathbf{v}}^{ppr})$ and $e_{\mathrm{s}}(A_{\mathbf{v}}^{ppr})$.

Knowing how each parameter affects each performance metric, allows us to potentially focus our optimization efforts on specific parameters based on the requirements of an application. We can see in Tables (5.2-5.4) some examples of how single parameters for each steering algorithm can effect a particular metric. We found that ped-avoid-rate has little effect on efficiency, $e(A_{\mathbf{v}}^{ppr})$ while it does affect deficiency, $d(A_{\mathbf{v}}^{ppr})$, and quality, $q^{\mathrm{d}}(A_{\mathbf{v}}^{ppr})$. Therefore, it may be a suitable parameter to explore if we need to improve quality or deficiency without affecting efficiency.

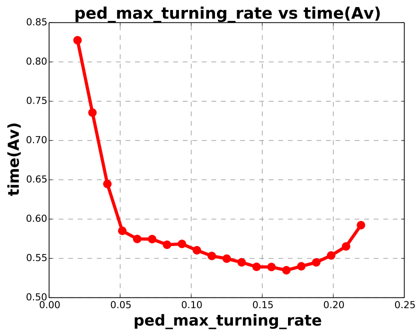
To gain insight on the simultaneous effect of multiple variables we perform one bi-variate analysis for **PPR**. Figure 5.1 shows the coverage of **PPR** with respect to the Cartesian product of two parameters, ped-avoid-rate, and typical speed. The shape of the resulting surface indicates that deficiency depends non-trivially on both parameters at the same time.



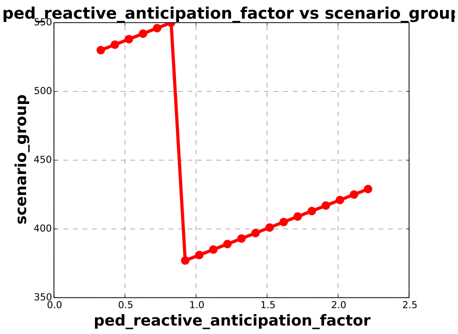
(a)



(b)



(c)



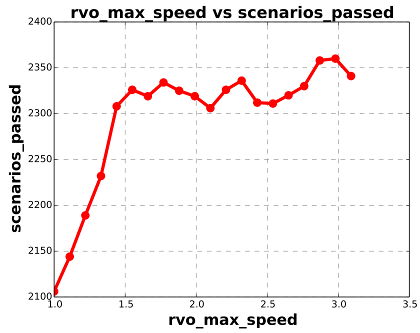
(d)

Table 5.2: Graphs of a few objectives for the **PPR** algorithms. These graphs show the effects a few of the parameters have on the objectives.

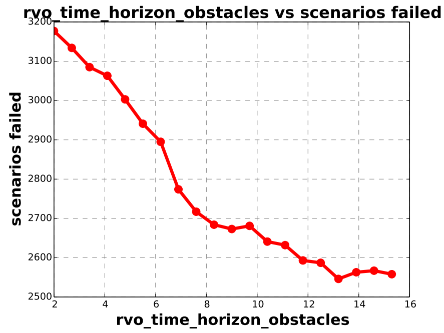
5.1.1 Discussion

The analysis in this section offers valuable insights on the effects of each parameter on the objectives.

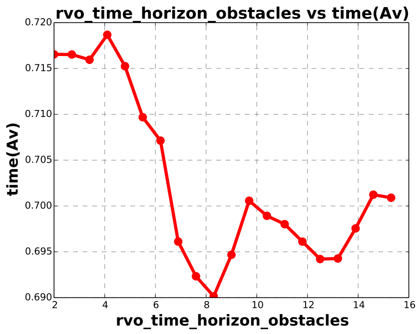
- We can easily identify which values of the parameters we should avoid, and which might be good choices.
- The experiments indicate that for an algorithm the default parameters are not necessarily optimal. They also verify that, as expected, the objectives are generally not separable functions of the parameters, v. We



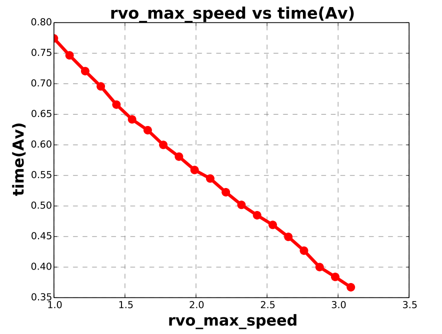
(a)



(b)



(c)

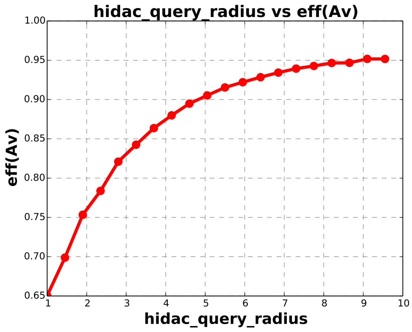


(c)

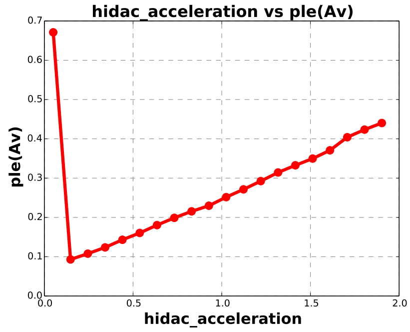
Table 5.3: Graphs of a few objectives for the **ORCA** algorithms. These graphs show the effects a few of the parameters have on selected objectives.

therefore need to fit the parameters simultaneously using a multi-variate optimization method.

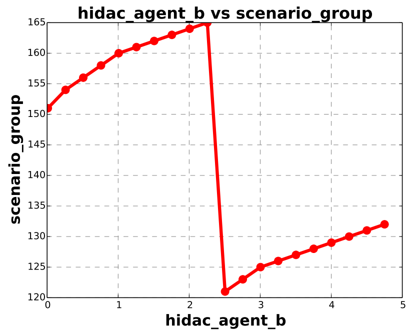
- For **PPR** we might be able reduced the number of parameters that we need to fit from 38 to the 23 that seem much more influential, which may significantly improve the time it takes to perform optimal fitting.



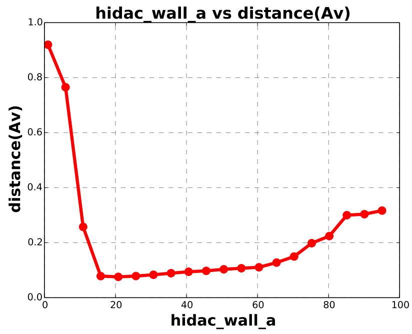
(a)



(b)



(c)



(d)

Table 5.4: Graphs of a few objectives for the **SF** algorithms. These graphs show the effects a few of the parameters have on selected objectives.

5.2 Parameter-Metric Analysis

It is interesting to identify which parameters change in relation to the objectives, and study the trade-offs that the algorithms essentially make with these changes. In our analysis we also computed correlations between the parameters of the steering algorithms and the metrics. A Spearman correlation is used because it computes a non-parametric correlation that is not based on any linearity and we suspect that the relationships between parameters and metrics and metrics and metrics are non-linear. Tables 5.5, 5.6, 5.7 list the results of this analysis.

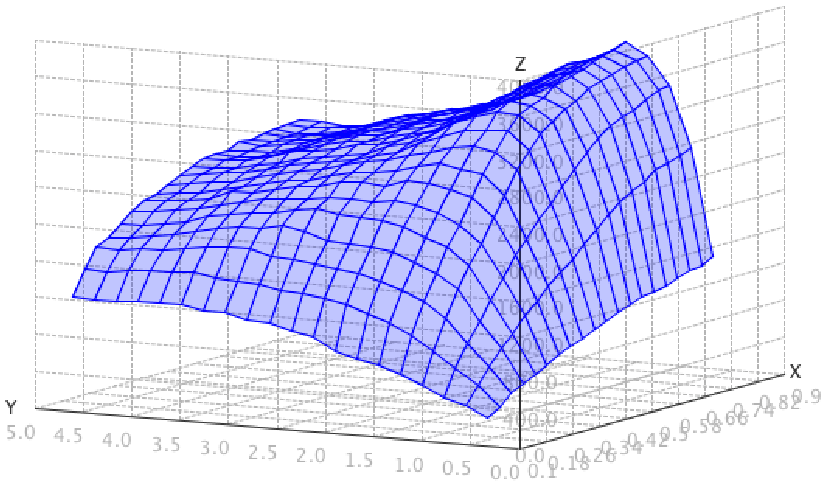


Figure 5.1: The coverage of **PPR**, $d(A_{\mathbf{v}}^{ppr})$ (vertical axis), with respect to two parameters, ped-avoid-rate and typical speed.

1. For **ORCA**, the maximum number of neighbours considered has the highest correlation with most metrics. The max speed seems to be the second most important parameter. It affects effort quality, $q^{\mathrm{e}}(A_{\mathbf{v}})$, negatively, and time quality $q^{\mathrm{t}}(A_{\mathbf{v}})$ positively.

2. For **PPR**, the max speed factor, which is a multiplier that increases the speed of an agent, is strongly correlated with the efficiency metric, e, and has a negative effect on all quality metrics.

3. For **PPR**, the size of the neighbourhood area and the distance to the furthest local target seem to be the parameters most strongly correlated with efficiency, e.

4. For **SF**, the parameters with the highest correlation to computational efficiency, e, have to do with proximity forces. When these are increased, agents push each other away forcefully, decreasing the likelihood that they

will interact again in the the next frame.

5. The parameters of **SF** that affect the quality measures the most are the wall repulsion coefficients.

The above analysis is not meant to be definite or complete, but rather to demonstrate that the proposed methodology can be notably more effective than manual tuning. The framework is an effective way to optimize, probe and analyze the behaviour of a steering algorithm in relation to its parameters, over a small or large set of test cases.

5.3 Metric-Metric Analysis

A correlation analysis clarifies the dependencies across metrics for a given algorithm. We generate 1000 samples in the parameter space of **ORCA**, and use them to compute each metric over the more than 5000 cases in $\mathcal{T}$. We then compute the Spearman correlation coefficients between pairs of metrics, shown in Tables (5.8 . We can identify the following correlations:

1. A weak negative correlation between computational efficiency, $e_{\mathrm{s}}(A_{\mathbf{v}})$, and the other metrics.

2. A strong negative correlation between time quality, $q^{\mathrm{t}}(A_{\mathbf{v}})$, and effort quality, $q^{\mathrm{e}}(A_{\mathbf{v}})$, which in general can be expected, as faster motion requires more energy.

3. A weak positive correlation between time quality, $q^{\mathrm{t}}(A_{\mathbf{v}})$, and distance quality, $q^{\mathrm{d}}(A_{\mathbf{v}})$. Also expected, since a shortest path often results in shorter completion time.

Parameter	$d(A_{\mathbf{v}})$	$q^{\mathrm{d}}(A_{\mathbf{v}})$	$q^{\mathrm{t}}(A_{\mathbf{v}})$	$q^{\mathrm{e}}(A_{\mathbf{v}})$	$e(A_{\mathbf{v}})$
max speed	−0.06	−0.12	−0.24	−0.04	−0.04
max force	−0.40	−0.41	−0.45	−0.38	−0.13
max speed factor	−0.58	−0.63	−0.72	−0.57	−0.23
faster speed factor	0.35	0.34	0.33	0.32	0.23
slightly faster speed factor	−0.06	−0.12	−0.25	−0.08	−0.06
typical speed factor	−0.40	−0.43	−0.62	−0.28	−0.26
slightly slower speed factor	0.30	0.28	0.28	0.26	0.00
slower speed factor	0.30	0.27	0.16	0.25	0.06
cornering turn rate	0.15	0.08	0.07	0.13	0.18
adjustment turn rate	−0.21	−0.24	−0.23	−0.22	−0.18
faster avoidance turn rate	−0.39	−0.39	−0.39	−0.35	−0.19
typical avoidance turn rate	−0.33	−0.34	−0.39	−0.37	−0.27
braking rate	−0.32	−0.28	−0.26	−0.27	−0.12
comfort zone	−0.30	−0.26	−0.26	−0.23	0.02
query radius	0.29	0.33	0.38	0.34	0.63
similar direction threshold	0.15	0.11	0.11	0.14	0.14
same direction threshold	0.52	0.55	0.64	0.52	0.11
oncoming prediction threshold	0.03	0.02	0.04	0.05	0.13
oncoming reaction threshold	−0.48	−0.50	−0.58	−0.49	−0.25
wrong direction threshold	0.23	0.25	0.29	0.23	0.05
threat distance threshold	0.12	0.10	0.14	0.13	0.00
threat min time threshold	0.38	0.40	0.46	0.37	0.19
threat max time threshold	−0.01	−0.04	−0.07	−0.00	0.02
predictive anticipation factor	−0.30	−0.29	−0.27	−0.28	−0.21
reactive anticipation factor	0.01	0.02	0.12	0.13	0.05
crowd influence factor	−0.35	−0.35	−0.38	−0.31	−0.12
facing static object threshold	0.21	0.21	0.27	0.18	−0.05
ordinary steering strength	0.04	0.03	0.07	0.02	0.04
oncoming threat avoidance strength	−0.25	−0.31	−0.35	−0.23	−0.16
cross threat avoidance strength	−0.08	−0.12	−0.18	−0.14	−0.01
max turning rate	0.43	0.35	0.33	0.29	0.17
feeling crowded threshold	−0.49	−0.53	−0.56	−0.46	−0.30
scoot rate	−0.12	−0.17	−0.24	−0.17	−0.11
reached target distance threshold	−0.26	−0.41	−0.44	−0.36	−0.30
dynamic collision padding	0.15	0.15	0.25	0.18	0.11
furthest local target distance	0.16	0.19	0.25	0.17	0.65
next waypoint distance	−0.07	−0.04	0.07	−0.07	0.01
max num waypoints	0.39	0.41	0.43	0.35	0.14

Table 5.5: This tables shows Spearman rank correlation coefficients between 5 metrics and all of the parameters for the **PPR** algorithm

Parameter	$d(A_\mathbf{v})$	$q^{\mathrm{d}}(A_\mathbf{v})$	$q^{\mathrm{t}}(A_\mathbf{v})$	$q^{\mathrm{e}}(A_\mathbf{v})$	$e(A_\mathbf{v})$
max speed	0.02	0.03	−0.34	0.58	0.14
neighbour distance	−0.09	−0.07	−0.13	−0.03	0.03
time horizon	−0.12	−0.08	0.10	0.04	0.07
time horizon obstacles	−0.09	−0.09	0.17	0.04	0.11
max neighbors	0.42	0.47	0.54	0.29	0.37

Table 5.6: This tables shows Spearman rank correlation coefficients between 5 metrics and all of the parameters for the **ORCA** algorithm.

Parameter	$d(A_\mathbf{v})$	$q^{\mathrm{d}}(A_\mathbf{v})$	$q^{\mathrm{t}}(A_\mathbf{v})$	$q^{\mathrm{e}}(A_\mathbf{v})$	$e(A_\mathbf{v})$
acceleration	0.14	0.18	0.15	0.18	−0.17
personal space threshold	−0.02	−0.01	−0.01	−0.01	0.02
agent repulsion importance	0.04	0.04	0.04	0.04	0.04
query radius	−0.01	−0.01	−0.01	−0.01	−0.00
body force	0.05	0.05	0.04	0.05	0.04
agent body force	0.00	0.01	0.00	0.01	−0.02
sliding friction force	0.00	0.01	0.01	0.01	−0.01
agent b	0.02	0.14	0.15	0.13	−0.37
agent a	−0.27	−0.21	−0.24	−0.21	−0.25
wall b	0.66	0.65	0.62	0.66	−0.01
wall a	0.37	0.37	0.34	0.37	−0.04

Table 5.7: This tables shows Spearman rank correlation coefficients between 5 metrics and all of the parameters for the **SF** algorithm

ORCA	$d(A_\mathbf{v})$	$q^{\mathrm{d}}(A_\mathbf{v})$	$q^{\mathrm{t}}(A_\mathbf{v})$	$q^{\mathrm{e}}(A_\mathbf{v})$	$e(A_\mathbf{v})$
$d(A_\mathbf{v})$	1	1.00	0.20	0.35	−0.18
$q^{\mathrm{d}}(A_\mathbf{v})$	1.00	1	0.21	0.36	−0.16
$q^{\mathrm{t}}(A_\mathbf{v})$	0.20	0.21	1	−0.63	−0.02
$q^{\mathrm{e}}(A_\mathbf{v})$	0.35	0.36	−0.63	1	−0.01
$e(A_\mathbf{v})$	−0.18	−0.16	−0.02	−0.01	1

Table 5.8: Spearman correlation coefficients between performance metrics for 1000 parameter samples with **ORCA**.

Chapter 6

Parameter Optimization

We present an optimization based framework for automatically fitting the parameters $\mathbf{v} \in \mathbb{V}$ of an algorithm, $A_{\mathbf{v}}$. Our framework automatically selects optimal parameter values $\mathbf{v}^* \in \mathbb{V}$ such that the performance of $A_{\mathbf{v}^*}$ minimizes certain performance criteria, over a set of benchmarks (test set). The next sections describe the elements involved in this problem and our approach to solving it.

6.1 Multi-Variate Optimization

The Covariance Matrix Adaptation Evolution Strategy technique (CMA-ES) [Hansen and Ostermeier, 1996; Hansen, 2011] is one of the many methods that can solve such problems. We chose CMA-ES because it is straightforward to implement, it can handle ill-conditioned objectives and noise, it is very competitive in converging to an optimal value in few iterations, and it has support for mixed integer optimization. The CMA-ES algorithm terminates when the objective converges to a minimum, when very little improvement is made between iterations, or after a fixed number of evaluations. Limiting the values of an algorithm's parameters transforms the problem of optimizing over an unbounded domain to a bounded one, which generally decreases the number of iterations needed for the optimization to converge. In most of our experiments the algorithm converged within 1000 evaluations.

Algorithm 6.1 describes the details of the CMA-ES algorithm we used for automatically selecting parameter values that optimize a given objective function.

The CMA-ES algorithm searches iteratively the parameter space for the optimal parameter values in an evolutionary fashion. At each iteration it generates N-samples of the parameter vector and keeps a subset of the samples that exhibit high fitness (minimize the objective). The algorithm then tries to increase the probability of successful candidate solutions and search steps, in a maximum-likelihood sense. The mean of the probability distribution of the samples is updated such that the likelihood of successful solutions is increased. A covariance matrix that captures the pair-wise dependencies between parameter distributions is also updated such that the likelihood of previously successful steps is increased. Samples are taken from a normal multivariate distribution with the computed mean and covariance matrix. A key feature of the algorithm is the way it controls the step size between iterations and the evolution paths. For more details see [Hansen and Ostermeier, 1996], and http://en.wikipedia.org/wiki/CMA-ES.

Example: Figure 6.2 illustrates an optimization process. The parameters of **ORCA** $\mathbf{v} = \{$max speed, neighbour distance, time horizon, time horizon obsta-

`input` Test set $\mathcal{T}$, Objective $f(A_{\mathbf{v}}, w)$, Algorithm $A_{\mathbf{v}}$, parameters $\mathbf{v} \in \mathbb{V}$

Initialize, mean m, covariance matrix C

while *not* `termination_condition` **do**

  while $i < N$ **do**

    $\mathbf{v}_i =$ `Sample` $\mathcal{N}(m, C)$

    `Compute Objective` $f_i = f(A_{\mathbf{v}_i}, w)$

  end while

  $\{\mathbf{v}_0, \mathbf{v}_1 ... \mathbf{v}_{N-1}\} = \arg\mathrm{sort}_{\{\mathbf{v}_i\}}(\{f_i | \forall i\})$

  $\mathbf{v}^* =$ `Update` $(\{\mathbf{v}_0, \mathbf{v}_1 \cdots \mathbf{v}_{N-1}\})$

  `Update Mean`, m

  `Update search paths`

  `Update Covariance Matrix`, C

end while

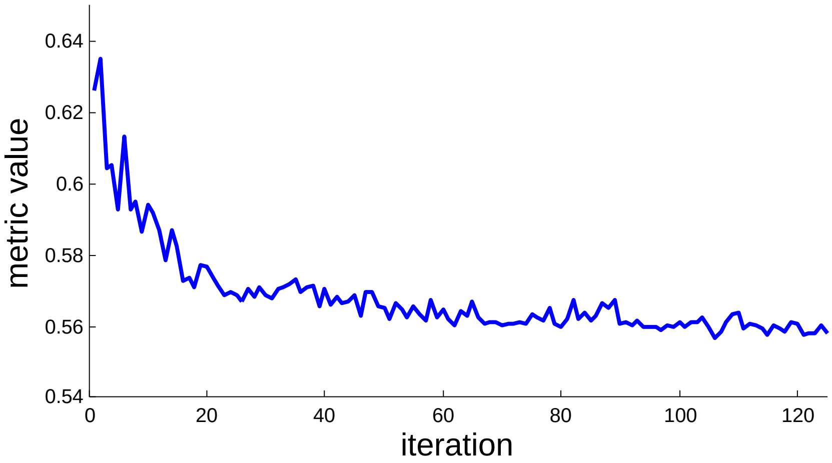
return $\mathbf{v}^*$

Figure 6.1: Main loop of CMA-ES Algorithm for parameter optimization of steering algorithms.

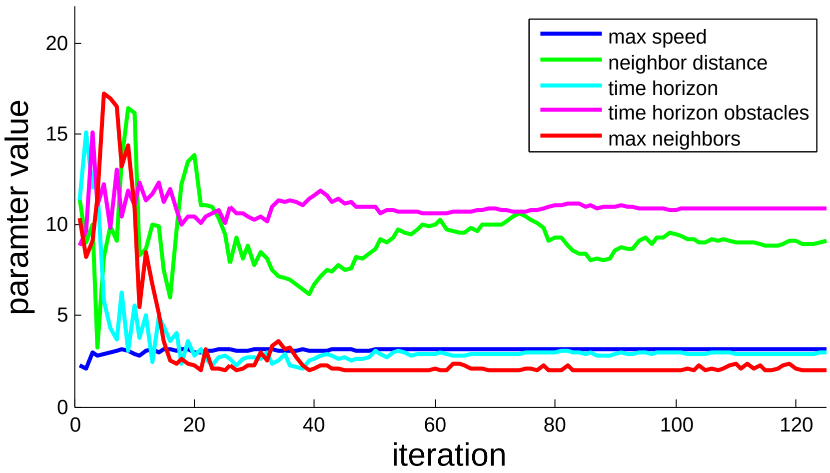
cles, max neighbours} are optimally fitted to an equally weighted combination of metrics over the test set $\mathcal{T}$. After 60 iterations the optimization converges to approximately 10% better objective value. A quick observation shows that the optimization has reduced the number of neighbours that the algorithm considers for each agent, max neighbours, from 10 to 2.

6.2 Objective Optimization

The default parameters for **PPR**, **ORCA** and **SF** cannot solve 39%, 56%, and 26% of the sampled scenarios respectively. Using the optimal parameter selection for **PPR**, the algorithm only fails in 9% of the scenarios, an improvement of 30% over the default settings. The significant optimization in time quality,



(a) Objective Function



(b) **ORCA** parameter values

Figure 6.2: Optimizing **ORCA** parameters to minimize the uniformly weighted combination of metrics over the test set $\mathcal{T}$. Each iteration is equal to 8 metric evaluations. As can be seen convergence occurs around 60 iterations.

$q^{\mathrm{t}}(A_{\mathbf{v}})$, for the **PPR** algorithm is impressive as well. **ORCA** does not show significant results over the metrics with the exception of q^{t}. On the other hand **SF** shows impressive improvement over most metrics, achieving the smallest failure rate d and the minimum energy expenditure, q^{e}. Table 10.4 lists the objective values for these findings and Figure 6.3 shows the realtive percent improvement.

To optimize failure rate, $d(A_{\mathbf{v}})$, **PPR** chooses very high values for predictive avoidance parameters and minimal values for speed thresholds, and trades off performance by selecting higher spatial querying distances.

When optimizing distance quality $q^{\mathrm{d}}(A_{\mathbf{v}})$ **PPR** changes different speed multipliers in an attempt to minimize any extra distance covered around corners. To improve computational efficiency e, **PPR** minimizes parameters that would trigger changes in its planned path, which would require an expensive path re-planning operation. To minimize failure rate and meet the time limit, **ORCA** raises its time horizon to increase the number of agents it considers in its velocity calculations, and increases its max speed so that agents cover as much distance as possible. For distance quality, $q^{\mathrm{d}}(A_{\mathbf{v}})$, **ORCA** reduces max speed just like **PPR**. In general, **SF** reduces acceleration parameters to minimum values for all quality metrics to prevent agents from overreacting.

6.2.1 Validation

We verify the statistical significance of the results shown in Figure 6.3 in two ways. First, we observe that for all three algorithms and for all the scenarios in the test set, $\mathcal{T}$, which are more than 5000, the optimization did not time out but converged to at least a local minimum. In the context of numerical optimization that is a sufficiently strong indication that the results are not random. Second, we perform a cross validation study on an equally large test set of similar, but previously unseen scenarios, $\mathcal{T}^v$. The results of this study can be found in Table 6.1. Comparing the values of the objectives for the default parameters of the algorithms, and for the optimized ones, we see that the optimized parameters on average perform better even on scenarios that were

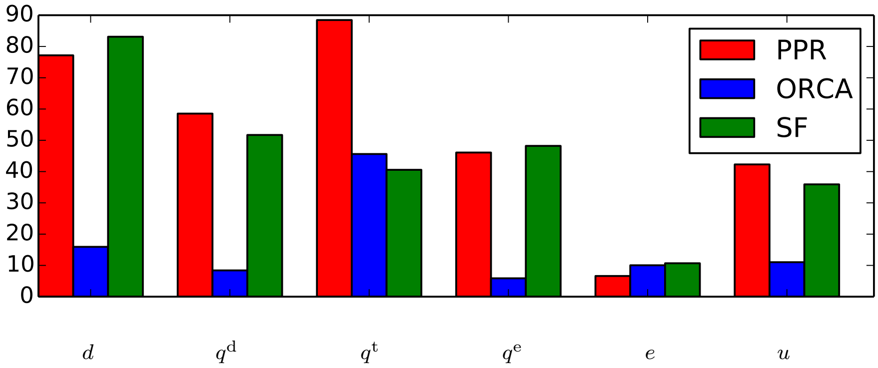


Figure 6.3: Relative percent improvement of failure rate d, distance quality q^{d}, time quality q^{t}, effort quality q^{e}, computational efficiency e, and a uniform combination of metrics u for the three steering algorithms.

not used during the optimization.

6.3 More Metric-Metric Analysis

It is interesting to investigate whether relationships exist between performance metrics. For example, does optimizing for distance quality, q^{d}, also optimize time quality q^{t}? To answer such questions, we compute the value of each metric obtained with parameter values that are optimized for the other metrics, Tables (6.2 - 6.4). We observe that the optimal parameters for distance quality, $q^{\mathrm{d}}(A_{\mathbf{v}})$, produce near-optimal results for failure rate, $d(A_{\mathbf{v}})$ for **PPR** and **ORCA**. However, the opposite does not hold true. Optimizing for failure rate does not yield optimal results for distance quality.

$A_{\mathbf{v}}$	$\mathbf{v}$	$d(A_{\mathbf{v}})$	$q^{\mathrm{d}}(A_{\mathbf{v}})$	$q^{\mathrm{t}}(A_{\mathbf{v}})$	$q^{\mathrm{e}}(A_{\mathbf{v}})$	$e_{\mathrm{s}}(A_{\mathbf{v}})$	$u(A_{\mathbf{v}})$
PPR	`DEF`	0.39	0.49	0.57	0.53	0.96	0.59
	`OPT`	0.10	0.22	0.07	0.30	0.91	0.34
ORCA	`DEF`	0.53	0.61	0.56	0.67	0.84	0.64
	`OPT`	0.51	0.57	0.29	0.62	0.82	0.58
SF	`DEF`	0.27	0.42	0.50	0.46	0.89	0.51
	`OPT`	0.05	0.20	0.30	0.23	0.81	0.33

Table 6.1: Validation of the Comparison of $q^{\mathrm{e}}(A_{\mathbf{v}})$, $d(A_{\mathbf{v}})$,$q^{\mathrm{d}}(A_{\mathbf{v}})$, $e_{\mathrm{s}}(A_{\mathbf{v}})$, $q^{\mathrm{t}}(A_{\mathbf{v}})$, and a uniform combination of all metrics for both steering algorithms using: (a) `DEF`: default parameter values and (b) `OPT`: optimal parameter values on second set of scenarios that were not used in training.

PPR	d	q^{d}	q^{t}	q^{e}	e	u
$d(A_{\mathbf{v}})$	0.47	0.46	0.49	0.48	0.65	0.48
$q^{\mathrm{d}}(A_{\mathbf{v}})$	0.59	0.56	0.58	0.57	0.71	0.57
$q^{\mathrm{t}}(A_{\mathbf{v}})$	0.39	0.52	0.30	0.63	0.43	0.32
$q^{\mathrm{e}}(A_{\mathbf{v}})$	0.73	0.66	0.71	0.63	0.79	0.71
$e(A_{\mathbf{v}})$	0.72	0.74	0.71	0.74	0.67	0.74
$u(A_{\mathbf{v}})$	0.59	0.59	0.56	0.61	0.65	0.55

Table 6.2: Comparison of failure rate $d(A_{\mathbf{v}})$, distance quality $q^{\mathrm{d}}(A_{\mathbf{v}})$, time quality $q^{\mathrm{t}}(A_{\mathbf{v}})$, effort quality $q^{\mathrm{e}}(A_{\mathbf{v}})$, computational efficiency $e_{\mathrm{s}}(A_{\mathbf{v}})$, and a uniform combination of all metrics $u(A_{\mathbf{v}})$ for the **PPR** steering algorithms. Each cell is the computation of the objective (row) using the parameters settings from optimizing for the objective (column). The optimal value for each objective is along the diagonal.

ORCA	d	q^{d}	q^{t}	q^{e}	e	u
$d(A_{\mathbf{v}})$	0.09	0.09	0.15	0.12	0.32	0.13
$q^{\mathrm{d}}(A_{\mathbf{v}})$	0.23	0.20	0.26	0.23	0.44	0.26
$q^{\mathrm{t}}(A_{\mathbf{v}})$	0.61	0.64	0.07	0.30	0.73	0.06
$q^{\mathrm{e}}(A_{\mathbf{v}})$	0.41	0.42	0.34	0.28	0.57	0.34
$e(A_{\mathbf{v}})$	0.98	0.96	0.97	0.94	0.89	0.90
$u(A_{\mathbf{v}})$	0.46	0.46	0.36	0.38	0.59	0.34

Table 6.3: Comparison of failure rate $d(A_{\mathbf{v}})$, distance quality $q^{\mathrm{d}}(A_{\mathbf{v}})$, time quality $q^{\mathrm{t}}(A_{\mathbf{v}})$, effort quality $q^{\mathrm{e}}(A_{\mathbf{v}})$, computational efficiency $e_{\mathrm{s}}(A_{\mathbf{v}})$, and a uniform combination of all metrics $u(A_{\mathbf{v}})$ for the **ORCA** steering algorithms. Each cell is the computation of the objective (row) using the parameters settings from optimizing for the objective (column).

SF	d	q^{d}	q^{t}	q^{e}	e	u
$d(A_{\mathbf{v}})$	0.04	0.05	0.05	0.05	1.00	0.05
$q^{\mathrm{d}}(A_{\mathbf{v}})$	0.20	0.20	0.20	0.20	1.00	0.20
$q^{\mathrm{t}}(A_{\mathbf{v}})$	0.30	0.28	0.29	0.28	1.00	0.29
$q^{\mathrm{e}}(A_{\mathbf{v}})$	0.24	0.23	0.24	0.23	1.00	0.23
$e(A_{\mathbf{v}})$	0.83	0.83	0.83	0.83	0.80	0.83
$u(A_{\mathbf{v}})$	0.32	0.32	0.32	0.32	0.96	0.32

Table 6.4: Comparison of failure rate $d(A_{\mathbf{v}})$, distance quality $q^{\mathrm{d}}(A_{\mathbf{v}})$, time quality $q^{\mathrm{t}}(A_{\mathbf{v}})$, effort quality $q^{\mathrm{e}}(A_{\mathbf{v}})$, computational efficiency $e_{\mathrm{s}}(A_{\mathbf{v}})$, and a uniform combination of all metrics $u(A_{\mathbf{v}})$ for the **SF** steering algorithms. Each cell is the computation of the objective (row) using the parameters settings from optimizing for the objective (column).

6.4 Summery

We study the effects of parameter fitting using the combined test sets, $\mathcal{T}$ and $\mathcal{T}^v$. Our goal is to identify whether parameter fitting has a significant effect and to understand the relation between algorithmic parameters and performance. For each of the three algorithms, **PPR**, **ORCA** and **SF**, we compute the optimal parameter values for each of the five metrics, failure rate $d(A_{\mathbf{v}})$, distance quality $q^{\mathrm{d}}(A_{\mathbf{v}})$, time quality $q^{\mathrm{t}}(A_{\mathbf{v}})$, PLE $q^{\mathrm{e}}(A_{\mathbf{v}})$, efficiency $e(A_{\mathbf{v}})$, as well as a uniform combination of these metrics, $u(A_{\mathbf{v}})$, over the entire combined set, $\mathcal{T}$. For comparison, we also compute the same metrics for all algorithms with their parameters set to default values. The results in Figure 6.3 show a strong increase in optimality for all metrics.

Chapter 7

Multi-Objective Optimization

Optimizing a steering algorithm's parameters across a large test set is computationally expensive. The computational complexity increases with the number of parameters and the cardinality of a test set. For example, it takes ~ 20 hours to optimize the 11 parameters of **SF** over the representative test set $\mathcal{T}$. In a weighted multi-objective optimization application, it is desirable to model the relationship between objectives and algorithm parameters. This avoids running an expensive optimization every time we wish to change the associated weights. This can be accomplished by computing the optimal parameters for a discrete set of weighted combinations that can then be interpolated. There are two problems with this approach. First, it can waste significant amounts of computation since each sample point is a result of an independent process that could be visiting the same points in the domain. Second and most important, it is not looking at relationships between the objectives but rather at their weighted combination. Both of these problems can be addressed by computing a *Pareto Optimal Front*. Pareto optimality is a very important concept in optimization which has sparingly been used in computer animation. Our method based on *Pareto Optimality* not only avoids unnecessary computation but also provides a more principled model of the optimal relationships between multiple objectives.

Multi-Objective optimization using a weighted or scalarized combination of the objectives still has many disadvantages. Small changes in the weights used can result in large changes in the parameters and small changes in the objective value. An uneven sampling of the trade off between objectives can occur. Requires forms of scaling of the objectives in order to prevent one objective from being over represented in the weighted sum. A user needs to provide weights to a function that could return unintuitive results, especially when the user wants to change the weights. If the *pareto-optimal front* is non-convex points that represent optimal trade-offs between objectives can be missed [Caramia and Dell'Olmo, 2008].

7.1 Pareto Optimal Front

Pareto Optimality (or Efficiency) refers to a situation where no objective can be improved further without worsening one of the other objectives. The set of points that are Pareto optimal constitute the *pareto-optimal front*, a hypersurface that captures the optimal relationships between the objectives. Computing this front is not trivial and is, in fact, an active area of research. Current state of the art techniques are primarily based on genetic algorithms. We have chosen to use the software DEAP [Fortin *et al.*, 2012] and the algorithm NSGA-II [Deb *et al.*, 2002] to estimate the *pareto-optimal front*.

7.1.1 Application

A standard evolutionary approach to solving a multi-objective optimization problem models the fitness of samples using a single objective function that is the weighted sum of multiple objectives, where the samples chosen in each iteration minimize the combined objective. In contrast, the goal of *pareto-optimal front* approximation is to maximize the hyper-volume constructed by the non-dominated samples (see Figure 7.1). A point dominates another if it is superior in all Pareto dimensions.

First, we optimize the **ORCA** steering algorithm for e and q^{e} over a bottleneck scenario. The process and resulting *pareto-optimal front* can be seen in Figure 7.2. Second, we optimize the **SF** algorithm for the same scenario and three metrics, e, q^{e} and $g(A_{\mathbf{v}},\mathcal{G}\ )$ (the result can be found in Figure 7.3(a)). The ground truth set $\mathcal{G}$, is a recording of people funnelling into a small bottleneck, very similar to the scenario used. We optimize for the same objectives with the **ORCA** steering algorithm and the resultant *pareto-optimal front* can be see in Figure 7.3(b). The pareto front is able to capture the non-linear relationships between contradictory objectives and efficiently encodes the tradeoffs between them. For example, optimizing q^{e} has an adverse effect on $g(A_{\mathbf{v}},\mathcal{G}\ )$, as shown in Figure 7.3(a and b).

The *pareto-optimal front* provides a principled model of the optimal relationships between the objectives. The number of dimensions is equal to the number of objectives, thus for two objectives the result is a 2D curve and for

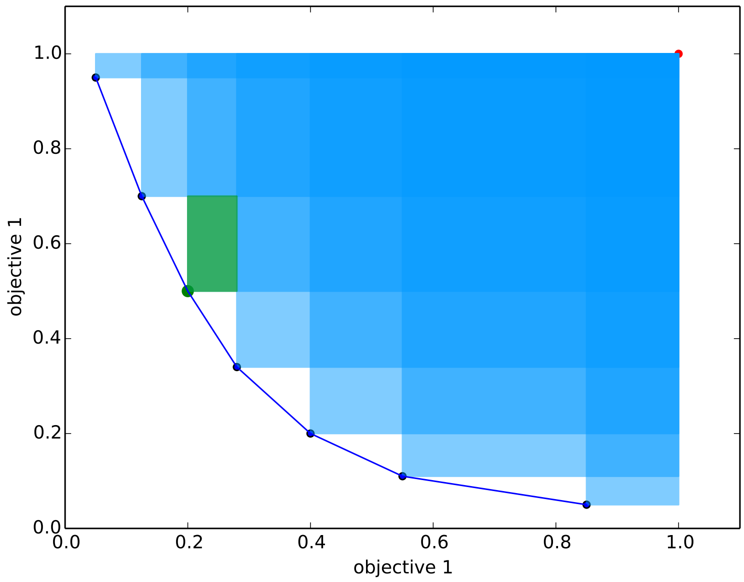


Figure 7.1: This figure shows the construction of the hyper-volume from the non-dominated points. Each of the points are considered more optimal than any point in the shaded region defined by the point. The addition of the green point increases the area of the hyper-volume by the green area.

three objectives a 3D surface. For most practical applications three objectives should be sufficient.

7.2 Pareto Optimal Front Interpolation

Having an estimate of the *pareto-optimal front* for a set of objectives provides us with the basis to estimate optimal parameters for the associated algorithm with arbitrary combinations of the objectives.

The first step in developing an interpolation model for arbitrary combinations of the objectives is to transform the *pareto-optimal front* from objective

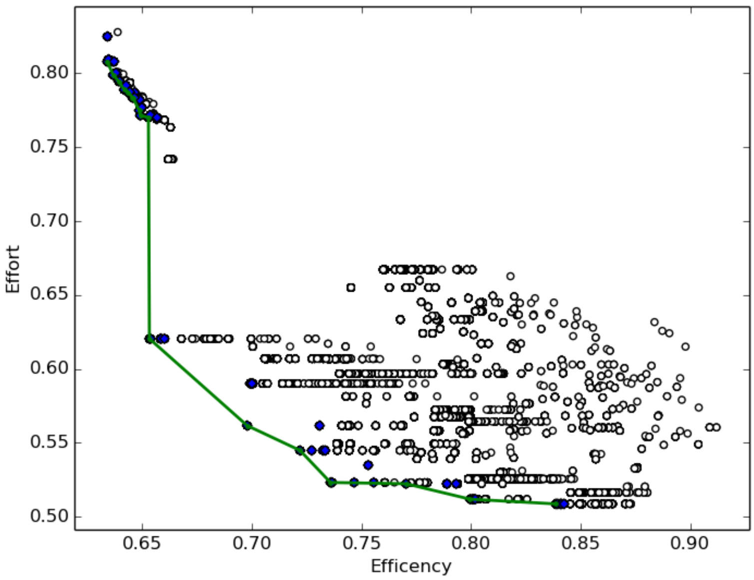
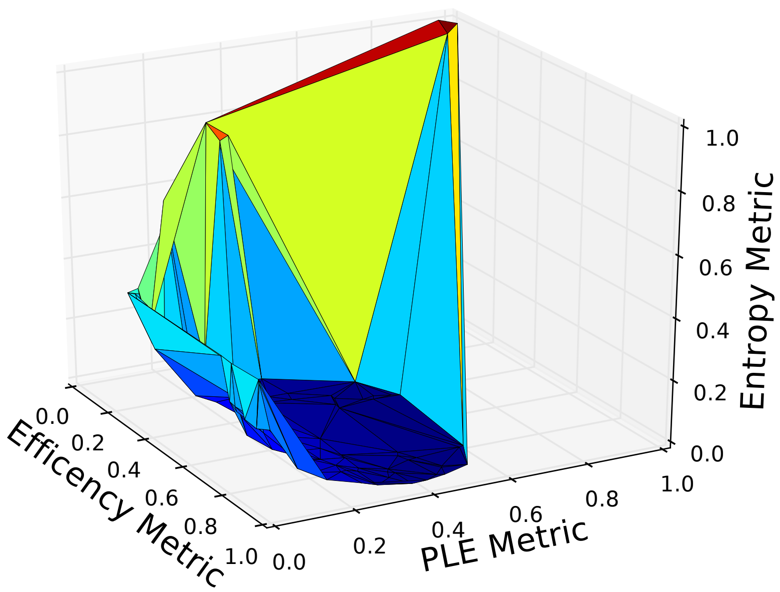


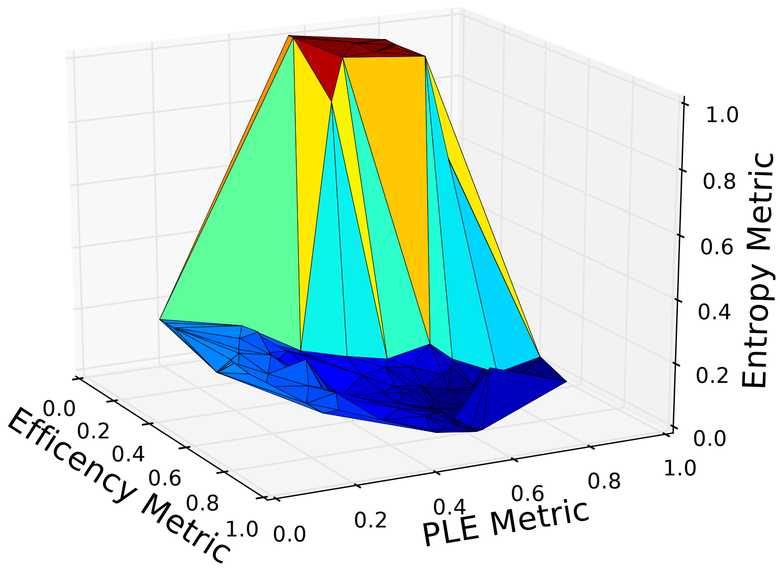
Figure 7.2: This Figure shows the final *pareto-optimal front* of non-dominated points (in green) for the **ORCA** steering algorithm over two objectives. The points in blue are the samples in the last generation and the circles are from previous generations.

space to weight space. For m objectives the *pareto-optimal front* contains a set of m-dimensional points, $\mathcal{P} = \{\mathbf{b}_p | \forall p = 1, ..., N\}$, including a set of points $\mathcal{P}_O = \{\mathbf{b}_p^O | \forall p = 1, ..., m\}$, that correspond to minimizing each objective while ignoring the others. These latter points have known coordinates in weight space that correspond to the standard unit vectors, and hold the minimum value in the associated dimension.

We transform the *pareto-optimal front* from the m-dimensional objective space, $[b_i]$, to the m-dimensional weight space, $[w_i]$, using the following steps: (a) we normalize the *pareto-optimal front* so that each dimension maps to $[0, 1]$,



(a)



(b)

Figure 7.3: Figures (a and b) show the final computed *pareto-optimal front* of three objectives for the **SF** and **ORCA** steering algorithms over a bottleneck scenario.

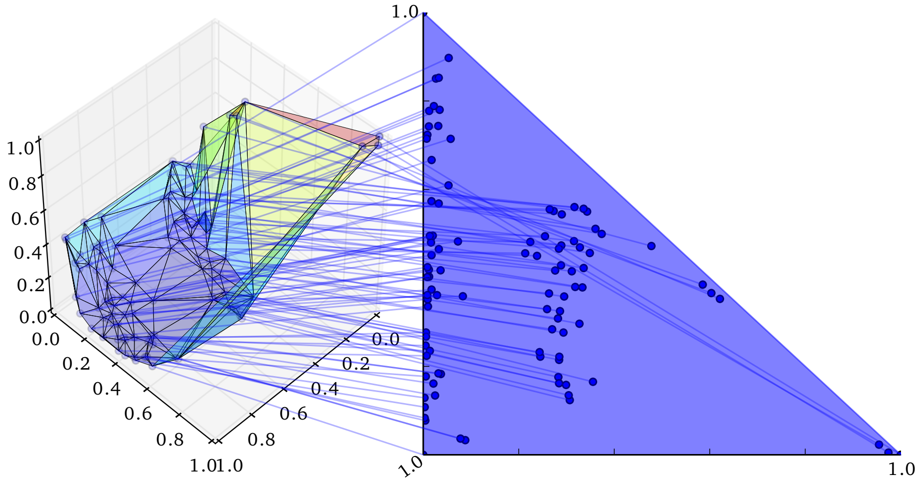


Figure 7.4: Projecting the 3D *pareto-optimal front* from Figure 7.3 (a) to a triangular normalized weight domain.

(b) we replace each point with its distances from the normalized points in $\mathcal{P}_O$, (c) we project the points, $\mathbf{b}'$, resulting from the previous stage onto the $\sum_i b'_i = 1$ plane and (d) we subtract them from 1. The transformed *pareto-optimal front* is now mapped onto a normalized simplex from which we can compute the relative weights of each original point as its barycentric coordinates, (Figure 7.4).

Having the *pareto-optimal front* in weight space, we can now use a standard multidimensional interpolation method such as Mardy quadratics or variants of Shepard's method. A common choice within the Mardy quadratics family of methods is radial basis function interpolation. For three objectives, the associated domain forms a triangle. In this case, given a new set of weights, we can use Delaunay triangulation to compute the three points that make up the bounding simplex whose associated parameters will be interpolated with a standard inverse distance approach.

Chapter 8

Additional Results And Examples

Chapter 6 demonstrates that it is both beneficial and revealing to fit the parameters of a steering algorithm to performance objectives over a large set of test cases. This section presents a series of experiments that demonstrate the potential applications of parameter fitting for more specific cases. We refer the reader to the accompanying video for a visual demonstration of the results and additional experiments.

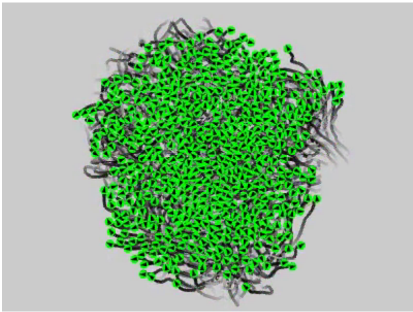
8.1 Single-Objective Results

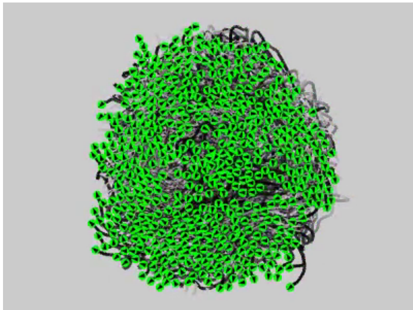
Circlular Benchmark. A popular and challenging scenario, often used to test the effectiveness of a steering algorithm, distributes the agents on a circular fashion with diametrically opposite goals. Such a configuration forces dense simultaneous interactions in the middle of the circle. Using a group of 500 agents, we compare the results of **ORCA** with the default and optimized parameter values that minimize time quality $q^{\mathrm{t}}(A_{\mathbf{v}})$. With the optimal parameters, **ORCA** takes 50% less time to complete the benchmark and exhibits a more organized emerging behaviour. Agents seem to form groups that follow a smooth curved trajectory, Figure 8.1 (top). For reference, the optimal parameter set in this case is: {max speed: 3.2, neighbour distance: 13.63, time horizon: 2.32, time horizon obstacles: 5.30, max neighbours: 7}.

Room Evacuation. Evacuation benchmarks are important for a range of application domains. In this benchmark, a group of 500 agents must exit a room. For this experiment, we use the social forces, **SF**, method with the default as well as optimized parameter values that minimize the effort quality metric, $q^{\mathrm{e}}(A_{\mathbf{v}})$. **SF** with optimal parameters spends 66% less energy on average per agent, exhibits tighter packing, and visibly reduces the turbulence of the crowd's behaviour, Figure 8.1 (bottom).

Office Evacuation. A more challenging evacuation scenario places 1000 agents in a complex, office-like ground floor. Optimizing **ORCA** for time quality, $q^{\mathrm{t}}(A_{\mathbf{v}})$, reduces the average time it takes to exit the building by almost 60%. In addition, it exhibits higher crowd density and higher throughput at the exits, as seen in Figure 8.2. Here we use ADAPT [Kapadia *et al.*, 2014] to render bipedal characters.

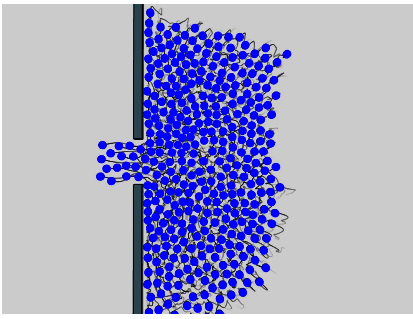
Optimizing for Ground Truth. There are a few methods that use recorded crowd motion to influence and direct virtual crowds. Here, we simply show that our methodology can also support this application. We optimize the behaviour of the three test algorithms to match real world data contained in the ground truth test set, $\mathcal{G}$, Section 4.1. Our experiments showed that, in most cases,

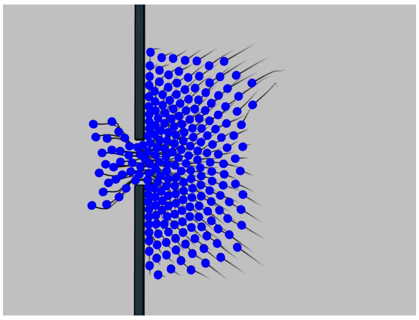




(a) Scenario I : default

(b) Scenario I: optimal time





(c) Scenario II: default

(d) Scenario II: optimal effort

Figure 8.1: Comparison of simulations using default [(a), (c)] and optimized [(b), (d)] parameters. Top: Agents are initially in a circle with anti-diametric goals. The **ORCA** algorithm, optimized to reduce time-to-completion, completes the task **twice** as fast as its default configuration and exhibits a less turbulent pattern. Bottom: The **SF** algorithm, optimized to minimize effort, requires a **third** of the energy spent by its default configuration, and produces a smoother, faster and tighter room evacuation.

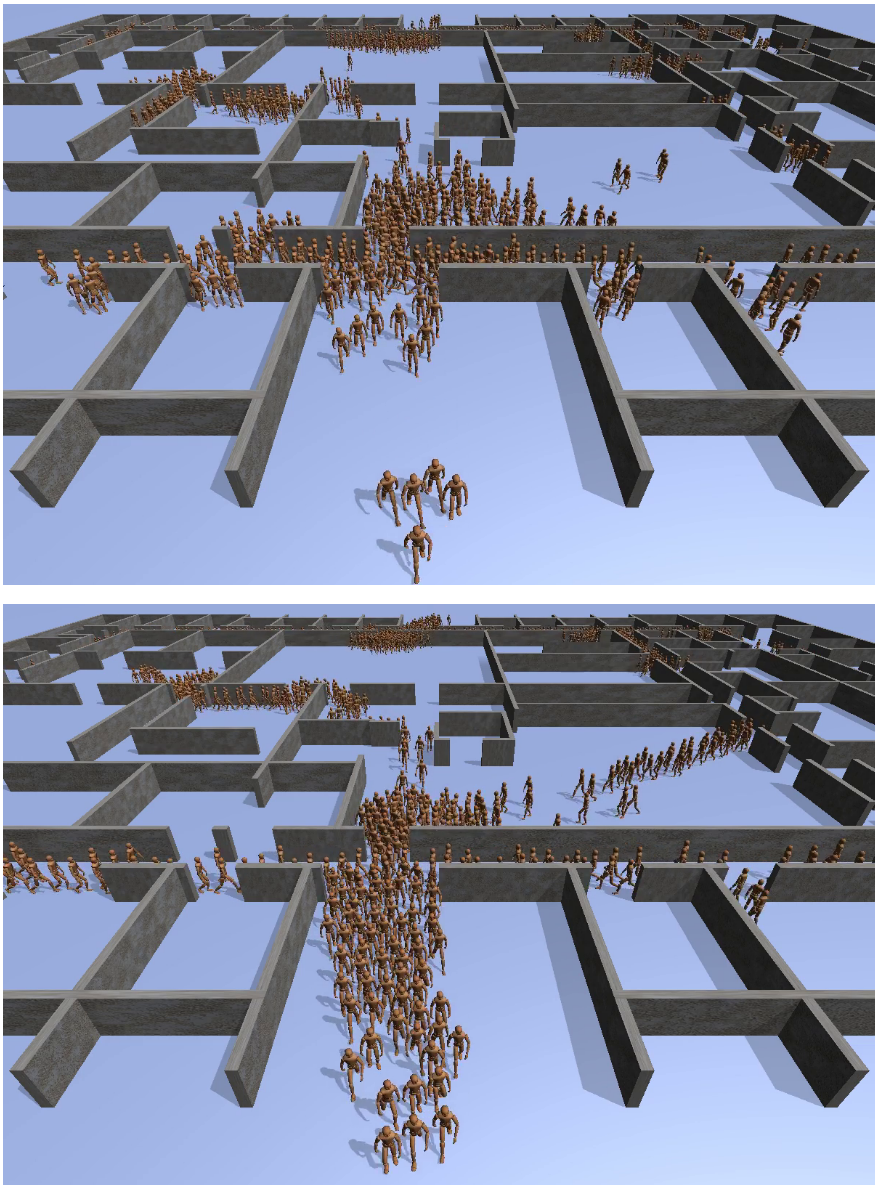


Figure 8.2: Office evacuation with **ORCA**. Simulation with parameters optimized for time quality (bottom) take half the time to complete as compared to the default parameters (top).

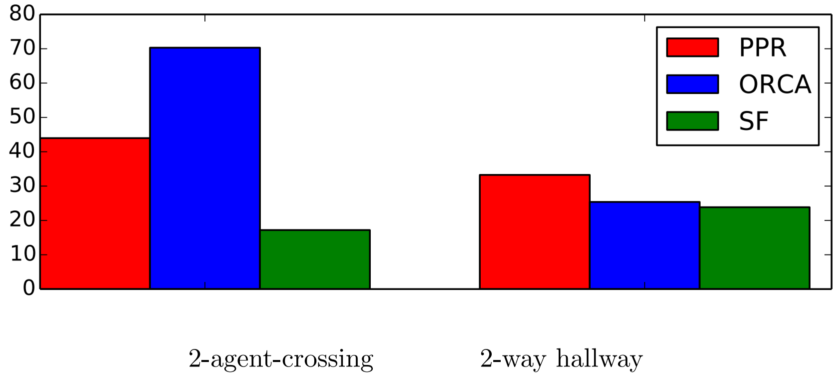


Figure 8.3: Relative percent improvement of entropy metric values after optimization on two different benchmarks.

the optimization was able to significantly alter the resulting steering behaviour and increase the similarity to the recorded data. Figure 8.3 reports the reduction in the entropy metric, g, (increase in similarity) as a result of parameter optimization for all three algorithms and two different benchmarks.

Dynamically Adapting Steering Parameters. Our methodology can create a large number of samples that relate parameter values to performance metrics. Figure 8.4 shows a snapshot from an interactive demo of a busy bi-directional hallway, that allows the user to switch dynamically between optimal parameter values that correspond to different objectives. The parameter settings used in this demo are fundamentaly the standard unit vectors described in Section 7.2.

8.2 Pareto Optimal Front Results

Interactive Parameter Blending. Using a precomputed *pareto-optimal front*, Chapter 7, we can automatically adapt an algorithm's parameters to provide optimal trade-offs for interactively defined combinations of the associated objectives. Figure 8.5 shows a snapshot of such blending between three objectives.

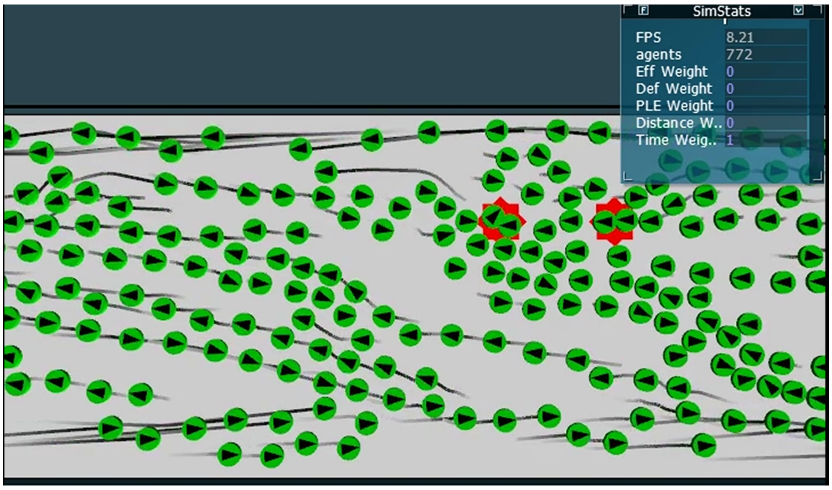


Figure 8.4: A prototype system for interactively setting the relative weights of the metrics in the objective. When the weights change, the algorithm's parameters are set automatically to the corresponding optimal values.

This process is best demonstrated in the accompanying video.

8.3 Implementation details

The primary factors affecting the computational performance of the optimization are the size of the test set, the number and range of parameters that are fitted, and the number of agents in the test cases. Although CMA-ES is an efficient optimization method, fitting a large number of parameters over a sizeable test set is computationally expensive. For reference, a 12 core, 2.4 GHz, 12 GB, computer (with hyper-threading), using 10 parallel threads, takes ~ 20 hours to optimize the **SF** algorithm over the representative test set $\mathcal{T}$. It takes ~ 3 days running 16 parallel threads to compute a *pareto-optimal front* with 3 objectives. Interactive blending the *pareto-optimal front* is done in realtime.

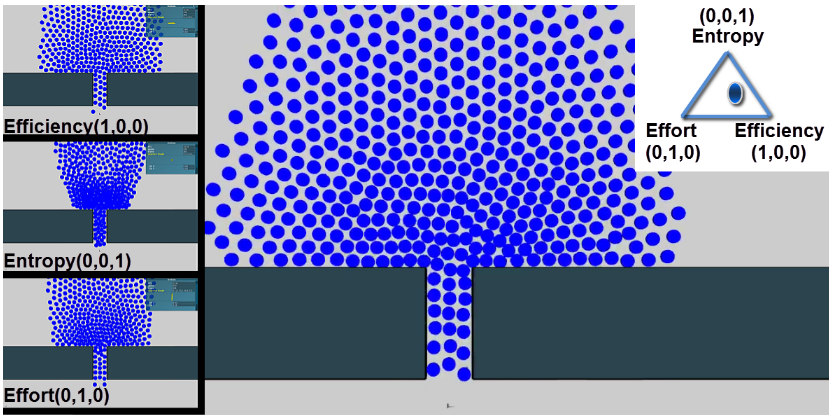


Figure 8.5: Blending interactively three objectives (Efficiency, Entropy, and Effort) using a pre-computed *pareto-optimal front*.

Chapter 9

Conclusion

We have presented a framework for optimizing the parameters of a steering algorithm for multiple objectives. Using cross-validation, we show that optimizing over a representative set of scenarios produces optimal parameters that generalize well to new test cases. We have also proposed a method to model trade-offs between the objectives using a *pareto-optimal front*. The *pareto-optimal front* essentially captures the optimal relationships between objectives. Although our approach can be applied to any number of objectives, three is a practical choice. Thus, we have demonstrated an interactive example that uses the computed *pareto-optimal front* to blend between three objectives.

Our study shows that parameter fitting not only can be used to improve the performance of an algorithm, but it can also serve as an analysis tool to produce a detailed view of an algorithm's range of behaviour relative to its internal parameters. This detailed view can be the basis of a thorough introspective analysis that allows both developers and end-users to gain insights on the performance and behaviour of an algorithm. Our framework and methodology are general. Most elements can be tailored to the needs of a particular application. For example, one can use different performance metrics, objectives, test sets, and optimization methods. The supplementary document provides the optimal parameter values of the three steering algorithms for the different objectives which AI developers and enthusiasts can directly use to improve the performance of their crowd simulations. The computational expense of optimizations, especially for large-scale crowds is one of the reasons why we are committed to sharing our results with the community.

9.1 Limitations

Optimization-based methods have certain well-known limitations. For example, it might not be easy or even possible for an optimization process to construct what is essentially a relationship between the parameters of a steering algorithm and global, or long-term, type of objectives. Furthermore, describing desired behaviours as combinations of objectives is not always straightforward and may require experimentation. Although estimating the *pareto-optimal front* is much more efficient and effective than naive domain sampling, it still requires significant offline computation.

9.2 Future Work

We plan to address heterogeneous crowds by using different parameters per agent or group of agents. We also plan to thoroughly investigate the sampling and complexity issues related to the estimation of the *pareto-optimal front*, focusing on objectives that are common in crowd simulation. Evaluating additional crowd simulation techniques with different agent representations and parameterizations is the subject of future work.

Chapter 10

Appendix

10.1 Parameter Settings

The Tables (10.1 - 10.3) list the default parameters for each algorithm as well as the optimal parameter settings for the first 5 objectives and the equally weighted combination of the first 5 objectives.

10.2 Optimization Values

Table 10.4 contains the default and optimized objective values for the three steering algorithms over the first 5 objectives and their combination. Similarly, Table 10.5 shows the default and optimized objective values for the three steering algorithms but specifically for the entropy metric for two scenarios.

Parameter Name	DEF	Min	Max	d	q^{d}	q^{t}	q^{e}	e	u
max speed	2.60	1	4	3.29	1	4.00	1.66	4	3.03
max force	14	8	22	14.45	15.15	22	18.76	15.62	19.50
max speed factor	1.70	0.60	4.70	3.38	1.11	4.70	0.60	3.77	3.45
faster speed factor	1.31	0.55	4.20	0.62	3.92	0.81	2.84	1.22	0.71
slightly faster speed factor	1.15	0.40	3.40	1.72	2.41	3.40	3.09	0.90	2.29
typical speed factor	1	0.50	1.50	0.53	0.50	1.50	1.04	0.50	1.50
slightly slower speed factor	0.77	0.15	1.20	0.22	0.19	1.20	0.70	0.59	0.84
slower speed factor	0.50	0.10	1	0.11	0.10	0.10	0.10	0.57	0.10
cornering turn rate	1.90	0.83	3.76	3.45	2.30	1.69	3.51	2.64	1.53
adjustment turn rate	0.16	0.03	1.54	0.03	0.58	0.29	0.13	0.29	0.37
faster avoidance turn rate	0.55	0.15	1.87	0.72	1.06	1.01	0.79	0.89	1.30
typical avoidance turn rate	0.26	0.08	0.75	0.66	0.62	0.75	0.59	0.62	0.71
braking rate	0.95	0.50	1.50	0.52	1.50	0.55	1.17	0.67	1.44
comfort zone	1.50	0.70	2.80	1.41	1.70	1.32	2.01	0.86	1.63
query radius	10	5	21	17.40	11.03	5	8.22	5	5
similar direction threshold	0.94	0.78	1.00	0.93	0.95	0.78	0.89	0.99	0.80
same direction threshold	0.99	0.89	1.00	0.90	0.89	0.91	0.92	0.91	0.93
oncoming prediction threshold	−0.95	−0.99	−0.78	−0.97	−0.81	−0.81	−0.92	−0.99	−0.92
oncoming reaction threshold	−0.95	−0.99	−0.78	−0.87	−0.85	−0.78	−0.94	−0.95	−0.87
wrong direction threshold	0.55	0.23	0.78	0.26	0.23	0.23	0.29	0.45	0.25
threat distance threshold	8	3	16.80	13.70	16.19	8.59	6.02	6.90	9.48
threat min time threshold	0.80	0.37	1.45	0.38	0.79	1.17	0.39	1.11	0.37
threat max time threshold	4	1.22	8.77	7.99	5.15	6.46	8.21	3.97	3.99
predictive anticipation factor	5	2.33	8.39	4.30	4.74	4.78	6.87	5.85	5.38
reactive anticipation factor	1.10	0.33	2.31	0.95	1.03	0.97	1.01	0.62	0.99
crowd influence factor	0.30	0.11	0.61	0.35	0.22	0.30	0.44	0.11	0.59
facing static object threshold	0.30	0.08	0.61	0.09	0.34	0.29	0.61	0.19	0.46
ordinary steering strength	0.05	0.00	0.20	0.02	0.00	0.00	0.08	0.11	0.02
oncoming threat avoidance strength	0.15	0.05	0.40	0.40	0.12	0.06	0.09	0.17	0.08
cross threat avoidance strength	0.90	0.73	1.00	0.76	0.91	0.95	0.74	0.90	0.95
max turning rate	0.10	0.02	0.23	0.10	0.10	0.15	0.13	0.10	0.10
feeling crowded threshold	3	1	8	2	2	1	4.06	1	5
scoot rate	0.40	0.17	0.78	0.78	0.60	0.78	0.78	0.72	0.71
reached target distance threshold	0.50	0.10	0.90	0.78	0.90	0.90	0.90	0.12	0.89
dynamic collision padding	0.20	0.02	0.43	0.43	0.24	0.17	0.20	0.16	0.19
furthest local target distance	20	10	50	34	22	39	15	10	10
next waypoint distance	50	30	70	62	39	64	38	32	44
max num waypoints	20	10	50	22	15	10	44	32	13

Table 10.1: Parameters for **PPR** algorithm with their default values, bounds, and optimal values obtained using multi-variate analysis for different objective functions.

Parameter Name	DEF	Min	Max	d	q^{d}	q^{t}	q^{e}	e	u
max speed	2	1	3.20	3.20	2.15	3.20	1.52	3.14	3.14
neighbor distance	15	2	22	17.39	13.37	14.75	12.08	8.18	8.99
time horizon	10	2	16	16	3.71	2	2.72	8.44	2.92
time horizon obstacles	7	2	16	12.30	16	9.60	11.81	2	10.92
max neighbors	10	2	22	8	11	2	15.03	2	2

Table 10.2: Parameters for **ORCA** algorithm with their default values, bounds, and optimal values obtained for each metric separately, and a uniform combination of the metrics.

Parameter Name	DEF	Min	Max	d	q^{d}	q^{t}	q^{e}	e	u
acceleration	0.50	0.05	2	0.05	0.05	0.05	0.05	1.90	0.05
personal space threshold	0.30	0.10	1	0.69	0.28	0.50	0.10	0.10	0.41
agent repulsion importance	0.30	0.05	1	0.05	0.05	0.05	0.11	0.66	0.38
query radius	4	1	10	1	10	9.44	10	2.08	3.28
body force	1500	500	5000	2431.40	2778.10	3832.20	500	3498.40	4858.80
agent body force	1500	500	5000	500	4677.80	1573.70	4027.40	3009.50	1073.20
sliding friction force	3000	1000	10 000	3281.10	1000	6795.70	10 000	8489.20	6091.30
agent b	0.08	0.01	5	0.09	0.08	0.09	0.11	3.81	0.13
agent a	25	1	100	46.25	48.21	58.27	53.24	52.00	53.37
wall b	0.08	0.01	5	0.15	0.10	0.18	0.08	5	0.09
wall a	25	1	100	100	67.15	55.05	61.65	98.20	60.87

Table 10.3: Parameters for **SF** algorithm with their default values, bounds, and optimal values obtained using multi-variate analysis for different objective functions.

$A_{\mathbf{v}}$	$\mathbf{v}$	$d(A_{\mathbf{v}})$	$q^{\mathrm{d}}(A_{\mathbf{v}})$	$q^{\mathrm{t}}(A_{\mathbf{v}})$	$q^{\mathrm{e}}(A_{\mathbf{v}})$	$e_{\mathrm{s}}(A_{\mathbf{v}})$	$u(A_{\mathbf{v}})$
PPR	`DEF`	0.39	0.49	0.56	0.53	0.96	0.58
	`OPT`	0.09	0.20	0.07	0.28	0.89	0.34
ORCA	`DEF`	0.56	0.61	0.56	0.67	0.75	0.62
	`OPT`	0.47	0.56	0.30	0.63	0.67	0.55
SF	`DEF`	0.26	0.41	0.50	0.45	0.87	0.50
	`OPT`	0.04	0.20	0.29	0.23	0.78	0.32

Table 10.4: Comparison of failure rate $d(A_{\mathbf{v}})$, distance quality $q^{\mathrm{d}}(A_{\mathbf{v}})$, time quality $q^{\mathrm{t}}(A_{\mathbf{v}})$, effort quality $q^{\mathrm{e}}(A_{\mathbf{v}})$, computational efficiency $e_{\mathrm{s}}(A_{\mathbf{v}})$, and a uniform combination of all metrics $u(A_{\mathbf{v}})$ for the three steering algorithms using: (a) `DEF`: default parameter values and (b) `OPT`: optimal parameter values.

$A_{\mathbf{v}}$	$\mathbf{v}$	$g(A_{\mathbf{v}}$,2-agent-crossing$)$	$g(A_{\mathbf{v}}$,two-way-hallway$)$
PPR	`DEF`	3.42	3.40
	`OPT`	1.92	2.27
ORCA	`DEF`	2.12	2.95
	`OPT`	0.63	2.20
SF	`DEF`	3.74	3.62
	`OPT`	3.10	2.76

Table 10.5: Comparison of entropy metric values before and after optimization to match real world data. `DEF`: default parameter values, `OPT`: optimal parameter values.

Bibliography

[Axel Buendia, 2002] Jerome Hoibian Axel Buendia. SpirOps: Scientific Research Labs in Artificial Intelligence, 2002.

[Berseth *et al.*, 2013] Glen Berseth, Mubbasir Kapadia, and Petros Faloutsos. Steerplex: Estimating scenario complexity for simulated crowds. In *Proceedings of Motion on Games*, MIG '13, pages 45:67–45:76, New York, NY, USA, 2013. ACM.

[Boatright *et al.*, 2013] Cory D. Boatright, Mubbasir Kapadia, Jennie M. Shapira, and Norman I. Badler. Context-sensitive data-driven crowd simulation. In *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, VRCAI '13, pages 51–56, New York, NY, USA, 2013. ACM.

[Bruckner and Moller, 2010] Stefan Bruckner and Torsten Moller. Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1468–1476, November 2010.

[Caramia and Dell'Olmo, 2008] M. Caramia and P. Dell'Olmo. *Multi-objective Management in Freight Logistics: Increasing Capacity, Service Level and*

Safety with Optimization Algorithms. Springer, 2008.

[Davidich and Koester, 2011] M. Davidich and G. Koester. Towards automatic and robust adjustment of human behavioral parameters in a pedestrian stream model to measured data. In Richard D. Peacock, Erica D. Kuligowski, and Jason D. Averill, editors, *Pedestrian and Evacuation Dynamics*, pages 537–546. Springer US, 2011.

[Deb *et al.*, 2002] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.

[Fortin *et al.*, 2012] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[Guy *et al.*, 2010] Stephen J. Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming Lin, and Dinesh Manocha. Pledestrians: a least-effort approach to crowd simulation. In *Proceedings of SCA*, pages 119–128. Eurographics Association, 2010.

[Guy *et al.*, 2012] Stephen J. Guy, Jur van den Berg, Wenxi Liu, Rynson Lau, Ming C. Lin, and Dinesh Manocha. A statistical similarity measure for aggregate crowd dynamics. *ACM Trans. on Graphics*, 31(6):11, 2012.

[Ha *et al.*, 2013] Sehoon Ha, Jim McCann, C. Karen Liu, and Jovan Popović. Physics Storyboards. *Computer Graphics Forum (The proceeding of Eurographics 2013)*, 2013.

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

[Hansen, 2011] Nikolaus Hansen. A CMA-ES for Mixed-Integer Nonlinear Optimization. Research Report RR-7751, INRIA, October 2011.

[Helbing *et al.*, 2000] Dirk Helbing, Illes Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[Helbing *et al.*, 2005] Dirk Helbing, Lubos Buzna, Anders Johansson, and Torsten Werner. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transp. Science*, 39(1):1–24, 2005.

[Henderson, 1971] L. F. Henderson. The statistics of crowd fluids. *Nature*, 229(5284):381–383, February 1971.

[Hoogendoorn, 2003] Serge P. Hoogendoorn. Pedestrian travel behavior modeling. In *In 10th International Conference on Travel Behavior Research, Lucerne*, pages 507–535, 2003.

[Huerre *et al.*, 2010] Stephanie Huerre, Jehee Lee, Ming Lin, and Carol O'Sullivan. Simulating believable crowd and group behaviors. In *ACM SIGGRAPH ASIA 2010 Courses*, SA '10, pages 13:1–13:92, New York, NY, USA, 2010. ACM.

[Johansson *et al.*, 2007] A Johansson, D Helbing, and P Shukla. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. *Advances in Complex Systems*, 10(supp02):271–288, 2007.

[Ju *et al.*, 2010] Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. Morphable crowds. In *ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA '10, pages 140:1–140:10, New York, NY, USA, 2010. ACM.

[Kapadia *et al.*, 2009] Mubbasir Kapadia, Shawn Singh, William Hewlett, and Petros Faloutsos. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 215–223, New York, NY, USA, 2009. ACM.

[Kapadia *et al.*, 2011a] Mubbasir Kapadia, Matt Wang, Shawn Singh, Glenn Reinman, and Petros Faloutsos. Scenario space: characterizing coverage, quality, and failure of steering algorithms. In *Proceedings of SCA*, SCA '11, pages 53–62. ACM, 2011.

[Kapadia *et al.*, 2011b] Mubbasir Kapadia, Matthew Wang, Shawn Singh, Glenn Reinman, and Petros Faloutsos. Scenario space: Characterizing coverage, quality, and failure of steering algorithms. In *Proceedings of SCA*, SCA '11, New York, NY, USA, 2011. ACM.

[Kapadia *et al.*, 2012] Mubbasir Kapadia, Shawn Singh, William Hewlett, Glenn Reinman, and Petros Faloutsos. Parallelized egocentric fields for autonomous navigation. *The Visual Computer*, 28(12):1209–1227, 2012.

[Kapadia *et al.*, 2014] Mubbasir Kapadia, Nathan Marshak, and Norman I. Badler. Adapt: The agent development and prototyping testbed. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints):1, 2014.

[Kulpa *et al.*, 2011] Richard Kulpa, Anne-Hélène Olivierxs, Jan Ondřej, and Julien Pettré. Imperceptible relaxation of collision avoidance constraints in virtual crowds. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 138:1–138:10, New York, NY, USA, 2011. ACM.

[Lamarche and Donikian, 2004] F. Lamarche and S. Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. In *Computer Graphics Forum 23.*, 2004.

[Lee *et al.*, 2007] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of SCA*, pages 109–118. Eurographics Association, 2007.

[Lemercier *et al.*, 2012] Samuel Lemercier, A. Jelic, Richard Kulpa, Jiale Hua, Jérôme Fehrenbach, Pierre Degond, Cécile Appert-Rolland, Stéphane Donikian, and Julien Pettré. Realistic following behaviors for crowd simulation. *Comput. Graph. Forum*, 31(2):489–498, 2012.

[Lerner *et al.*, 2007] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. *Computer Graphics Forum*, 26(3):655–664, September 2007.

[Lerner *et al.*, 2010] Alon Lerner, Yiorgos Chrysanthou, Ariel Shamir, and Daniel Cohen-Or. Context-dependent crowd evaluation. *Comput. Graph. Forum*, 29(7):2197–2206, 2010.

[Lovas, 1994] G.C. Lovas. Modeling and simulation of pedestrian traffic flow. In *Transportation Research Record*, pages 429–443, 1994.

[McDonnell *et al.*, 2008] Rachel McDonnell, Michéal Larkin, Simon Dobbyn, Steven Collins, and Carol O'Sullivan. Clone attack! perception of crowd variety. *ACM Trans. Graph.*, 27(3):26:1–26:8, 2008.

[Milazzo *et al.*, 1998] J. Milazzo, N. Rouphail, J. Hummer, and D. Allen. The effect of pedestrians on the capacity of signalized intersections. In *Transportation Research Record*, pages 37–46, 1998.

[Mononen, 2009] Mikko Mononen. Recast: Navigation-mesh Construction Toolset for Games, 2009.

[Musse *et al.*, 2012] Soraia R. Musse, Vinicius J. Cassol, and Cludio R. Jung. Towards a quantitative approach for comparing crowds. *Computer Animation and Virtual Worlds*, 23(1):49–57, 2012.

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

[Ondřej *et al.*, 2010] Jan Ondřej, Julien Pettré, Anne-Hélène Olivier, and Stéphane Donikian. A synthetic-vision based steering approach for crowd simulation. *ACM Trans. Graph.*, 29(4):123:1–123:9, July 2010.

[Paris *et al.*, 2007] Sébastien Paris, Julien Pettré, and Stéphane Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. In *EUROGRAPHICS 2007*, volume 26, pages 665–674, 2007.

[Pelechano *et al.*, 2007] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of SCA*, pages 99–108. Eurographics Association, 2007.

[Pelechano *et al.*, 2008] Nuria Pelechano, Jan M. Allbeck, and Norman I. Badler. *Virtual Crowds: Methods, Simulation, and Control*. Synthesis Lectures on Computer Graphics and Animation. Morgan & Claypool Publishers, 2008.

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

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

[Pinter, 2001] Marco Pinter. Toward more realistic pathfinding. *Gamasutra.com*, 2001.

[Regelous,] Stephen Regelous. Massive. `www.massivesoftware.com`.

[Reynolds, 1987] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of ACM SIGGRAPH*, pages 25–34. ACM, 1987.

[Reynolds, 1999] C W Reynolds. Steering behaviors for autonomous characters. *Game Developers Conference*, 1999(9602):763–782, 1999.

[Seyfried *et al.*, 2010] Armin Seyfried, Maik Boltes, Jens Khler, Wolfram Klingsch, Andrea Portz, Tobias Rupprecht, Andreas Schadschneider, Bernhard Steffen, and Andreas Winkens. Enhanced empirical data for the fundamental diagram and the flow through bottlenecks. In Wolfram W. F. Klingsch, Christian Rogsch, Andreas Schadschneider, and Michael Schreckenberg, editors, *Pedestrian and Evacuation Dynamics 2008*, pages 145–156. Springer Berlin Heidelberg, 2010.

[Singh *et al.*, 2009a] By Shawn Singh, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinman. Steerbench : a benchmark suite for evaluating steering behaviors. *Computer Animation And Virtual Worlds*, 20(February):533–548, 2009.

[Singh *et al.*, 2009b] Shawn Singh, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinman. An open framework for developing, evaluating, and sharing steering algorithms. In *Proceedings of MIG*, pages 158–169. Springer-Verlag, 2009.

[Singh *et al.*, 2011] Shawn Singh, Mubbasir Kapadia, Billy Hewlett, Glenn Reinman, and Petros Faloutsos. A modular framework for adaptive agent-based steering. In *Proceedings of I3D*, I3D '11, pages 141–150, New York, NY, USA, 2011. ACM.

[Sud *et al.*, 2007] Avneesh Sud, Russell Gayle, Erik Andersen, Stephen Guy, Ming Lin, and Dinesh Manocha. Real-time navigation of independent agents using adaptive roadmaps. In *Proceedings of VRST*, pages 99–106. ACM, 2007.

[Thalmann and Musse, 2013] Daniel Thalmann and Soraia Raupp Musse. *Crowd Simulation, Second Edition.* Springer, 2013.

[Treuille *et al.*, 2006] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168, 2006.

[van den Berg *et al.*, 2011] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer Berlin Heidelberg, 2011.

[Wang *et al.*, 2010] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 29(4):73:1–73:8, July 2010.

[Whittle, 2007] M. Whittle. *Gait analysis: an introduction.* Butterworth-Heinemann, 2007.

[Wolinski *et al.*, 2014] David Wolinski, Stephen Guy, Anne-Helene Olivier, Ming Lin, Dinesh Manocha, and Julien Pettré. Parameter estimation and comparative evaluation of crowd simulations. In *Eurographics*, 2014.