

University of Oxford

Department of Engineering Science



Learning Equilibria in Energy Markets

Luke Rickard

Worcester College

May 19, 2021

Supervised by:

Prof. Kostas Margellos

Dr. Jan-Peter Calliess

Abstract

We consider two multi-agent non-cooperative games, involving agents who are affected by uncertainty in their objective functions, firstly for the case of agents affected by a common tax penalty, then for agents with a different uncertain term in their objective functions. We are motivated in particular by applications in distributed battery charging of Plug-in Electric Vehicles. These uncertain terms then can be seen to encode volatility in the underlying energy price. We take a data driven scenario-based approach to uncertainty, and will incorporate a sampling-and-discarding approach to trade feasibility to performance and further improve agents' cost functions whilst leading to a reduction in robustness to new samples. We provide results from playing these games to compare experimental results with those predicted in theory. A number of different algorithms for playing these games are introduced, and we provide a comparison of their effectiveness in this game as measured by their convergence rate, their efficiency in the sense of quantifying the difference of the agents' objective functions from the social welfare optimum, and the potential improvements by adopting a sampling-and-discarding approach. The contributions of this report are thus twofold, to introduce the sampling-and-discarding approach in the context of electric vehicle charging and to provide an empirical comparison of these different algorithms in order to aid selection in future applications.

Contents

1	Introduction	1
2	Problem Statement	4
2.1	Problem set-up and main variables	4
2.2	Constraints	5
2.3	Objective Functions	5
3	Game Formulation	7
3.1	Distributed Setup	7
3.2	Solution Concepts	8
3.3	Proximal Based Algorithm	9
3.4	Gradient Descent Algorithm	11
3.5	Subgradient Descent Algorithm	14
3.6	Wardrop Equilibrium Seeking Algorithm	15
4	Scenario Discarding	17
4.1	Theoretical Bounds on the Violation Probability	17
4.2	Compression Set Computation	19
4.3	Scenario Discarding Algorithms	21
4.4	Violation Rate After Discarding Samples	22
5	Numerical Results	24
5.1	General Setup	24
5.2	Results with a Common Penalty Term	24
5.2.1	Algorithm Comparison	25
5.2.2	Scenario Discarding	28
5.2.3	Alternative Cost Functions	32

5.2.4	Incremental Scenario Addition	34
5.3	Results for Non-homogenous Objectives	35
5.3.1	Algorithm Comparison	35
5.3.2	Scenario Discarding	38
5.3.3	Alternative Cost Functions	41
5.3.4	Incremental Scenario Addition	42
6	Conclusion	44
6.1	Summary	44
6.2	Future Work	46
	Bibliography	48
A	Risk Assessment	51

Chapter 1

Introduction

Energy markets in which electric vehicles can act as flexible consumers, charging selectively to spread energy requirements more uniformly across a day (and hence reducing their own cost), are a particularly interesting application of game theory with significant attention in the control systems community [1–3]. There are a number of different algorithms coming from game theory [1, 4, 5]. Of particular interest are the Nash and Wardrop equilibrium concepts since they define how selfish agents may behave in games of this type (see [6, 7] and [8] respectively). The latter has been applied primarily to traffic congestion problems but recent work in [5] has applied the Wardrop equilibrium solution concept to non-cooperative games. The Nash equilibrium concept is hugely useful in non-cooperative games since it represents a set of strategies where agents are unable to lower their cost by unilaterally changing their strategy, thus selfish agents do not have any reason to change their strategy and no further changes in strategy occur. The Wardrop equilibrium considers the scenario where agents' contributions to the average strategy become negligible (i.e. for large population sizes).

Methods for dealing with uncertainty in these games include the use of stochastic or worst-case approaches. Two possible considerations for the stochastic approach involve chance-constrained (or risk-averse) games [9–11] and expected-payoff criteria [12–16]. The results in these papers typically require some underlying assumptions on the probability distribution of the uncertain set. The alternative worst-case approach is generally tackled through robust control theory [17–19], however, these results similarly rely on assumptions about the geometry of the uncertainty set [11, 20].

Further work has extended to taking data driven approaches where uncertainty is represented by a finite set of scenarios drawn by some prediction model [4]. Using this approach it is possible to compute equilibria using a probably approximately correct (PAC) learning framework [21–23], employing the scenario approach [24, 25]. Separate work in [26] has looked into taking a sampling-and-discarding

approach to the sample counterpart to a chance-constrained optimisation problem, removing samples from a finite set of sampled constraints.

In this report, we will be investigating non-cooperative games with a data driven approach to uncertainty, taking a sampling-and-discarding approach in order to trade feasibility for performance and improve agents' costs. We introduce a number of algorithms which are available to play these games, and which could be extended to other non-cooperative games. A comparison between these different algorithms is made, considering their convergence rate, computational complexity, their efficiency in the sense of quantifying the difference of the agents' objective functions from the social welfare optimum, and the potential improvements by adopting a sampling-and-discarding approach. We will generally not provide detailed proofs for the results in this paper, but will note where proofs are available and which proofs may be useful as a focus of future research.

In Chapter 2, we formally define the electric vehicle charging problem and provide an optimisation based framework to solve this problem in a centralised fashion. Chapter 3 then moves on to redefine this problem as a non-cooperative game, defining the solution concepts we are interested in, and outlining the structure of the various algorithms. In Chapter 4, we approach this problem under a data driven lens using the so called scenario approach, and in particular we introduce sampling-and-discarding considerations that allow offering probabilistic feasibility certificates for the computed equilibria, while allowing for an improvement in the associated cost. Finally, Chapter 5 contains a detailed comparative study discussing the relative merits of different algorithmic schemes for equilibrium computation. Chapter 6 then summarises these results, providing the main comparison of the algorithms and discussing potential areas for future work. Table 1.1 summarise the main algorithms and discarding schemes we will look at.

Table 1.1: Main Algorithms

	Summary
Algorithm 1 (Page 10): Proximal Based	Best response approach on each iteration in an augmented game.
Algorithm 2 (Page 12): Gradient Descent	Performs projected gradient descent step using an augmented (differentiable) objective function on each iteration.
Algorithm 3 (Page 15): Subgradient Descent	Finds a valid subgradient of the objective function and steps in this direction at each iteration.
Algorithm 4 (Page 16): Wardrop Seeking	Best response approach but assumes the average strategy to be fixed, thus tending towards a Wardrop equilibrium of the game.
Algorithm 6 (Page 21): Discarding of Worst Scenario	Identifies and removes the scenario resulting in the largest cost increase.
Algorithm 7 (Page 22): Discarding of All Active Scenarios	Finds all scenarios which currently have an impact on cost and removes these.

Chapter 2

Problem Statement

In this Chapter, we will focus on the problem of electric vehicle charging and outline the set-up. In doing so, we will construct this as a centralised optimisation problem where a single coordinating authority determines the optimal charging schedule for all agents. This formulation will act as a benchmark for the algorithms of the next Chapter. We will then extend this to the distributed gaming set-up in Chapter 3.

2.1 Problem set-up and main variables

We first begin by defining the variables relevant to our problem. We will consider a population of N Plug-in Electric Vehicles (PEVs) defined by the set $\mathcal{N} = \{1, \dots, N\}$, hereafter referred to as *agents* as well, with the aim of fully charging the population over some time period of n time steps (which will usually be 24 hours). The vehicle charging game can be described as follows. The action $x_i \in \mathbb{R}^n$ of each agent $i \in \mathcal{N}$ is called a schedule and $x_{i,t}$ is the charging rate of agent i at time $t \in \mathcal{T} := \{1, \dots, n\}$. The collection of all agents' schedules can then be defined as $x = (x_i)_{i \in \mathcal{N}}$. In addition, we are interested in the aggregate of all agents' schedules which we define as $\sigma(x) = \sum_{i \in \mathcal{N}} x_i$. For any given agent $i \in \mathcal{N}$ we can also denote the collection of the schedules of all other agents (excluding agent i) as $x_{-i} = (x_j)_{j \in \mathcal{N} \setminus \{i\}}$.

Uncertainty in our game comes in the form of volatility in the energy price at any given timestep, one possible cause of this volatility could come in the form of renewables in electricity generation. In order to encode uncertainty in the cost functions defined in the sequel we make use of a finite set of independent and identically distributed (i.i.d.) scenarios of some uncertain vector $\theta \in \Theta$. This set can be defined as $(\theta_1, \dots, \theta_M) \in \Theta^M$ for some $M \in \mathbb{N}$, henceforth referred to as the M -multisample. We denote by \mathbb{P} the probability measure according to which $\theta \in \Theta$ is distributed. We will see later that

in both objective functions we look at the pointwise maximum over functions parameterised by these different samples. This encodes our interest in the worst-case scenario, and is useful in allowing for some theoretical proofs later.

2.2 Constraints

In order to formulate our problem we must define constraints on agents' schedules that reflect the rules of the charging application. We will denote by $\mathcal{X}_i \subset \mathbb{R}^n$ the set of feasible actions, given by the constraints on agent i and by $\mathcal{X} = \mathcal{X}_1 \times \dots \times \mathcal{X}_N \subset \mathbb{R}^{nN}$ the set for the collection of all agents schedules x . Then any valid solution must satisfy $x \in \mathcal{X}$.

Each agent's feasible set will consist of a few components. Firstly, we define an interval for each element of the schedule vectors $x_{i,t} \in [0, M_{i,t}]$, for all $i \in \mathcal{N}, t \in \mathcal{T}$. This ensures that no agent can supply to the grid (encoded by the lower bound of 0) and that each agent has some upper bound $M_{i,t}$ for each time t . This upper bound generally encodes physical bounds on the charging rate, and we can then also allow agents to self define time instances throughout the day when they will not charge by setting $M_{i,t} = 0$. In order to ensure agents are fully charged at the end of the time period we also use the constraint $\sum_{t \in \mathcal{T}} b_i x_{i,t} = \gamma_i$ where $b_i \in (0, 1]$ is a constant to represent any given agent's battery efficiency and γ_i is an agent's battery capacity.

2.3 Objective Functions

Finally, we define two similar objective functions, accepting different interpretations, which we are interested in studying. The first has been used previously in the literature [4] and includes some theoretical guarantees in the distributed case, whilst the second can be used to more accurately reflect reality but with the loss of any current theoretical proofs. Both objective functions will feature a deterministic cost and an uncertain cost, it is this uncertain cost that differs in the two formulations. In both cases we take a pointwise maximum over the samples, motivated by our interest in the worst-case scenario.

In the first instance the uncertainty is introduced in the form of a common penalty term which is shared across all agents. This could arise due to a centrally managed fleet receiving the same "tax penalty" or agents participating in a common aggregation plan. In this case, we can define the global

cost as

$$\begin{aligned}
J(x) &= \sum_{i \in \mathcal{N}} \left(x_i^\top (A_0 \sigma(x) + b_0) + \max_{m \in \{1, \dots, M\}} \frac{\sigma(x)^\top}{N} (A_m \sigma(x) + b_m) \right) \\
&= \sum_{i \in \mathcal{N}} \left(f_i(x_i, x_{-i}) + \max_{m \in \{1, \dots, M\}} g(x_i, x_{-i}, \theta_m) \right).
\end{aligned} \tag{2.1}$$

In the second case the uncertainty encodes price volatility and is directly added on to the deterministic cost, resulting in a heterogeneous cost penalty for the agents. This gives us a more general form for agents who are not centrally managed in any way. This cost is defined as

$$\begin{aligned}
\hat{J}(x) &= \sum_{i \in \mathcal{N}} \max_{m \in \{1, \dots, M\}} \left(x_i^\top ([A_0 + A_m] \sigma(x) + [b_0 + b_m]) \right) \\
&= \sum_{i \in \mathcal{N}} \left(x_i^\top (A_0 \sigma(x) + b_0) + \max_{m \in \{1, \dots, M\}} x_i^\top (A_m \sigma(x) + b_m) \right) \\
&= \sum_{i \in \mathcal{N}} \left(f_i(x_i, x_{-i}) + \max_{m \in \{1, \dots, M\}} g_i(x_i, x_{-i}, \theta_m) \right).
\end{aligned} \tag{2.2}$$

In both cases we have $\theta_m = (A_m, b_m)$ for $m = 1, \dots, M$ defining the uncertainty, and generally for $m = 0, 1, \dots, M$ we have diagonal matrices $A_m \in \mathbb{R}^{n \times n}$ and vectors $b_m \in \mathbb{R}^n$. A_0 is reformulated from historical usage data available at [27], $b_0 = 0$, A_m is sampled from a Log-Normal distribution, and b_m from a uniform distribution. These represent a difference from a baseline electricity cost (A_0) and could be sampled from historical data for a more realistic game.

We have defined the probability distribution \mathbb{P} in such a way that A_m are all positive definite (since a Log-Normal distribution has support $(0, +\infty)$). Since our baseline cost must be defined such that increased charging leads to an increased cost, A_0 must also be at least positive semidefinite, and so the overall objective function will be strictly convex. This could be further relaxed as the only necessary requirements on A_0 and A_m is that their sum is positive definite. Thus, we find that both objective functions are strictly convex, this restriction is required for theoretical guarantees and makes logical sense in requiring that increased charging during any given time slot leads to an increased cost.

The full problems we are optimising are then $\min_{x \in \mathcal{X}} J(x)$ and $\min_{x \in \mathcal{X}} \hat{J}(x)$ and we will denote by J^* and \hat{J}^* the minimum value for both of these problems respectively, these will later be referred to as the *social optimum* or *centralised solution*. We note that the minimisers that achieve these minimums may not necessarily be unique.

This concludes a full definition for the problem we wish to solve, in the following chapter we will proceed look at solution concepts and the formulation of a game to solve this problem.

Chapter 3

Game Formulation

Following the formulation of our problem in the case of a centrally managed set of PEVs we now proceed to a more interesting case where agents perform optimisation locally, participating in a non-cooperative game to each try and minimise their own local cost. This non-cooperative game would be played as a repeated planning game, once convergence is reached the schedules planned could then be actioned upon in the real world. The decomposition of this problem into a repeated planning game allows for agents to preserve privacy regarding their exact constraints, significantly reduces the computation required by the central coordinator, and allows agents to act selfishly. In the format of our charging problem this can be seen as each vehicle individually optimising a much simpler problem than the one presented in Chapter 2, sending only its planned schedule and receiving only the aggregate schedule and energy prices. The central coordinator then simply collects and aggregates agents' schedules, and broadcasts these along with the M-multisample and base energy price.

3.1 Distributed Setup

Firstly, we will consider each agent's local objective function, in this case, we can simply take the relevant term in the summation from equations (2.1) and (2.2).

$$\begin{aligned} J_i(x_i, x_{-i}) &= x_i^\top (A_0 \sigma(x_i, x_{-i}) + b_0) + \max_{m \in \{1, \dots, M\}} \frac{\sigma(x_i, x_{-i})^\top}{N} (A_m \sigma(x_i, x_{-i}) + b_m) \\ &= f_i(x_i, x_{-i}) + \max_{m=1, \dots, M} g(x_i, x_{-i}, \theta_m), \end{aligned} \tag{3.1}$$

$$\begin{aligned}
\hat{J}_i(x_i, x_{-i}) &= \max_{m \in \{1, \dots, M\}} \left(x_i^\top ([A_0 + A_m] \sigma(x_i, x_{-i}) + [b_0 + b_m]) \right) \\
&= f_i(x_i, x_{-i}) + \max_{m=1, \dots, M} g_i(x_i, x_{-i}, \theta_m).
\end{aligned} \tag{3.2}$$

Such that $\sum_{i \in \mathcal{N}} J_i = J$ (i.e. if each agent pays cost J_i the total cost is still $J(x)$). In the sequel we refer to the local cost of agent i as J_i and \hat{J}_i for the 2 different objectives, respectively.

We then can describe the non-cooperative games as the tuples $\mathcal{G} = \langle \mathcal{N}, (\mathcal{X}_i)_{i \in \mathcal{N}}, (J_i)_{i \in \mathcal{N}}, \{\theta_m\}_{m=1}^M \rangle$ and $\hat{\mathcal{G}} = \langle \mathcal{N}, (\mathcal{X}_i)_{i \in \mathcal{N}}, (\hat{J}_i)_{i \in \mathcal{N}}, \{\theta_m\}_{m=1}^M \rangle$, having different games for different realisations of the M-multisample. In order to play these games, we will effectively iterate two steps until convergence.

At each iteration k :

1. Collect $\sigma(x(k)) = \sum_{i \in \mathcal{N}} x_i(k)$ from coordinator.
2. Agents update strategies in parallel, resulting in a new decision vector $x_i(k+1)$ based on the past iterate $x_i(k)$ and the aggregate $\sigma(x(k))$.

The form of the update step will vary according to the algorithm being used.

3.2 Solution Concepts

In non-cooperative games it is necessary to discuss the concept of equilibria as solution concepts. In this report we will discuss two equilibrium concepts, firstly the Nash Equilibrium which is defined as the point in which no agent is able to improve upon their objective function, provided all other agents remain fixed in their schedules. We denote by x^N the collection of agents' schedules that satisfies this equilibrium defined as follows:

Definition 3.2.1 (Nash Equilibrium) *We denote by $\Omega \subseteq \mathcal{X} = \mathcal{X}_1 \times \dots \times \mathcal{X}_N$ the set of Nash equilibria of \mathcal{G} defined as*

$$\Omega = \{x^N = (x_i^N)_{i \in \mathcal{N}} \in \mathcal{X} : x_i^N \in \arg \min_{x_i \in \mathcal{X}_i} J_i(x_i, x_{-i}^N), \forall i \in \mathcal{N}\}. \tag{3.3}$$

Where $J_i(x_i, x_{-i}^N)$ can be replaced with $\hat{J}_i(x_i, x_{-i}^N)$ to define the equilibria for our second game.

An alternative equilibrium we can define is that of the Wardrop equilibrium. In this equilibrium, we are interested in the point at which no agent can improve upon their objective function given that the sum of all agents' schedules is fixed (i.e. $\sigma(x)$ is fixed), we shall denote by x^W this equilibrium formally defined as:

Definition 3.2.2 (Wardrop Equilibrium) We denote by $\Psi \subseteq \mathcal{X} = \mathcal{X}_1 \times \dots \times \mathcal{X}_N$ the set of Wardrop equilibria of \mathcal{G} defined as

$$\Psi = \{x^W = (x_i^W)_{i \in \mathcal{N}} \in \mathcal{X} : x_i^W \in \arg \min_{x_i \in \mathcal{X}_i} J_i(x_i, \sigma(x^W)), \forall i \in \mathcal{N}\}. \quad (3.4)$$

Notice that in this concept we implicitly assume that the agents' fleet is of large size such that the effect of an individual agent on $\sigma(x)$ is considered to be negligible, and hence in the minimisation in (3.4) the argument of σ is fixed. We also note that $\Omega \subseteq \Psi$.

In both cases, we have defined the equilibria for the game \mathcal{G} but they extend trivially to the game $\hat{\mathcal{G}}$. With these equilibria defined we are also interested in how their associated costs compare to the social optima J^* and \hat{J}^* , which we calculate from the central optimisation problems. We can thus also define the efficiency of a given equilibrium (we will define this for the Nash equilibrium but it holds analogously for the Wardrop equilibrium as well).

Definition 3.2.3 (Equilibrium Efficiency) We denote by η the efficiency of a given equilibrium relative to the social optimum defined as:

$$\eta = \frac{\sum_{i \in \mathcal{N}} J_i(x_i^N, \sigma(x_i^N, x_{-i}^N))}{J^*}. \quad (3.5)$$

Where J_i refers to the local cost function as above and J^* refers to the centralised optimum. Using these solution concepts we can then move on to define algorithms capable of reaching such equilibria, and later on, we will empirically evaluate their efficiencies.

3.3 Proximal Based Algorithm

The first algorithm we will define is a proximal based algorithm, where agents choose their best response in the update step. This algorithm is taken from [4] and requires an augmentation of the problem to provide a differentiable form as well as the addition of a regularisation term to ensure a unique NE is chosen. Non-differentiability arises due to the presence of the max term in the agents' objective functions.

The first game \mathcal{G} is augmented through an adjustment to the pointwise maximum (which introduces the non-differentiability) by replacing this maximum with a sum $h(x_i, x_{-i}, y) = \sum_{m=1}^M y_m g(x_i, x_{-i}, \theta_m)$, so that, instead of $J(x_i, x_{-i})$, agents receive the cost $J'_i(x_i, x_{-i}) = f_i(x_i, x_{-i}) + h(x_i, x_{-i}, y)$.

We then also include an additional agent which seeks to maximise h with respect to y , so that h

then approximates the pointwise maximum:

$$\max_{y \in \Delta} \sum_{m=1}^M y_m g(x, \theta_m) \quad (3.6)$$

$$\Delta = \{y \in \mathbb{R}^M : y \geq 0, \sum_{m=1}^M y_m = 1\}. \quad (3.7)$$

Similarly in the game $\hat{\mathcal{G}}$ we introduce a new function $\hat{h}(x_i, x_{-i}, y^i) = \sum_{m=1}^M y_m^i g(x_i, x_{-i}, \theta_m)$ which uses a unique vector y^i for each individual agent. Thus we must also introduce N additional agents to maximise each y vector separately using g_i which depends on which agent it is being calculated for:

$$\max_{y \in \Delta} \sum_{m=1}^M y_m g_i(x_i, x_{-i}, \theta_m). \quad (3.8)$$

Again, we define a new cost $\hat{J}_i'(x_i, x_{-i}) = f_i(x_i, x_{-i}) + h_i(x_i, x_{-i}, y)$ which is now being optimised.

We then use our schedule x alongside y to denote a new strategy for all agents as $z \in \mathcal{X} \times \Delta$ and can thus define the algorithm for \mathcal{G} as follows:

Algorithm 1: Proximal based algorithm for finding NE of \mathcal{G}

Input: $\bar{z}^{(0)} \in \mathcal{X} \times \Delta, \tau > 0, \gamma_{inn} > 0, \gamma_{out} > 0$

```

1  $k \leftarrow 0$ ;
2 repeat outer loop
3    $l \leftarrow 0$ ;
4   repeat inner loop
5     for  $i \leftarrow 1$  to  $N$  do
6        $x_i^{(l+1)} \leftarrow \arg \min_{v_i \in \mathcal{X}_i} f_i(v_i, x_{-i}^{(l)}) + h(v_i, x_{-i}^{(l)}, y^{(l)}) - \frac{\delta^{(k)}}{2} \|(v_i, x_{-i}^{(l)}, y^{(l)})\|^2 + \frac{\tau}{2} \|v_i - \bar{x}_i^{(k)}\|^2$ ;
7        $y^{(l+1)} \leftarrow \arg \max_{v \in \Delta} h(x^{(l)}, v) - \frac{\delta^{(k)}}{2} \|(x^{(l)}, v)\|^2 + \frac{\tau}{2} \|v - \bar{y}^{(k)}\|^2$ ;
8      $l \leftarrow l + 1$ ;
9   until  $\|z^{(l)} - z^{(l-1)}\| \leq \gamma_{inn}$ ;
10   $\bar{z}^{(k+1)} \leftarrow z^{(l)}$ ;
11   $k \leftarrow k + 1$ ;
12 until  $\|\bar{z}^{(k)} - \bar{z}^{(k-1)}\| \leq \gamma_{out}$ ;

```

The key update steps for this algorithm occur on lines 6 and 7. Line 6 contains the update step for the agents (which could be completed in parallel) and takes the form of a best response step for agents' decisions. The last term acts as a regularisation term that adds "inertia" from agents' decisions in the previous time step. Line 7 admits a very similar interpretation, but instead refers to the additional

maximising agent. Lines 9 and 10 show our convergence checks, in general we choose $\gamma_{inn} < \gamma_{out}$. The exact choice of γ both here and in later equations is left up to the user, the smaller this value is the longer convergence will take but the closer the solution will be to an exact Nash Equilibrium, we provide a few different values in Chapter 5, typically choosing a value around 10^{-5} .

The algorithm for $\hat{\mathcal{G}}$ is then similar with the exception that the update for y is performed inside the for loop for each agent's y_i (which is then used in the update for x) as well as relevant functions being replaced by their corresponding counterparts.

In order to hold a formal proof that our equation will tend to a Nash equilibrium for \mathcal{G} we require $\{\delta^{(k)}\}_{k=0}^{\infty}$ to be a sequence such that $\delta^{(k)} > 0$ for all k , $\sum_{k=0}^{\infty} \delta^{(k)} = \infty$ and $\lim_{k \rightarrow \infty} \delta^{(k)} = 0$ [28, Theorem 21]. Requirements for a lower bound on τ are also defined in [28, Lemma 20]. To the best of our knowledge, such a proof does not exist for the case of $\hat{\mathcal{G}}$, however we will later provide numerical evidence that both games do tend to Nash equilibria. The downside to this algorithm is in its complexity, for each inner loop iteration we need to perform $N + 1$ quadratic programs for \mathcal{G} or $2N$ for $\hat{\mathcal{G}}$. The complexity of these quadratic programs may also be large themselves, potentially leading to very long run times.

3.4 Gradient Descent Algorithm

Using a similar augmentation as before (although we now neglect the regularisation terms) we can also define a gradient descent algorithm, based loosely on gradient ascent for convex programming [29] and adapted for our specific game. In this case, we do not directly solve a minimisation problem for each agent, instead, agents take a step in the negative gradient of their objective. We do however continue to directly solve the quadratic program in the maximisation step.

Algorithm 2: Gradient descent algorithm for finding NE of \mathcal{G}

Input: $z^{(0)} \in \mathcal{X} \times \Delta, \gamma > 0, \tau > 0$

```

1  $k \leftarrow 0$ ;
2 repeat
3   for  $i \leftarrow 1$  to  $N$  do
4      $x_i^{(k+1)} \leftarrow \prod_{\mathcal{X}_i} \left( x_i^{(k)} - \epsilon^{(k)} \nabla_{x_i} (f_i(x_i^{(k)}, x_{-i}^{(k)}) + h(x_i^{(k)}, x_{-i}^{(k)}, y^{(k)})) \right)$ ;
5      $y^{(k+1)} \leftarrow \arg \max_{v \in \Delta} h(x^{(k+1)}, v) - \frac{\delta^{(k)}}{2} \|(x^{(k+1)}, v)\|^2 + \frac{\tau}{2} \|v - \bar{y}^{(k)}\|^2$ ;
6      $\bar{y}^{(k+1)} \leftarrow y^{(k+1)}$ ;
7      $k \leftarrow k + 1$ ;
8 until  $\|z^{(k)} - z^{(k-1)}\| \leq \gamma$ ;
```

For this algorithm the main update steps are on lines 4 and 5. The maximisation step on line 5 is identical to that of Algorithm 1, whereas line 4 now takes the form of a projected gradient step, whereby each agent takes a step in the direction of the negative gradient and projects back onto the feasible set.

Once again, this algorithm can be relatively easily extended to $\hat{\mathcal{G}}$ by iterating over update steps for y .

In Algorithm 2 we denote by $\prod_{\mathcal{X}_i}(\cdot)$ the projection on to the set \mathcal{X}_i . We hold the same requirements on $\delta^{(k)}$ as in 1. We now offer two choices for the value of $\epsilon^{(k)}$ which plays the role of the stepsize of the gradient-descent step that appears inside the projection operator. One option is to define it as a sequence satisfying requirements identical to $\delta^{(k)}$ as is the more common approach in gradient descent methods [29]. Alternatively, we can define $\epsilon^{(k)} = \epsilon$, where ϵ is a constant stepsize. With the constraints $\epsilon > 0$ and

$$\epsilon < \frac{-L_F^2 + \sqrt{L_F^4 + 4\alpha^2}}{2\alpha} \quad (3.9)$$

as defined in [5] with L_F denoting the Lipschitz constant of the operator $F_N(x) : \mathcal{X} \subset \mathbb{R}^{Nn} \rightarrow \mathbb{R}^{Nn}$,

defined and simplified in (3.10) and α being its monotonicity constant.

$$\begin{aligned}
F_N(x) &:= [\nabla_{x_i} (f_i(x_i^{(k)}, x_{-i}^{(k)}) + h(x_i^{(k)}, x_{-i}^{(k)}, y^{(k)}))]_{i=1}^N \\
&= [\nabla_{x_i} \left(x_i^\top (A_0 \sigma(x) + b_0) + \sum_{m=1}^M y_m \sigma(x)^\top (A_m \sigma(x) + b_m) \right)]_{i=1}^N \\
&= [\nabla_{x_i} (x_i^\top (A_0 \sigma(x) + b_0) + \sigma(x)^\top (A_{tot} \sigma(x) + b_{tot}))]_{i=1}^N \\
&= [\nabla_{x_i} (x_i^\top (A_0(x_i + x_{-i}) + b_0) + (x_i + x_{-i})^\top (A_{tot}(x_i + x_{-i}) + b_{tot}))]_{i=1}^N \quad (3.10) \\
&= [\nabla_{x_i} (x_i^\top A_0 x_i + x_i^\top A_0 x_{-i} + x_i^\top b_0 + x_i^\top A_{tot} x_i + x_i^\top A_{tot} x_{-i} \\
&\quad + x_{-i}^\top A_{tot} x_i + x_{-i}^\top A_{tot} x_{-i} + x_i^\top b_{tot} + x_{-i}^\top b_{tot})]_{i=1}^N \\
&= [2A_0 x_i + A_0 x_{-i} + b_0 + 2A_{tot} x_i + 2A_{tot} x_{-i} + b_{tot}]_{i=1}^N \\
&= [A_0 x_i + A_0 \sigma(x) + b_0 + 2A_{tot} \sigma(x) + b_{tot}]_{i=1}^N.
\end{aligned}$$

We use here $A_{tot} = \sum_{m=1}^M y_m A_m$ and $b_{tot} = \sum_{m=1}^M y_m b_m$. We can similarly define $\hat{F}_N(x)$ for \hat{G} and this results in $\hat{F}_N(x) := [A_0 x_i + A_0 \sigma(x) + b_0 + A_{tot} x_i + A_{tot}^i \sigma(x) + b_{tot}^i]_{i=1}^N$ with $A_{tot}^i = \sum_{m=1}^M y_m^i A_m$ and $b_{tot}^i = \sum_{m=1}^M y_m^i b_m$. With these defined we can then find values for the constants L_F and α .

First, we use the following characterisation of strong monotonicity.

Lemma 3.4.1 ([30, Proposition 2.3.2]) *A continuously differentiable operator $F : \mathcal{K} \subset \mathbb{R}^d \rightarrow \mathbb{R}^d$ is strongly monotone with monotonicity constant $\alpha > 0$ if and only if $\nabla_x F(x) \succeq \alpha I$ (resp. $\nabla_x F(x) \succeq 0$) for all $x \in \mathcal{K}$.*

The definition of the Lipschitz constant of a differentiable functions is as follows:

Definition 3.4.1 (Lipschitz constant for differentiable functions) *A continuously differentiable operator $F : \mathcal{K} \subset \mathbb{R}^d \rightarrow \mathbb{R}^d$ with bounded derivative will have Lipschitz constant $L_F = \sup_x \|\nabla_x F(x)\|_2$.*

Thus, in order to find values for L_F and α , we are in fact interested in the maximum and minimum eigenvalues of the Jacobian of F_N respectively. This follows directly from the definition of the Lipschitz constant and from the fact that defining $\alpha = \inf_x \|\nabla_x F(x)\|_2$ provides a value satisfying Lemma 3.4.1. To determine this Jacobian we will use the fact that

$$\nabla_{x_j} (A_0 x_i + A_0 \sigma(x) + b_0 + 2A_{tot} \sigma(x) + b_{tot}) = \begin{cases} 2A_0 + 2A_{tot}, & \text{if } i = j \\ A_0 + 2A_{tot}, & \text{otherwise} \end{cases} \quad (3.11)$$

And thus provide an equation for the Jacobian as:

$$\nabla_x F(x) = \begin{bmatrix} 2A_0 + 2A_{tot} & A_0 + 2A_{tot} & \dots & A_0 + 2A_{tot} \\ A_0 + 2A_{tot} & 2A_0 + 2A_{tot} & \dots & A_0 + 2A_{tot} \\ \dots & \dots & \dots & \dots \\ A_0 + 2A_{tot} & A_0 + 2A_{tot} & \dots & 2A_0 + 2A_{tot} \end{bmatrix} \quad (3.12)$$

By exploiting the structure of this result we find that the minimum eigenvalue of the Jacobian will simply be the smallest eigenvalue of the matrix $2A_0 + 2A_{tot} - A_0 + 2A_{tot} = A_0$, and since A_0 is diagonal then α is just the smallest value on the diagonal of A_0 . Similarly, the maximum eigenvalue of the Jacobian is the maximum eigenvalue of the matrix $(2A_0 + 2A_{tot}) + (N - 1)(A_0 + 2A_{tot}) = (N + 1)A_0 + 2NA_{tot}$, again since A_0 and A_{tot} are diagonal this is just the maximum value in this matrix.

Because y is optimised in the algorithm, we cannot use this in our calculation, and therefore we are also unable to directly use A_{tot} . Instead, we can find a lower bound on the maximum stepsize by maximising L_F with respect to y . Starting from the fact that L_F is simply the maximum value of $(N + 1)A_0 + 2NA_{tot}$, then noting that A_0 is independent of y we simply need to maximise the largest value of A_{tot} . We recognise that in order to maximise the largest value of A_{tot} , we would simply set $y_m = 1$ corresponding to the matrix A_m which contains the single largest value for $m = \{1, \dots, M\}$. Thus, to find an upper bound on L_F we simply find the largest single value across all matrices A_m for $m = \{1, \dots, M\}$ denoted as a_m^* , find a_0^* the largest value in A_0 and then simply calculate $L_F = (N + 1)a_0^* + 2Na_m^*$.

We now have arrived at an expression for the value of α and an upper bound on L_F . We can now, therefore, evaluate equation (3.9) to find a lower bound on the maximum stepsize for \mathcal{G} . Unfortunately, the structure seen in equation (3.12) for the Jacobian will not follow for $\hat{\mathcal{G}}$ as A_{tot} would need to be replaced by A_{tot}^i , which could have a different value on each row. We thus do not have a nice solution for the maximum stepsize as with \mathcal{G} . However, it is our belief that there will still exist an ϵ small enough to satisfy the constraint in (3.9), and we will later provide numerical evidence that it is still possible for this to converge.

3.5 Subgradient Descent Algorithm

One drawback of Algorithm 2 is that it still requires solving a quadratic program at each iteration, fortunately, this is a reduction in the number of quadratic programs from Algorithm 1. However, in the case of $\hat{\mathcal{G}}$ this would still require N quadratic programs. A further improvement can be made by

returning to the original formulation and defining instead a subgradient, thus removing the requirement for the y vector entirely. However, to the best of our knowledge, there exist no guarantees that this algorithm will converge to a Nash Equilibrium. To find this subgradient we find the currently active θ (i.e. the θ that maximises g) and use the gradient corresponding to the g parameterised by that θ . In the case of more than one θ leading to this maximum we arbitrarily choose the one with the lowest index. This is adapted from subgradient methods similar to those found in [31].

Algorithm 3: Subgradient descent algorithm for finding NE of \mathcal{G}

Input: $x^{(0)} \in \mathcal{X}, \gamma > 0$

```

1  $k \leftarrow 0$ ;
2 repeat
3   for  $i \leftarrow 1$  to  $N$  do
4      $m^* \leftarrow \arg \max_{m \in \{1, \dots, M\}} g(x_i^{(k)}, x_{-i}^{(k)}, \theta_m)$ ;
5      $x_i^{(k+1)} \leftarrow \prod_{\mathcal{X}_i} \left( x_i^{(k)} - \epsilon^{(k)} \nabla_{x_i} (f_i(x_i^{(k)}, x_{-i}^{(k)}) + g(x_i^{(k)}, x_{-i}^{(k)}, \theta_{m^*})) \right)$ ;
6 until  $\|x^{(k)} - x^{(k-1)}\| \leq \gamma$ ;
```

Once again, we can choose between a diminishing or fixed stepsize. We can also similarly define this algorithm for $\hat{\mathcal{G}}$, however, now all this requires is taking an argmax over g_i . Algorithm 3 is thus significantly quicker than Algorithm 1 or 2, provided the gradients are relatively easy to calculate (for us they are as we have quadratic forms for f_i , g and g_i). It is no longer necessary to carry out a full quadratic program at each iteration as the argmax here is simply defined over the finite scenarios we have drawn.

3.6 Wardrop Equilibrium Seeking Algorithm

Thus far all our algorithms have been concerned with finding Nash equilibria of our game, the final algorithm we are concerned with instead finds a Wardrop equilibrium. The benefit of doing this is that we can solve a linear program as opposed to a quadratic program at each time step, and can entirely disregard the uncertainty in \mathcal{G} . Intuitively, we see that as the population size becomes large, the impact of any agent's individual strategy on the aggregate strategy is reduced and the Wardrop equilibrium should tend to a Nash equilibrium.

Algorithm 4: Algorithm for finding Wardrop Equilibrium of \mathcal{G} [5, Algorithm 1]

Input: $x^{(0)} \in \mathcal{X}, \gamma > 0$

```
1   $k \leftarrow 0$  ;  
2   $z^{(0)} \leftarrow \sum_{i \in \mathcal{N}} x_i$  ;  
3  repeat  
4      for  $i \leftarrow 1$  to  $N$  do  
5           $x_i^{(k+1)} \leftarrow \arg \min_{v_i \in \mathcal{X}_i} f_i(v_i, z^{(k)})$  ;  
6           $\sigma^{(k+1)} \leftarrow \sum_{i \in \mathcal{N}} x_i$  ;  
7           $z^{(k+1)} \leftarrow (1 - \frac{1}{k})z^{(k)} + \frac{1}{k}\sigma^{(k+1)}$  ;  
8           $k \leftarrow k + 1$  ;  
9  until  $\|x^{(k)} - x^{(k-1)}\| \leq \gamma$  ;
```

It should be noted that we have here taken the second argument of f_i as the sum of all agents' schedules which we fix, this is in contrast to other algorithms where we fix other agents' schedules not the sum. Since in \mathcal{G} the penalty g is a function of the sum of agents' schedules only and does not have any terms relating solely to x_i then it will become a fixed value in our update step on line 5. For this to work for $\hat{\mathcal{G}}$, we have once again introduced h_i identically to algorithm 1 and added in an update step to maximise with respect to y .

Chapter 4

Scenario Discarding

In this chapter, we will look in more detail into the scenario approach as the means to accompany the equilibrium computed on the basis of data, with guarantees on the probability of a new sample violating our equilibrium, and then further looking into the possibility of scenario removal.

With the game formulation as laid out in Chapter 3, we have characterised our uncertainty in price due to varying loads and production by way of a finite set of samples, termed the M-multisample. This set is drawn from some unknown but fixed distribution. With this characterisation, it is possible to formulate an equation to describe the probability that the Nash equilibrium produced by this set will be violated with the introduction of a new sample, and this will be laid out in Section 4.1.

We will also investigate the possibility of removing samples from the M-multisample to reduce agents' individual cost functions, since we can choose to remove a sample that is causing the greatest cost increase for all agents a reduction in individual cost follows naturally. This sample removal will have the downside of leading to a less robust equilibrium, since new samples are more likely to violate our equilibrium. This motivates a trade-off between cost and robustness that would need to be chosen to suit the overall needs of the network.

4.1 Theoretical Bounds on the Violation Probability

Firstly, we will explore some equations which quantify the probability of violation in our games, and then the expected change with the removal of samples. Much of this builds upon previous work from [4]

In order to do this we will consider the set of Nash equilibria Ω of our game \mathcal{G} , as defined in (3.3). Then further consider the introduction of a new sample $\theta \in \Theta$ such that a new game can be defined $\mathcal{G}^+ = \langle \mathcal{N}, (\mathcal{X}_i)_{i \in \mathcal{N}}, \{\theta_m\}_{m=1}^M \cup \{\theta\} \rangle$, with a set of Nash Equilibria Ω^+ . We define the violation

probability as the probability that the original Nash equilibrium $x^N \in \Omega$ is not a Nash equilibrium of the new game:

$$V(x^N) = \mathbb{P}\{\theta \in \Theta : x^N \notin \Omega^+\}. \quad (4.1)$$

It should be noted that since x^N depends on the random M -multisample it is inherently a random variable, and thus since $V(x^N)$ takes x^N as an argument it is also a random variable. We are therefore interested in finding a confidence level on $V(x^N)$ with respect to the product measure $\mathbb{P}^M = \mathbb{P} \times \dots \times \mathbb{P}$, but first we must define a few key terms.

We will define by $\Phi : \Theta^M \rightarrow \Omega$ a *single-valued* mapping from the set of M -multisamples to the set of Nash equilibria of \mathcal{G} . In our context Φ could be any of the algorithms presented in Chapter 3. Proof that the proximal based algorithm (Algorithm 1) is such a single-valued mapping is provided in [4], we have assumed our other algorithms are similarly single-valued but leave the proof to future work.

Definition 4.1.1 (Support sample [32]) *Fix any i.i.d. M -multisample $(\theta_1, \dots, \theta_M) \in \Theta^M$ and let $x^N = \Phi(\theta_1, \dots, \theta_M)$ be a Nash equilibrium of \mathcal{G} . Let $x^o = \Phi(\theta_1, \dots, \theta_{s-1}, \theta_{s+1}, \dots, \theta_M)$ be a Nash equilibrium of a new game with sample θ_s removed. If $x^o \neq x^N$ then we call θ_s a support sample*

Definition 4.1.2 (Compression set - Adapted from [25]) *Fix any i.i.d. M -multisample $(\theta_1, \dots, \theta_M) \in \Theta^M$ and let $x^N = \Phi(\theta_1, \dots, \theta_M)$ be a Nash equilibrium of \mathcal{G} . Consider a subset $\mathcal{C} \subseteq \{\theta_1, \dots, \theta_M\}$ and let $x^o = \Phi(\mathcal{C})$ be a Nash equilibrium of a game consisting of samples only in this subset. If $x^o = x^N$ then we call \mathcal{C} a compression set.*

Let $\mathfrak{C}(\theta_1, \dots, \theta_M)$ be the collection of all compression sets \mathcal{C} . We define the compression cardinality d^* to be the cardinality of some compression set $\mathcal{C} \in \mathfrak{C}(\theta_1, \dots, \theta_M)$.

Definition 4.1.3 (Non-degeneracy - Adapted from [33]) *For any $M \in \mathbb{N}$ with \mathbb{P}^M -probability equal to 1, the Nash equilibrium $x^N = \Phi(\theta_1, \dots, \theta_M)$ coincides with the Nash equilibrium returned by Φ when the latter takes only the support samples in its argument. The corresponding game is then called non-degenerate; otherwise, it is degenerate.*

Whilst it can generally be difficult to prove non-degeneracy we will later provide numerical evidence in Chapter 5 to show that the results assuming non-degeneracy do appear to hold for our game \mathcal{G} . Without imposing such an assumption we have that

$$\mathbb{P}^M\{(\theta_1, \dots, \theta_M) \in \Theta^M : V(x^N) \leq \epsilon(d^*)\} \geq 1 - \beta, \quad (4.2)$$

this provides us with a confidence level $\beta \in (0, 1)$ that the violation rate is less than $\epsilon(d^*)$ where $\epsilon : \{0, \dots, M\} \rightarrow [0, 1]$ is a function satisfying

$$\epsilon(M) = 1, \text{ and } \sum_{k=0}^{M-1} \binom{M}{k} (1 - \epsilon(k))^{M-k} = \beta. \quad (4.3)$$

One such function, defined in [25], which simply splits β evenly across the sum is:

$$\epsilon(k) = 1 - \sqrt[M-k]{\frac{\beta}{M \binom{M}{k}}}. \quad (4.4)$$

We have therefore arrived at a result allowing us to establish the probability that drawing a new sample would cause us to move from our current equilibrium, with some confidence of at least $1 - \beta$ (i.e. Equation (4.2) holds for any $\beta \in [0, 1]$ provided ϵ satisfies (4.4) (or more generally (4.3)) proof of this is available in [34]). These statements are also applicable to $\hat{\mathcal{G}}$, however computation of the compression set in this case requires some further consideration.

4.2 Compression Set Computation

There are several possible methods for calculating the compression set cardinality d^* for use in (4.2) which we will cover here. Ideally, we would like to find a minimal compression set in order to find the tightest possible bound. However, this can be practically difficult and computationally expensive, and so we present a few options to upper bound the compression set cardinality.

Firstly, a simple a priori upper bound is to use the sample-independent quantity $nN + 1$ [4]. This assumes that $f_i(\cdot, x_{-i})$ and $g(\cdot, x_{-i}, \theta)$ are convex for all $i \in \mathcal{N}$, for every fixed $x_{-i} \in \mathcal{X}_{-i}$ and $\theta \in \Theta$, and that the game is non-degenerate. These assumptions allow us to use this bound in place of the looser bound $(n + 1)N$ (however this bound does also require non-degeneracy, we will not prove that our game is non-degenerate but show later that these bounds are both reasonably conservative for our game).

However, this upper bound may be fairly loose, which leads us to examine some a posteriori methods for calculating a smaller compression set. One option is a greedy computation as laid out in Algorithm 5, outlined in [25]. In this algorithm scenarios are removed one by one and we then check if the Nash equilibrium has changed as a result of this removal. This method would lead to a reasonable estimation of the upper bound of the minimal compression set cardinality but is faced with 2 major drawbacks. The computational cost of such a scheme would be excessive, requiring the game to be played out M times, with each iteration requiring some complex asymptotic calculation as laid out in

Chapter 3. Furthermore, the condition on line 3 is difficult to evaluate, with the exact value of the Nash equilibrium being somewhat dependent on the numerical accuracy chosen.

Algorithm 5: Greedy computation for finding compression cardinality [4]

Input: $\mathcal{C} = \{\theta_1, \dots, \theta_M\}, x^N \in \Omega$

```

1 forall  $m \in \{1, \dots, M\}$  do
2    $x^o \leftarrow \Phi(\mathcal{C} \setminus \theta_M)$  ;
3   if  $x^o = x^N$  then
4      $\mathcal{C} \leftarrow \mathcal{C} \setminus \theta_M$  ;

```

An alternative, much simpler, option for evaluating the compression cardinality in the cases where we are using the gradient descent algorithm (Algorithm 2) or the proximal based algorithm (Algorithm 1) is through the inspection of the y vector at termination (or the union of all y_i vectors for $\hat{\mathcal{G}}$). Any non-zero elements in this vector relate to elements belonging to a compression set since only the non-zero elements of this vector will have an effect on the algorithm. More formally, the set

$$\mathcal{Y}^* \triangleq \{m \in \{1, \dots, M\} : y_m^* > 0\}, \quad (4.5)$$

(where y^* relates to the optimal y vector at the Nash equilibrium) corresponds to the indices of a compression set $x^N = \Phi(\theta_1, \dots, \theta_M) = \Phi(\{\theta_m\}_{m \in \mathcal{Y}^*})$. A similar result for the compression set also holds for $\hat{\mathcal{G}}$ with $\hat{\mathcal{Y}}^* \triangleq \{m \in \{1, \dots, M\} : \exists i \in \mathcal{N}(y_m^i)^* > 0\}$, equivalently we could use $\hat{\mathcal{Y}}^* \triangleq \{m \in \{1, \dots, M\} : (\sum_{i \in \mathcal{N}} (y_m^i)^*)_m > 0\}$. With this formulation we see that it could be useful to define

$$\hat{y} = \frac{\sum_{i \in \mathcal{N}} y^i}{N}. \quad (4.6)$$

These values must relate to a compression set since, provided we have arrived at a Nash equilibrium and y has converged to y^* , then $\sum_{m=1}^M y_m^* g(\sigma(x), \theta_m) = \sum_{m \in \mathcal{Y}^*} y_m^* g(\sigma(x), \theta_m)$. This holds since all other values of y are 0. If we were to run another iteration of the algorithm with this smaller set then no further changes could occur since $h(\cdot)$ is maintained by this equality and no agent would see any improvement in any direction due to the nature of the Nash equilibrium. Since no changes in x occur the optimisation step for y would not result in any changes either. Thus, the result would hold that $x^o = x^N$ for this set $\{\theta_m\}_{m \in \mathcal{Y}^*}$ and hence this is a compression set.

This procedure provides the means to determine the compression cardinality for our game in an a posteriori fashion, and we will show in Chapter 5 that this provides a sufficient upper bound for $V(x^*)$. Unfortunately, we do not have a similar method for estimating the compression cardinality in

the subgradient algorithm (Algorithm 3) since this algorithm is not augmented with the addition of the y vector.

4.3 Scenario Discarding Algorithms

Now that we have looked into some qualities regarding the scenario approach and how we can find bounds on the violation rate we will look into the possibility of scenario discarding. This approach allows improvements on agents' costs and we will see that it requires some trade-off between cost improvement and violation rate, but first, we will look into defining exactly how we go about discarding samples. We will look at two algorithms of interest, one which removes the “worst” scenario, and one which removes all scenarios in the compression set.

Firstly, removal of the worst scenario is achieved by removing the sample which has the largest value in y (i.e. $r = \arg \max_{m \in \{1, \dots, M\}} y_m^*$). This sample should be the sample causing the highest penalty since y is optimised to maximise the penalty term, then the largest value in y corresponds to the worst penalty. The algorithm which outlines the steps to carry out the removal is outlined in Algorithm 6.

Algorithm 6: Worst scenario removal

Input: $y^*, \{\theta_m\}_{m=1}^M$

- 1 $r = \arg \max_{m \in \{1, \dots, M\}} y_m^*$;
- 2 $\{\theta_m\}_{m=1}^{M-1} = \{\theta_m\}_{m=1}^M \setminus \theta_r$;
- 3 $\Delta \leftarrow \{y \in \mathbb{R}^{M-1} : y \geq 0, \sum_{m=1}^{M-1} y_m = 1\}$;
- 4 $M \leftarrow M - 1$;
- 5 $y \leftarrow \prod_{\Delta} y^*$;

This algorithm takes in the optimised y vector and the M-multisample and returns a multisample that contains one less element and a feasible y vector which could then be used in the next run of the algorithm. In order to do this we find the largest element of y on line 1, and remove the corresponding sample on line 2. Then the last few lines redefine the feasible set for y s and project the old y vector on to this new feasible set.

An alternative algorithm involves the removal of all elements of a compression set (which could be the set as defined by \mathcal{Y}^*), this would remove significantly more scenarios at each iteration and could therefore lead to a much faster cost improvement. However, this may not always work particularly well, for example, the least impactful element of the compression set may not be affecting the cost significantly and other elements of the M-multisample not present in the current compression set could

be more impactful later on. We will provide an empirical comparison in Chapter 5. The algorithm for carrying out sample removal in this manner is laid out in Algorithm 7.

Algorithm 7: Entire compression set removal

Input: $y^*, \{\theta_m\}_{m=1}^M$

```

1 for  $i \in \{1, \dots, M\}$  do
2   if  $\theta_i \neq 0$  then
3      $\{\theta_m\}_{m=1}^{M-1} = \{\theta_m\}_{m=1}^M \setminus \theta_i$ ;
4      $M \leftarrow M - 1$ ;
5  $\Delta \leftarrow \{y \in \mathbb{R}^M : y \geq 0, \sum_{m=1}^M y_m = 1\}$ ;
6  $y \leftarrow 0^M$ ;
7  $y_1 \leftarrow 1$ ;

```

In this algorithm rather than finding the largest element of y we instead find all non-zero elements of y and remove the corresponding samples, before again redefining the feasible set for y . Instead of projecting we know simply make the first element $y_1 = 1$ since after removing all non-zero elements y no longer contains any useful information and so a projection is not useful.

Removing fewer samples than are in the compression set would require some method to order samples to decide which to remove. The most obvious choice for this is to order by magnitude, thus removing the samples that cause the greatest increase in cost.

We also recognise that both these algorithms would similarly work for $\hat{\mathcal{G}}$, with y replaced by \hat{y} as defined in (4.6).

4.4 Violation Rate After Discarding Samples

We have now defined 2 algorithms for removing samples, we now look at how we expect this removal to affect our violation rate, as well as how we can decide how many samples to remove whilst staying above a certain violation rate.

We define $x_k^N = \Phi(\{\theta_m\}_{m=1}^M \setminus \{\theta_n\}_{n \in \mathcal{K}})$ to be the Nash equilibrium after k samples have been removed according to one of the algorithms above, where we denote by \mathcal{K} the set of indices relating to samples selected for removal.

Theorem 4.4.1 ([26, Theorem 2.1]) *Let $\beta \in (0, 1)$ be a small (typically on the order of 10^{-5}) confidence parameter value if M and k are such that*

$$\binom{k+d-1}{k} \sum_{i=0}^{k+d-1} \binom{M}{i} \epsilon^i (1-\epsilon)^{M-i} \leq \beta, \quad (4.7)$$

(where $d = (n + 1)N$ is the number of optimisation variables and k the number of samples removed), then $\mathbb{P}^M\{V(x_k^N) \leq \epsilon\} \geq 1 - \beta$.

Assuming convexity of both $f_i(\cdot)$ and $g(\cdot)$ would again allow us to use $d = nN + 1$ to obtain a tighter approximation. This is not a particularly attractive equation to solve directly, and further work in [26] shows that this can be rearranged to provide a lower bound on k as a function of the violation rate, confidence level, multisample size, and the number of optimisation variables.

$$k \leq \epsilon N - d + 1 - \sqrt{2\epsilon N \ln \frac{(\epsilon N)^{d-1}}{\beta}}. \quad (4.8)$$

Note that d in (4.8) is independent of the samples according to [26]. Hence, in our context we could select d as discussed at the beginning of Section 4.2. However, selecting d a posteriori by means of Algorithms 6 or 7, and then deciding the number of samples that may be removed by (4.8) leads to less conservative performance, while empirical validation indicates that the theoretical bound is still valid.

Equation (4.8) allows us to choose an acceptable confidence and violation rate and then use both of these to find an acceptable number of samples to remove. We will later compare the predicted violation rates from all of the different methods presented here to compare the quality of each prediction as well as the ease of computation.

In this chapter we have defined a number of different bounds on ϵ , allowing us to provide an upper bound on the violation probability as defined in (4.1). The two algorithms we make use of to predict this ϵ are those in (4.4) and (4.8). In the case of (4.4) we require some estimate of the compression set cardinality, and in order to do so we again have two options available to us. Firstly, we can make use of an a priori estimate $nN + 1$ (or the more general a priori estimate $(n + 1)N$ if our game does not satisfy some constraints on convexity), we refer to the bound on ϵ produced when using this estimate of the compression set cardinality as the *a priori* bound. Alternatively, following convergence we have a number of options available to estimate the cardinality of a smaller compression set, in particular we make use of (4.5) and refer to the bound produced on ϵ as the *a posteriori* bound. Finally, we can rearrange (4.8) to solve for ϵ for a given number of samples removed, we refer to this estimate as the *discarding prediction*. In the case where we wish to use our a priori or a posteriori bounds during sample removal we simply decrement the value of M used in the equation. A comparison of these various bounds for the game \mathcal{G} is available in Figure 5.4, and for $\hat{\mathcal{G}}$ in Figure 5.12.

Chapter 5

Numerical Results

In this chapter we will now look at numerical results following simulations of the games as laid out in the previous chapters. We will start by looking at the game with a common penalty term, and will investigate a number of the theoretical statements made throughout this report as well as offering a comparison of the different algorithms. Finally, we will investigate how the game might be played in a real world scenario where new scenarios are uncovered incrementally.

5.1 General Setup

For our simulations, unless otherwise stated, we will be using a population size of 10 agents, 6 timeslots and with A_0 being drawn from historical data. The diagonal elements of A_m are drawn from a log-normal distribution and b_m from a uniform distribution, with a sample size of 500. Agents' energy rates, efficiencies and battery capacity are generated randomly, ensuring that they are always feasible. We also add a constraint that all agents may have a random timeslot in which they are entirely unable to charge.

These simulations were all carried out in MATLAB for Windows 10, on a computer with 16GB RAM and an Intel Core i7-7700HQ running at 2.8GHz.

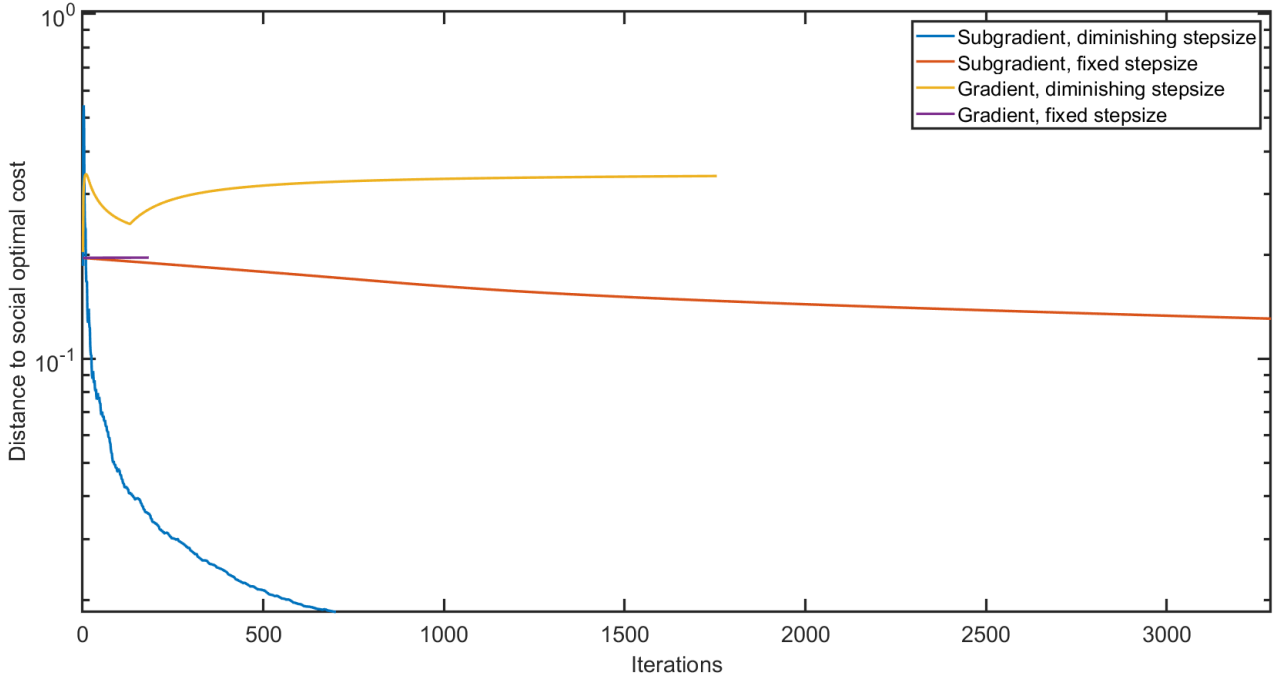
5.2 Results with a Common Penalty Term

In this first section we look solely at the simpler game \mathcal{G} , this game has some more solid theoretical backing and we can thus check that our experimental results match this theory before moving onto $\hat{\mathcal{G}}$.

5.2.1 Algorithm Comparison

We will firstly provide a comparison on the performance of the different algorithms, as laid out in Chapter 3 we have four different algorithms, with two of these algorithms offering a choice of either a diminishing stepsize or a fixed stepsize. As a first step we compare the two most similar algorithms in the gradient and subgradient descent algorithms for a fixed and a diminishing stepsize.

Figure 5.1: Graph providing comparison between different gradient-like algorithms



In Figure 5.1 we are able to see how effective each of these algorithms are and how many iterations they take to converge, this provides us with a useful comparison of the gradient based algorithms. Interestingly the subgradient descent algorithm appears to actually converge to the social optimal cost, achieving an efficiency of 1. Meanwhile, the Gradient descent algorithm results converges reasonably far from the social optimal, and takes more iterations to reach the same convergence criteria. In order to smooth out the results to an extent we have actually evaluated the cost at the average of strategies across iterations for all of these algorithms.

We also notice that the gradient descent algorithm with a fixed stepsize appears to terminate after relatively few iterations, and without making any significant progress. It is our belief that using the fixed stepsize along with the gradient descent leads to incredibly slow progress and, since we have used a common numerical tolerance, γ , this terminates after making very little change in cost (since progress is so slow as to be below the numerical tolerance chosen). By comparison the gradient descent algorithm with diminishing stepsize causes significant changes to the average cost

but terminates relatively far from the social optimal.

This slow movement of the fixed stepsize algorithms appears to also carry over to the subgradient descent algorithms. From looking at the stepsize needed to satisfy (3.9) this is not particularly suprising since we find a stepsize on the order of 10^{-5} , whereas a diminishing stepsize following the rule $\frac{1}{\sqrt{t}}$ will be larger than this fixed stepsize for (approximately) the first 10^{10} iterations. In future we choose to use this diminishing stepsize.

We now look at a comparison between all the main algorithms on two key metrics: time until convergence and efficiency at convergence (see Definition 3.2.3), in order to do this we have chosen a fixed convergence check (whereby we check that the current outer loop cost and previous cost differ by less than 10^{-9}). The graph for the performance of the various algorithms in this simulation is shown in Figure 5.2, and the resulting efficiencies and times to convergence are provided in Table 5.1. We have ommitted the majority of the iterations for the Wardrop equilibrium seeking algorithm in favour of providing a closer look at the other graphs.

Figure 5.2: Algorithm comparison for fixed tolerance

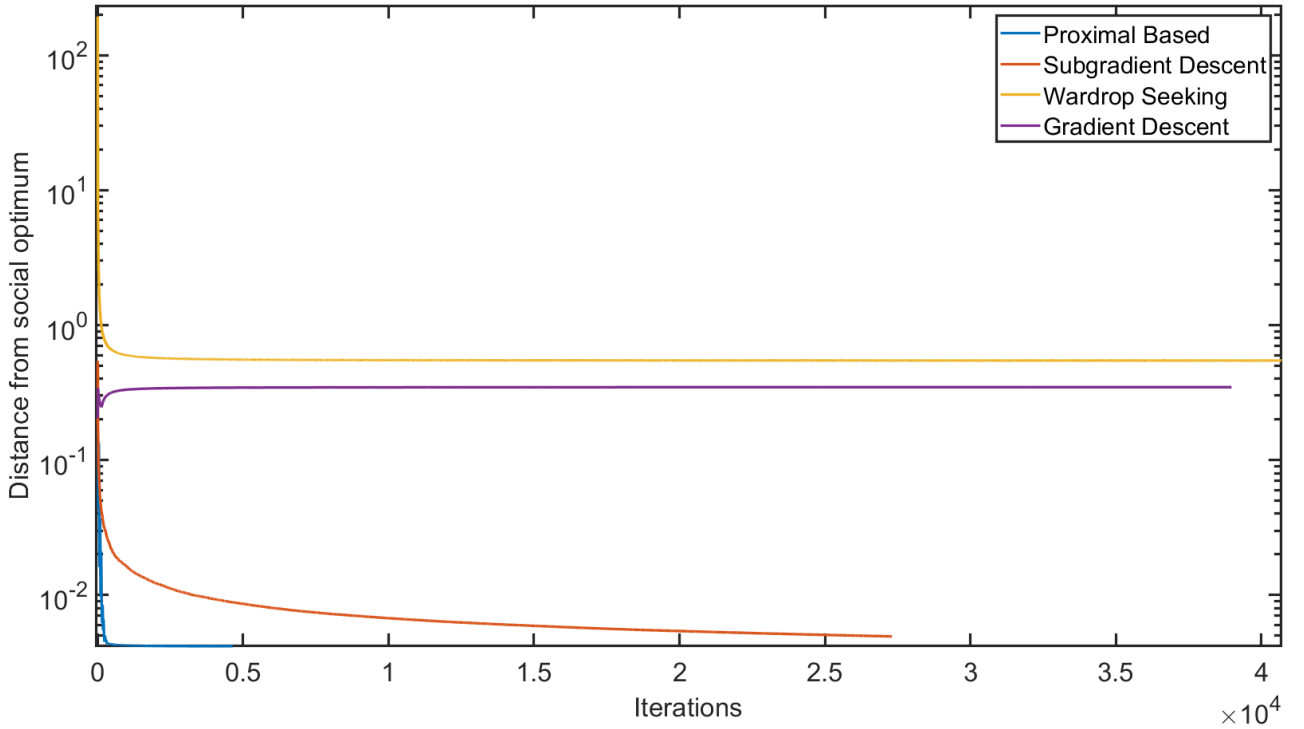


Table 5.1: Algorithm comparison

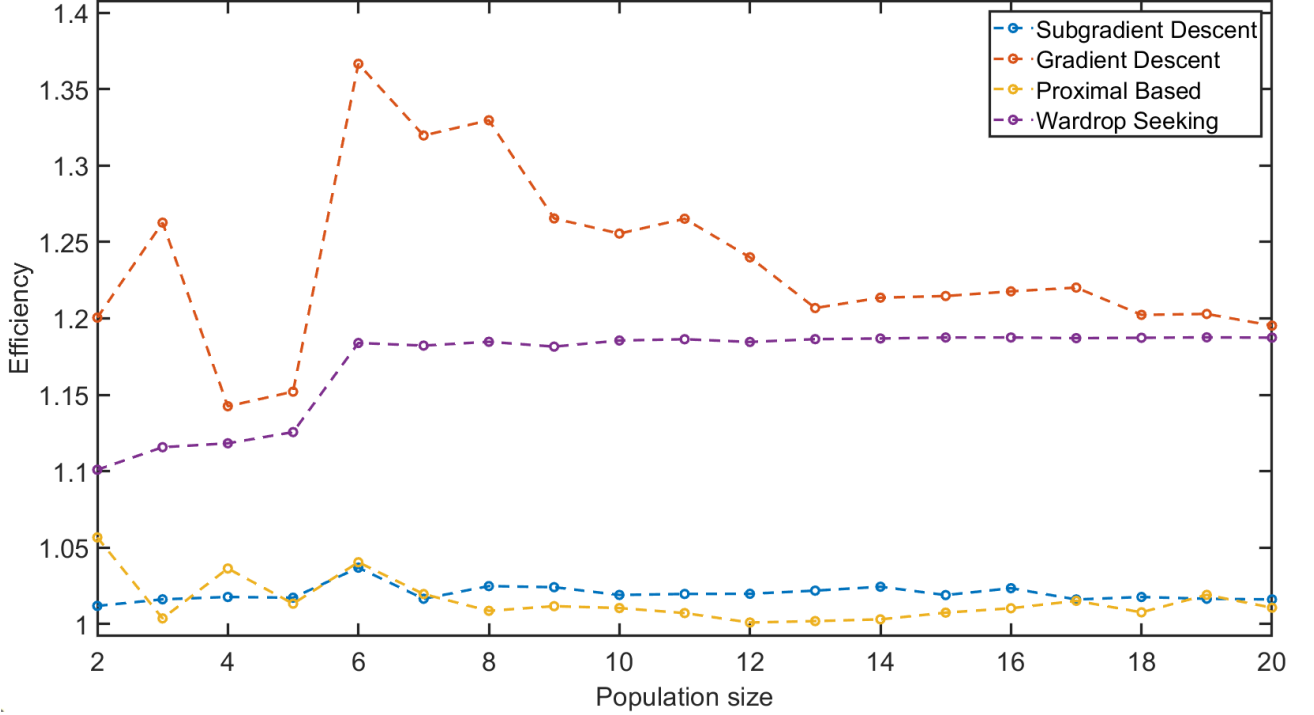
Algorithm	Time until convergence (seconds)	Efficiency at convergence
Proximal based	930	1.0034
Gradient descent	2667	1.2817
Subgradient descent	1742	1.0040
Wardrop equilibrium seeking	47539 ¹	1.4424

Table 5.1 and Figure 5.2 provide us with an initial comparison of our various algorithms so that we can make some preliminary analysis on their various performances. This simulation provides evidence that, given the selected numerical tolerance, the proximal based algorithm exhibits a superior performance. However, this algorithm appears to stay at this equilibrium (as expected), but the subgradient descent algorithm appears to continue to approach the social optimal as iterations increase. One interesting point is that even though the subgradient descent algorithm is slower than the proximal based algorithm overall, it is significantly faster per iteration. We can also see that the Wardrop equilibrium seeking algorithm converges significantly further from the social optimal than other algorithms, however, this is anticipated since the population size is relatively small here and, for \mathcal{G} , the Wardrop equilibrium seeking algorithm does not consider the impacts of the samples in its update step. Since the samples naturally lead to a different optimum schedule the Wardrop equilibrium will never reach the optimum (accounting for the samples) and thus has a poor efficiency at convergence. In addition, the Wardrop equilibrium seeking algorithm appears to have taken by far the longest to converge, this is due to the large number of iterations needed to reach convergence even though each iteration is faster than some of the alternative algorithms, there also seems to be little cost improvement after around 10,000 iterations suggesting that a looser numerical tolerance would be sensible for this algorithm.

We now proceed to show how the algorithms' efficiencies are affected by an increasing population size, from two agents up to twenty.

¹Note the Wardrop equilibrium seeking algorithm was terminated before convergence.

Figure 5.3: Algorithm results with increasing population sizes



In Figure 5.3 we see what happens with an increasing population size, allowing us to gain some insight as to how the population size might affect the performance of our algorithms. This appears to demonstrate that the population size has little effect on the proximal based or subgradient algorithms. We also note that it does not appear to be the case that the Wardrop equilibrium approaches the Nash equilibrium as the population size increases as we would predict, this may in part be due to the structure of the game as discussed previously. Furthermore, we can once again see that the gradient descent algorithm performs very poorly, in fact performing worse than the wardrop equilibrium seeking algorithm at all population sizes.

This concludes our first comparison of the different algorithms available to us, however a final summation will be available in Chapter 6 which will be aided by further work in this chapter.

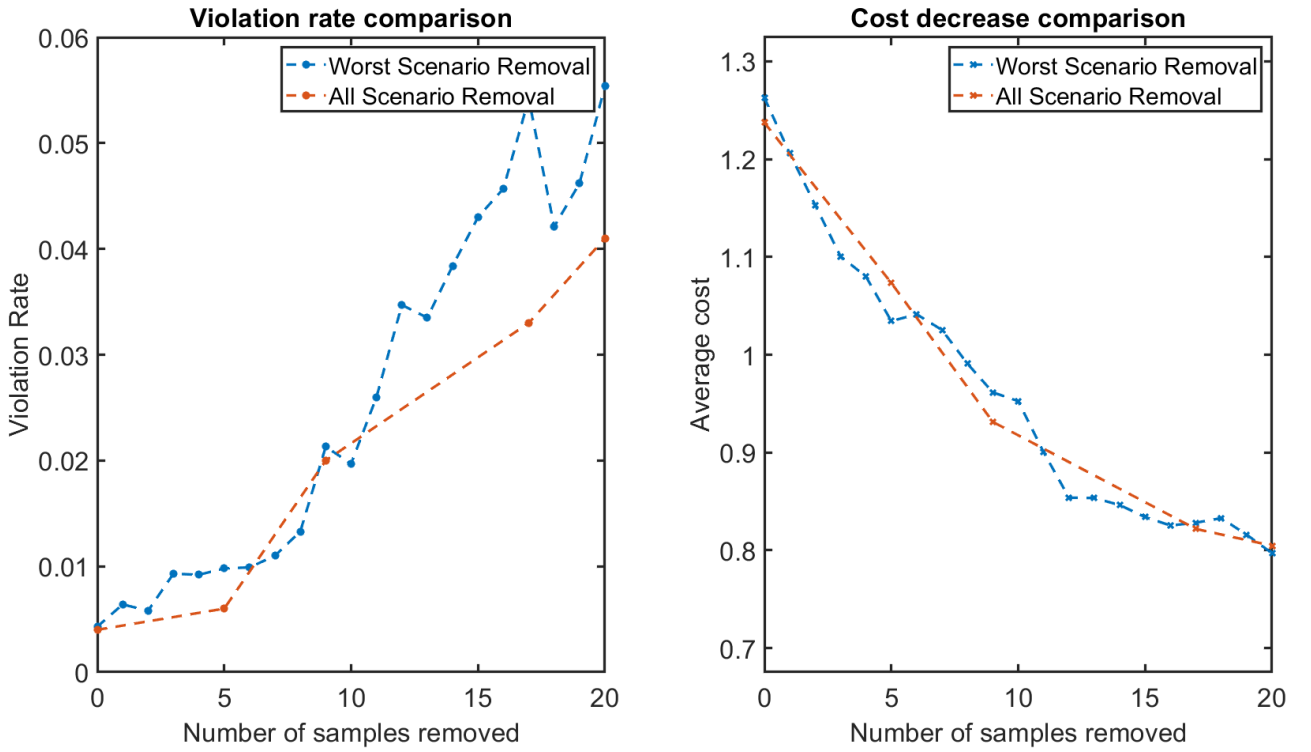
5.2.2 Scenario Discarding

We now look into how discarding of scenarios affects the efficiency and robustness of our equilibria, using the algorithms laid out in Chapter 4. We will firstly use the proximal based algorithm to investigate robustness through the use of Algorithm 8, then confirm that scenario removal leads to a decrease in cost for all algorithms.

Firstly, we compare the two algorithms for removing samples, either by simply removing all active samples (Algorithm 7), or the worst scenario (Algorithm 6). The results for removing up to 20 samples

are displayed in Figure 5.4, with the violation rate calculated using Algorithm 8. This graph shows the effectiveness of the discarding approach we have taken, and provides some insight into the trade-off between violation probability and performance. As we see in the graphs both algorithms result in very similar results in terms of violation rate and cost decrease, suggesting that there is no particular benefit to removing samples one at a time, and in fact removing samples in this way takes between 4 and 5 times as long (for removal up to 20 samples). This graph also demonstrates how the trade-off between violation rate and cost decrease arises, a suitable compromise should be made to decide on exactly how many samples to remove.

Figure 5.4: Comparison of sample removal algorithms



Algorithm 8: Algorithm to empirically validate violation rate

Input: $x^*, y^*, \delta, \tau, \{\theta_m\}_{m=1}^M, \{\theta_k\}_{k=M+1}^{M+K}$
Output: V

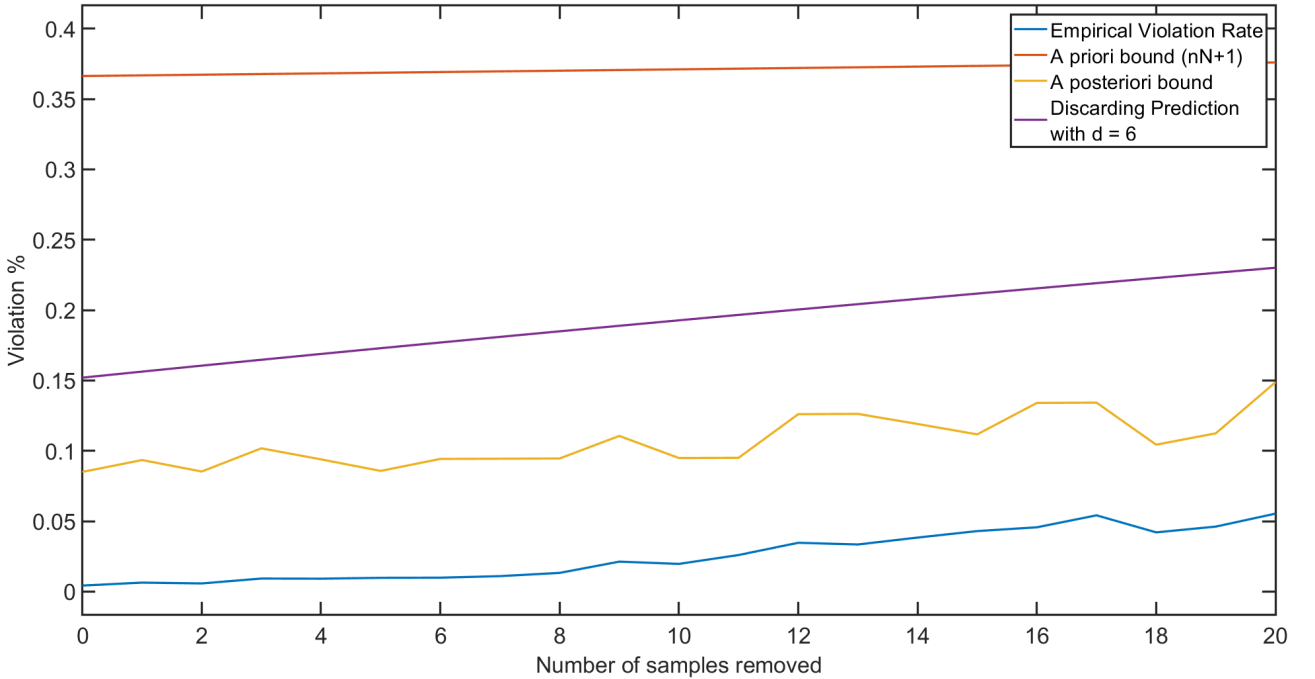
- 1 $V \leftarrow 0$;
- 2 $\Delta \leftarrow \{y \in \mathbb{R}^{M+1} : y \geq 0, \sum_{m=1}^{M+1} y_m = 1\}$;
- 3 **for** $i \in \{1, \dots, K\}$ **do**
- 4 $y \leftarrow \arg \max_{v \in \Delta} h(x^*, v, \{\theta_1, \dots, \theta_M, \theta_{M+i}\}) - \frac{\delta}{2} \|(x^*, v)\|^2 + \frac{\tau}{2} \|v - (y^*, 0)\|^2$;
- 5 **if** $y_{M+1} > 0$ **then**
- 6 $V \leftarrow V + \frac{1}{K}$;

Here we have $\{\theta_m\}_{m=1}^M$ being the set of scenarios used for optimisation and $\{\theta_k\}_{k=M+1}^{M+K}$ being some test set of size K , generated from the same distribution. We repeatedly add one scenario from this

test set to the optimisation set and then check to see if this sample is in the y vector (and thus could lead to a new equilibrium). If this is the case then we increase the empirical violation rate V to reflect that this sample could violate the equilibrium. The resulting value for the empirical violation rate will in fact be an upper bound (since it is possible a scenario may appear in y initially but not necessarily lead to a new equilibrium).

We now look at some of the options for predicting the violation rate, and compare these to the empirical violation rate for the worst case scenario removal. We will look at three options for predicting the violation probability. We use (4.4) for two of these, in the first case using the a priori upper bound on the compression set cardinality of $nN + 1$, the violation probability predicted by this method is henceforth referred to as the *A priori bound*. In the second case, we use (4.5) after the algorithm has converged to find a compression set cardinality (we refer to this violation probability prediction as the *A posteriori bound*). Finally, we solve (4.8) for ϵ to get another bound on the violation probability and refer to this as the *Discarding Prediction*. In all cases we have used a confidence parameter $\beta = 10^{-5}$. The results for these predictions, alongside the empirical violation rate, are presented in Figure 5.5. This is useful information since it allows us to see how relatively conservative each prediction is as more samples are removed.

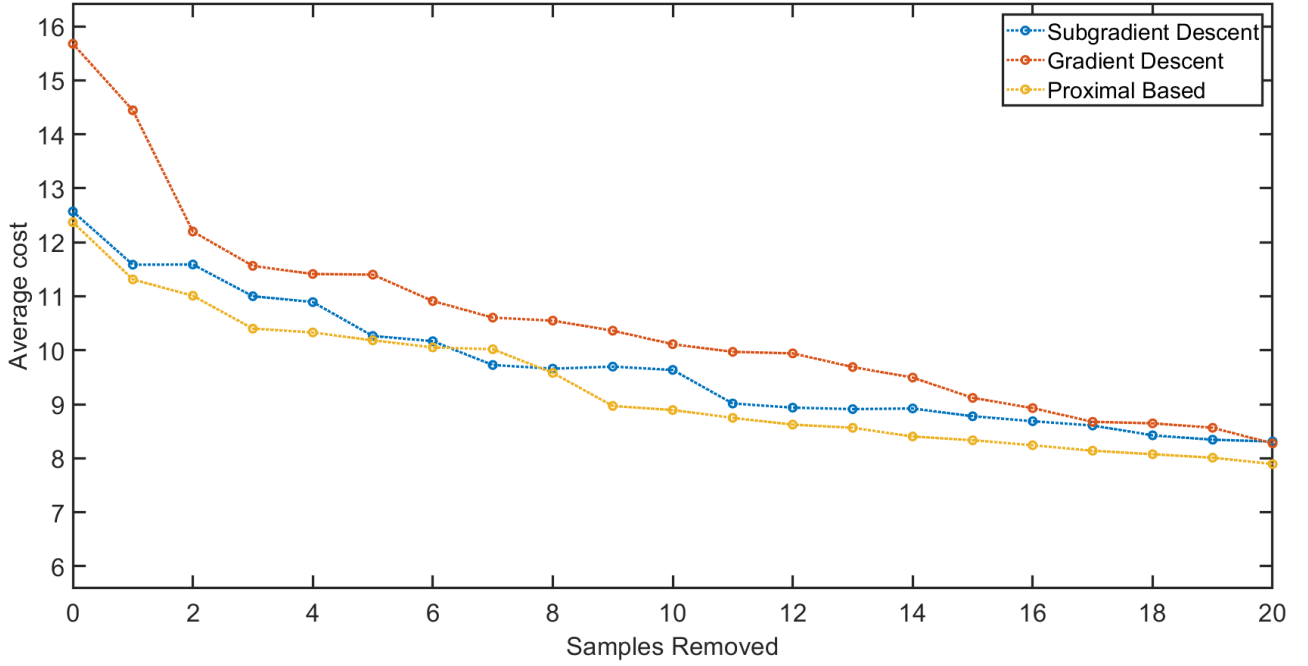
Figure 5.5: Comparison of violation rate prediction



We note that the resulting bounds all hold up to the maximum number of samples removed, although some are significantly more conservative than others. The a posteriori bound appears to be the least conservative.

Another intriguing feature of this graph is the slope of the violation rate which is not monotonically increasing as we might expect, the reasoning behind this is due to the probabilistic nature of the game and the fact we can only use a finite test set. Since every time we remove a sample we should move to a new equilibrium then the samples in the test set that violated the previous equilibrium do not necessarily violate the new equilibrium. We do note however, that the violation rate is generally increasing and with a larger test set we would expect to see a monotonically increasing violation rate (since fewer samples should lead to an increased violation rate [26, 34]).

Figure 5.6: Comparison of different algorithm cost improvements as samples are removed



We now move on to compare how useful the various algorithms are for this scenario removal. The results in Figure 5.6 demonstrate a similar cost decrease for all algorithms which is pretty much as expected, however, we are forced to use the worst case scenario removal algorithm in this comparison since the subgradient algorithm does not provide any easy method for checking the compression set. Because of this, the gradient descent or proximal based algorithms are much better suited for any significant scenario removal.

Furthermore, we have chosen not to present the Wardrop seeking algorithm in these results since the removal of any scenarios does not have an impact on agents' schedules, so the cost decrease is simply accounted for by finding the next worst sample at the equilibrium.

In this section we have been able to demonstrate how the violation rate can be estimated, both a priori and a posteriori, as well as how samples can be removed from the multisample to introduce a trade-off between violation rate and average cost.

5.2.3 Alternative Cost Functions

So far we have used the sum of agents' own cost functions in order to compare the performance of different algorithms, in this section we will investigate a few different alternative utility functions. In each case we shall attempt to motivate the introduction of each function and then use them to compare our algorithms.

Firstly, the utility function we have been using until now is

$$V_1(x) = \sum_{i \in \mathcal{N}} \left(f_i(x_i, x_{-i}) + \max_{m=1, \dots, M} g(x_i, x_{-i}, \theta_m) \right), \quad (5.1)$$

this simply accounts for agents' individual cost and sums these.

An alternative may be for us to only look at the deterministic cost $f_i(\cdot)$, this allows us to compare how different algorithms prioritise this deterministic cost over the uncertain cost. The utility function would then be

$$V_2(x) = \sum_{i \in \mathcal{N}} f_i(x_i, x_{-i}). \quad (5.2)$$

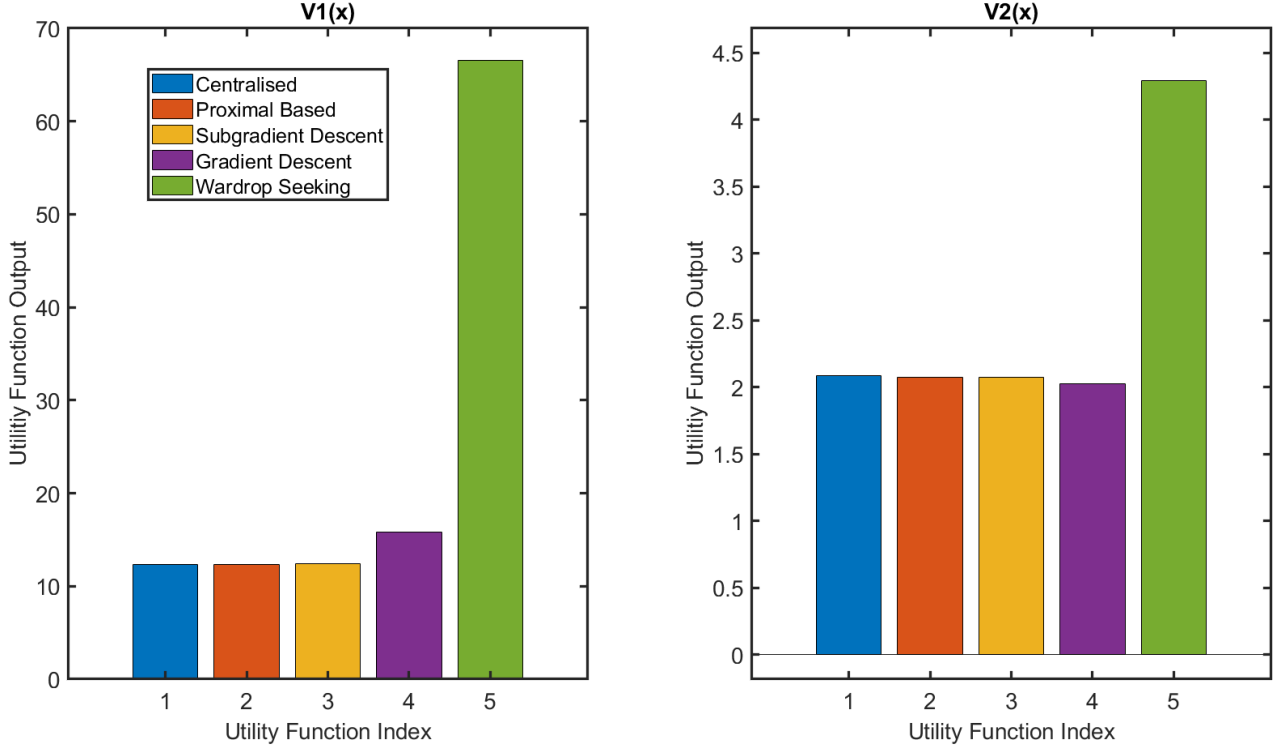
Then finally, we may be interested in how we can use the game formulation to approximate an entirely different cost which would be more interesting in the real world. To do this we will use a utility function which introduces a significantly higher cost if the total demand in a timeslot is larger than the grid can supply with renewables, this would be modelled in our game by having a sample with a large element in A_m to penalise agents. Now we model this explicitly in a utility function as follows

$$V_3(x) = \sum_{i \in \mathcal{N}} (x_i(A_0 \sigma(x_i, x_{-i}) + b_0)) + \|\delta(x)\|_1$$

$$\text{where } \delta(x)_n = \begin{cases} k, & \text{if } \sigma(x_i, x_{-i})_n \geq \rho_n \\ 0, & \text{otherwise.} \end{cases} \quad (5.3)$$

In (5.3) we use k to denote some arbitrary penalty for exceeding the threshold ρ (which is a vector consisting of the threshold at each timestep). The vector produced is then simply summed to produce the total penalty.

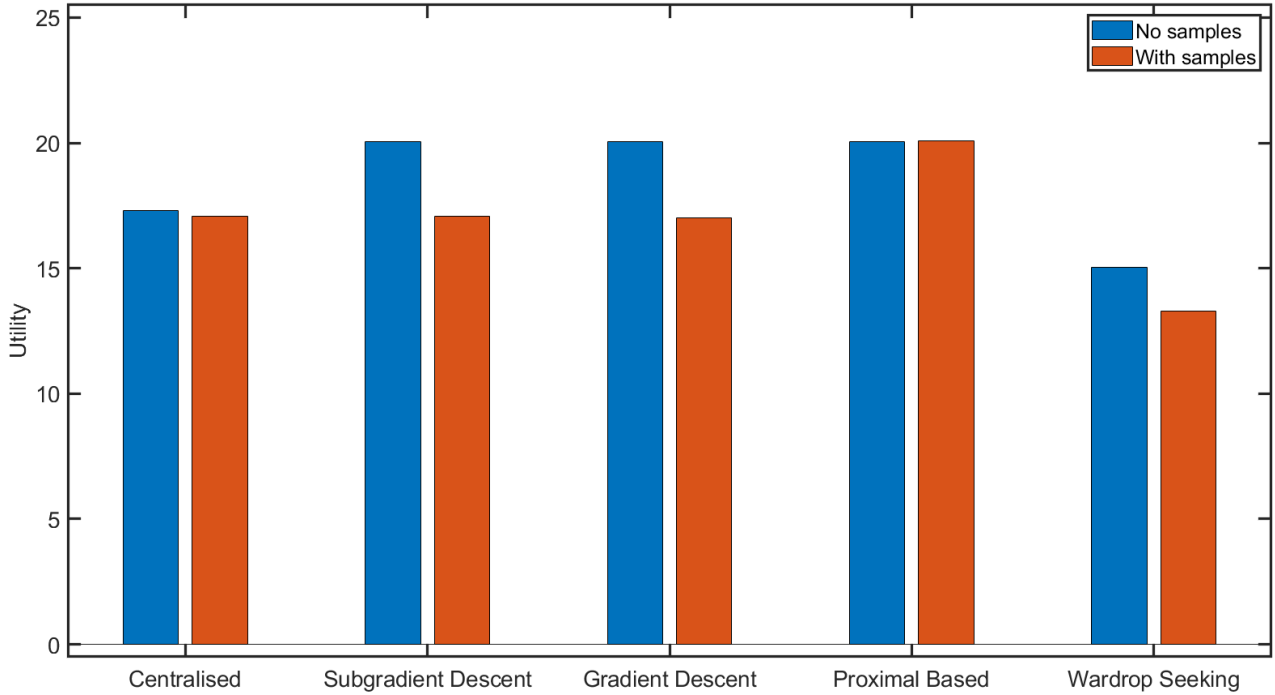
Figure 5.7: Comparison of two utility functions



In Figure 5.7 we can see the comparison of the utility obtained with $V_1(x)$ and $V_2(x)$, we will later look at how well $V_3(x)$ is satisfied by the game. These different utility functions allow us to gain some more insight into the algorithms, as well as offering up further motivation. Interestingly we see that the Wardrop equilibrium seeking algorithm has a utility significantly closer to the other functions when looking solely at $f_i(x_i, x_{-i})$, this is somewhat as expected since the wardrop is only optimising over $f_i(x_i, x_{-i})$. Perhaps more surprising is the performance of the gradient descent algorithm which performs worse than others by measure of $V_1(x)$ but better than the others according to $V_2(x)$, this might suggest that the reason this algorithm does not appear to perform well is that it tends to move more along the gradient of $f_i(x_i, x_{-i})$.

For $V_3(x)$ it is useful to compare the results of evaluating the utility following a game with no samples, compared to one with samples. The results of this can be seen in Figure 5.8, we can then immediately see that the introduction of samples helps to satisfy the usage threshold (hence leading to a lower cost). The cost difference between the bars is equal to k . We notice that almost all the algorithms are able to keep the total usage below ρ only through the introduction of these samples, offering an alternative interpretation of the samples as a method of directly affecting energy usage to keep it below some threshold (e.g. the capacity of renewables).

Figure 5.8: $V_3(x)$ utility with and without samples



We have therefore demonstrated the applications of a number of alternative utility functions, allowing for alternative methods to compare algorithms.

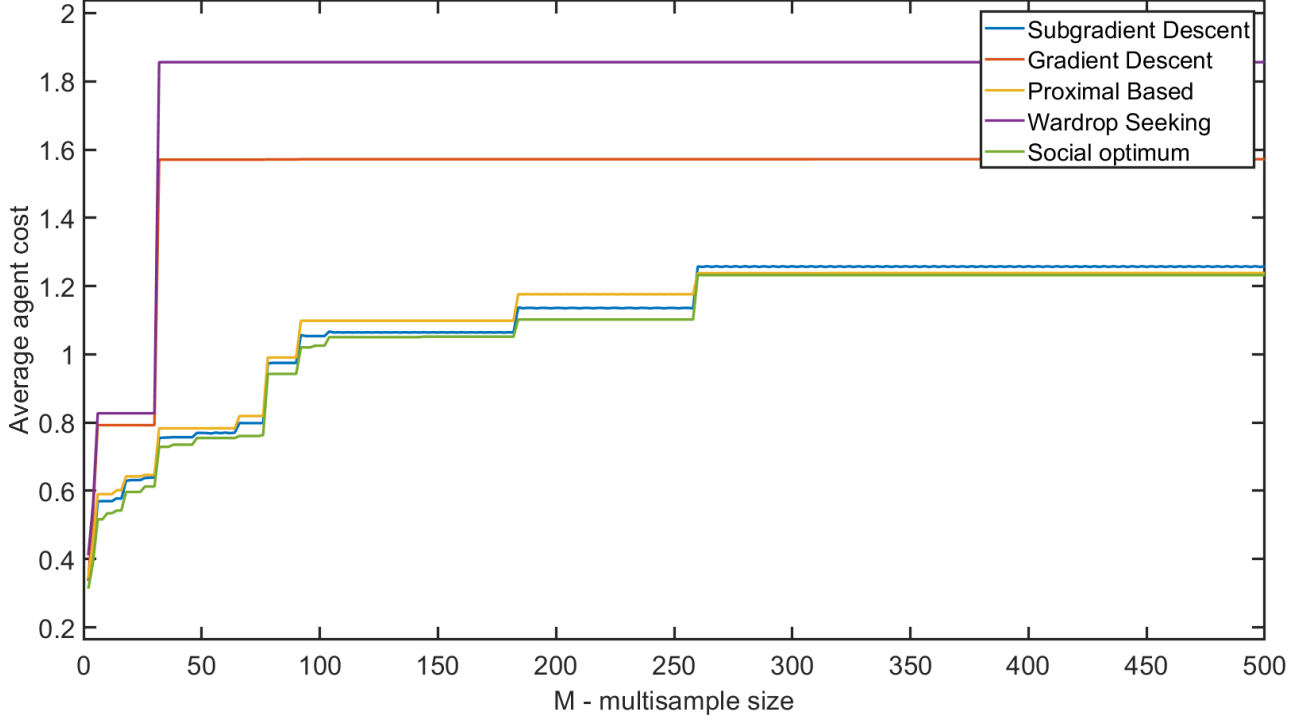
5.2.4 Incremental Scenario Addition

We now move on to look at how the game could be played in a more realistic scenario, starting with very few scenarios and incrementally adding them, viewed as uncovering a new scenario at the end of each day. We will look at the effect this has on agent's average cost each day.

We start with a multisample size of just two samples and then incrementally add two before running to convergence again. The results of this simulation are seen in Figure 5.9. We see that for the first 100 or so days (i.e the first 100 or so samples) the cost changes frequently, up until day 260 when no further changes occur. This behavior is exactly as we would predict since the probability that a new sample would violate our equilibrium decreases as more samples are added. It does provide some motivation for the real world scenario in that as the game is played out day by day there should occur some point at which the expected daily cost is unlikely to shift.

Some other interesting points to note in this graph are the behaviours of the gradient descent and Wardrop equilibrium seeking algorithms, in particular they both behave fairly similarly and are significantly more robust to new samples (although do behave very suboptimally). This suggests that the Wardrop equilibrium could be a more useful equilibrium if we wish to trade off cost for robustness.

Figure 5.9: Average agent cost as new scenarios are uncovered



We have therefore seen how the actual cost would change in a more realistic scenario with an increasing multisample size, and how the different algorithms respond to such a scenario.

5.3 Results for Non-homogenous Objectives

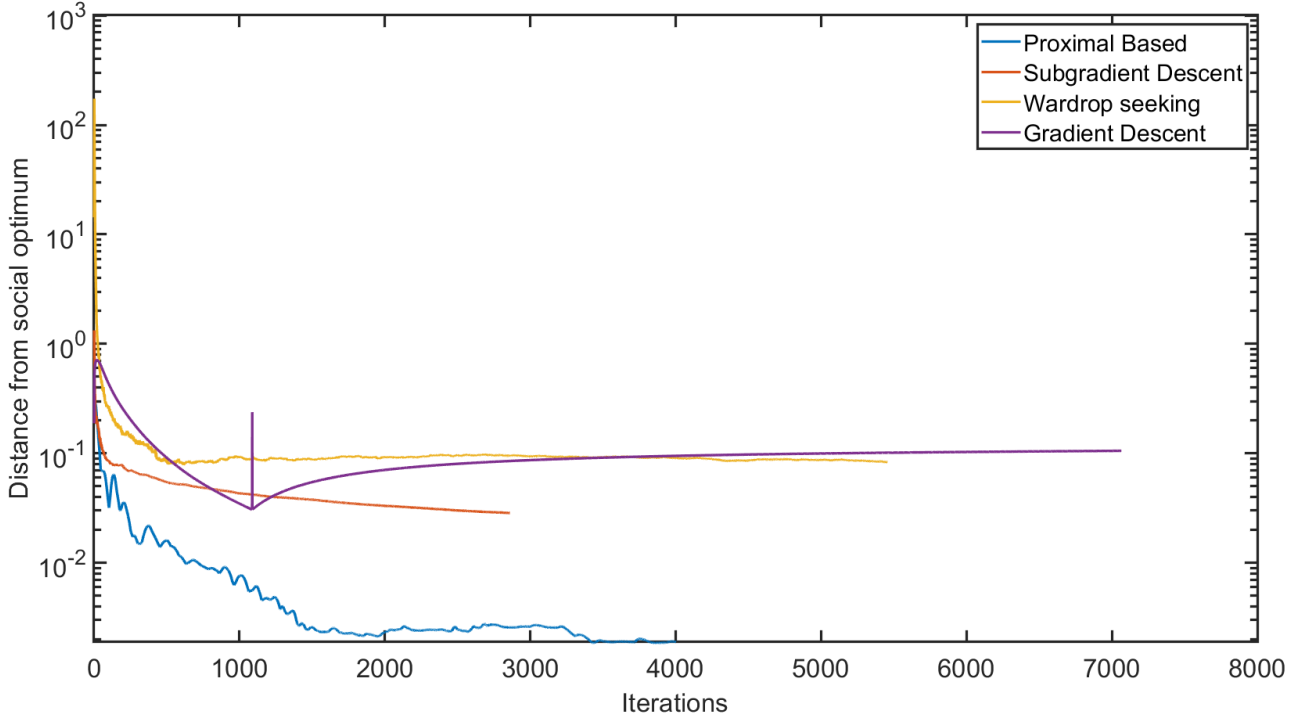
In this section we will now move on to look at $\hat{\mathcal{G}}$, this game now has a penalty function which is dependant on agents' individual schedules instead of just the total demand. We will again compare the available algorithms before looking at how we can discard scenarios, and finally seeing how this game behaves in a simulation where scenarios are uncovered incrementally.

5.3.1 Algorithm Comparison

In this section we will provide a comparison for the different algorithms, as well as checking that they do all converge in this new game. We start by looking at how they behave for a fixed number of agents over many iterations, and then their behaviour for increasing population sizes.

For the fixed population size, we used 10 agents and ran all the algorithms until their outer loop costs converged to within 10^{-6} . We then observe how their distance from the social optimum progressed with more iterations (we measure the distance here using the 1-norm of the difference in cost vectors divided by the population size, or equivalently the difference in the average agent cost).

Figure 5.10: Comparison of different algorithms for $\hat{\mathcal{G}}$



We can see from Figure 5.10 that all of the algorithms do appear to converge, the proximal based and Wardrop seeking algorithms did not converge to the limits specified but do both still appear to be converging (albeit very noisily in the case of the proximal based algorithm). We also notice that the gradient descent and Wardrop seeking algorithms both appear to converge to a similar point again. The runtimes and efficiency at convergence are shown in Table 5.2.

Table 5.2: Algorithm comparison

Algorithm	Time until termination (seconds)	Efficiency at convergence
Proximal based	7482 ²	1.0013
Gradient descent	1886	1.0722
Subgradient descent	315	1.0195
Wardrop equilibrium seeking	2278 ²	1.0567

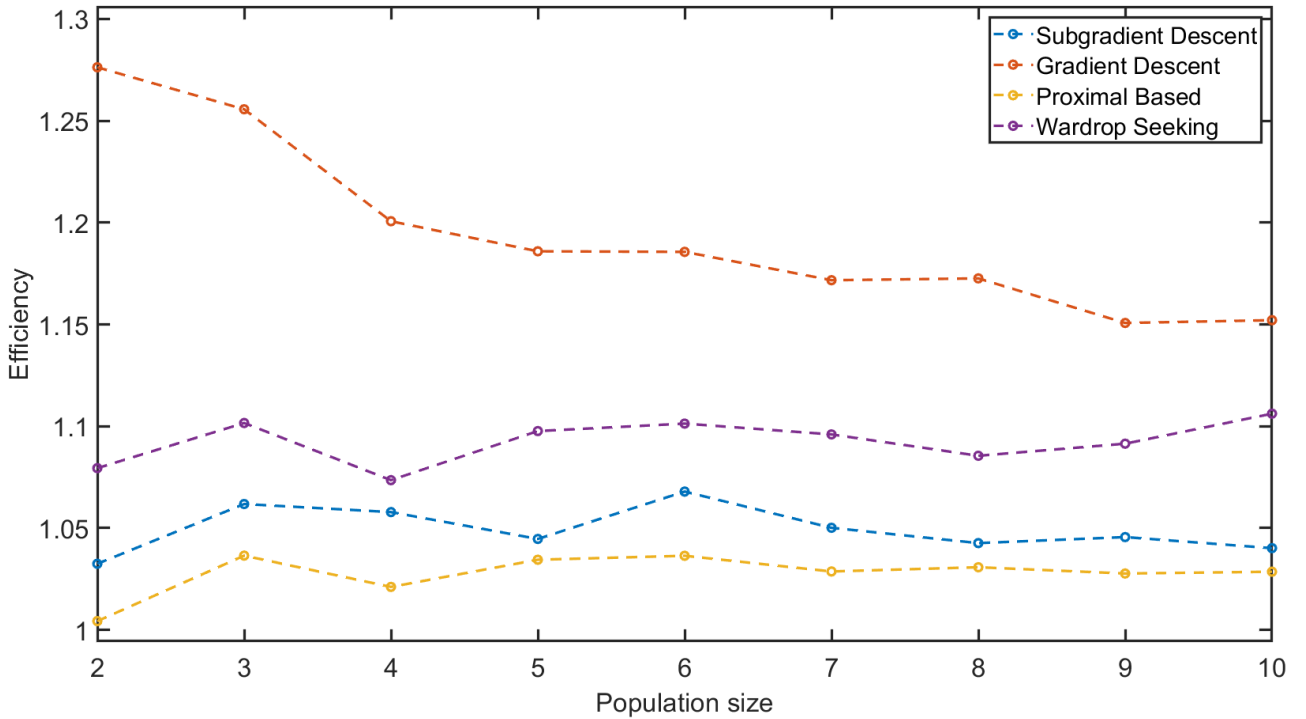
This table, in combination with Figure 5.10 allows us to start to see the different performances of the various algorithms. We can see from this table that although the subgradient descent algorithm has converged further away from the social optimum than the proximal based algorithm it has achieved this in less than one twentieth of the time, and six times as fast as the next fastest algorithm. This is because the subgradient descent algorithm is the only algorithm which does not perform any computationally

²Note the Proximal based and Wardrop equilibrium seeking algorithms were terminated before convergence.

expensive quadratic programs, instead performing a relatively simple subgradient step. This time difference is more significant here than in \mathcal{G} since now using y vectors to augment the game leads to an additional N quadratic programs (each with M variables) on each iteration.

Another interesting result comes from the Wardrop equilibrium seeking algorithm, this now terminates significantly closer to the social optimum as compared with the results after playing \mathcal{G} . This result may arise from the fact that the uncertain part of $\hat{\mathcal{G}}$ now has a term which is dependent on the individual agent strategy, as opposed to being solely reliant on the total strategy, this then means that the Wardrop equilibrium seeking algorithm does perform some optimisation over the samples (to do so we again augment the game with the addition of N y vectors).

Figure 5.11: Efficiency of different algorithms with increasing population size



We now look at how the algorithms respond to varying population sizes in order to gain some intuition as to how the algorithms' performances might be linked to population size. The results of running the algorithms with different population sizes are presented in Figure 5.11. Interestingly, all the algorithms seem to have a fairly constant efficiency regardless of population size (although this efficiency is dependent on the convergence check), the only exception to this is the gradient descent algorithm which appears to perform better the larger the population size is.

Furthermore, results are only available up to an agent population size of 10 since beyond this our centralised solver for computing the social optimum begins to fail. This does offer some further reasoning for optimising the game in a distributed manner, since $\hat{\mathcal{G}}$ quickly becomes too complex to

solve centrally.

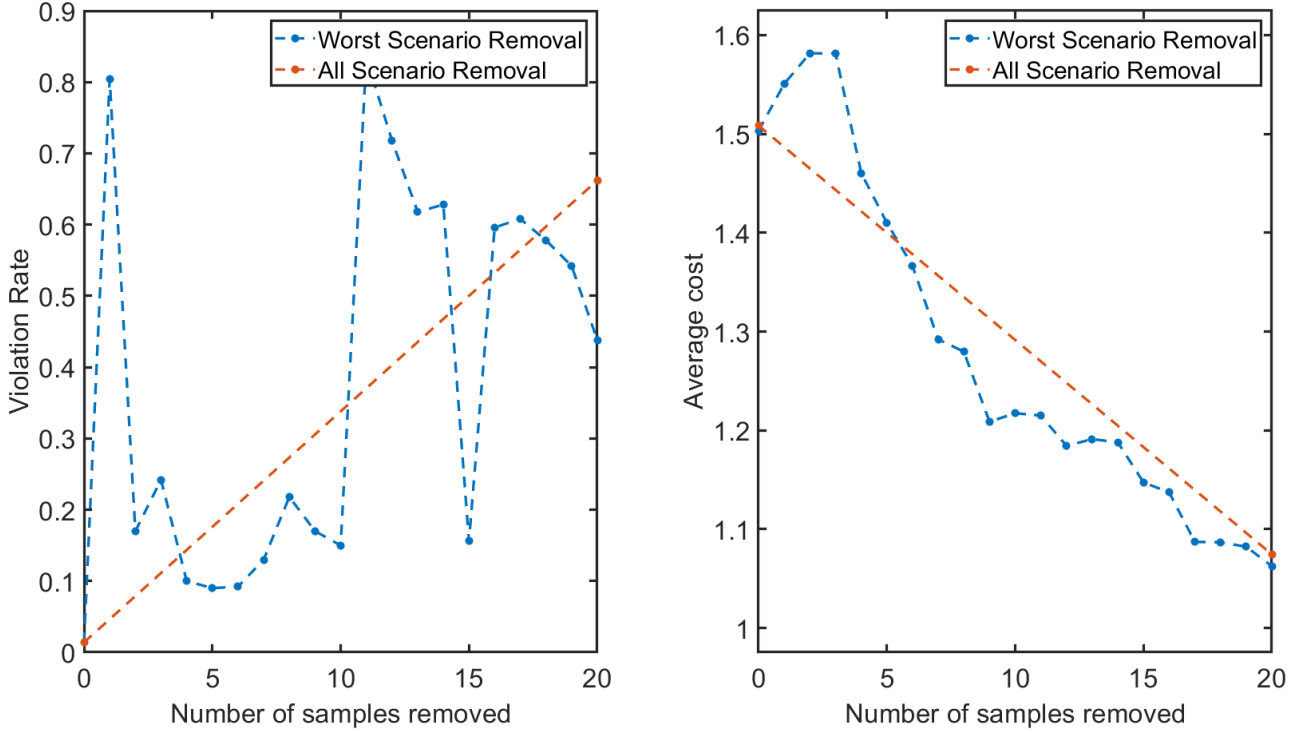
We have thus provided some brief comparison of the different algorithms available, and seen how they may behave differently in this game with different penalties for each agent.

5.3.2 Scenario Discarding

We will now once again move on to look at how we can discard samples from the M-multisample to improve agents' individual costs. Again providing the theoretical bounds on violation rate to check if they still hold for this game. These simulations then allow us to see how well the scenario-and-discarding approach carries over to our new game, as well as how useful our violation probability predictions are in this different game.

In order to empirically upper bound the violation a similar algorithm to Algorithm 8 will be used, with the minor addition that we will instead check the \hat{y} vector for each individual agent.

Figure 5.12: Comparison of sample removal algorithms for $\hat{\mathcal{G}}$



We can see from Figure 5.12 that once again both sample removal algorithms appear to behave similarly in terms of cost decrease. However, in this case the size of the compression set is significantly larger since we look at the union of all the samples which are active for each agent, therefore we can remove 20 samples immediately without any intermediate runs to find a new compression set. Interestingly, when removing the worst case scenario we do not achieve a monotonically decreasing cost as could be expected. One possible explanation for this is that, since agents are now receiving

an individual penalty, this worst case scenario is unlikely to be the worst case scenario for all agents, and in fact may not even be a support sample for some agents. Thus, when removing this sample it is possible that some agents actually see a cost increase which outweighs the cost decrease seen by other agents. This could motivate alternative discarding schemes, for example where agents are alternately selected to discard their own worst case scenario, however we leave this to future work. It should also be noted that the empirical upper bound on the violation rate we find now is significantly less useful, again since we need to look at \hat{y} then any new samples are much more likely to potentially lead to violation since they only need to appear in one agent's y vector. Because of this the bound obtained may be significantly looser than the bound in Figure 5.4, and we now look into this in more detail.

Figure 5.13: Comparison of violation rate prediction

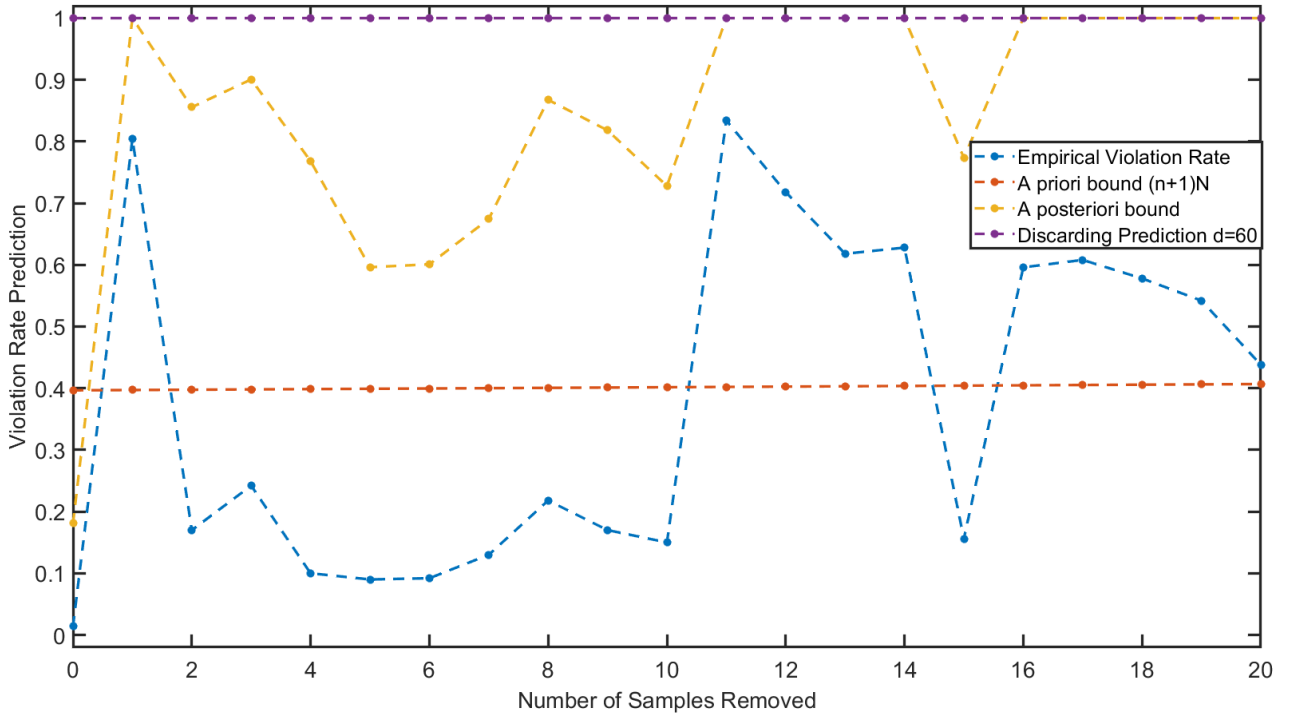
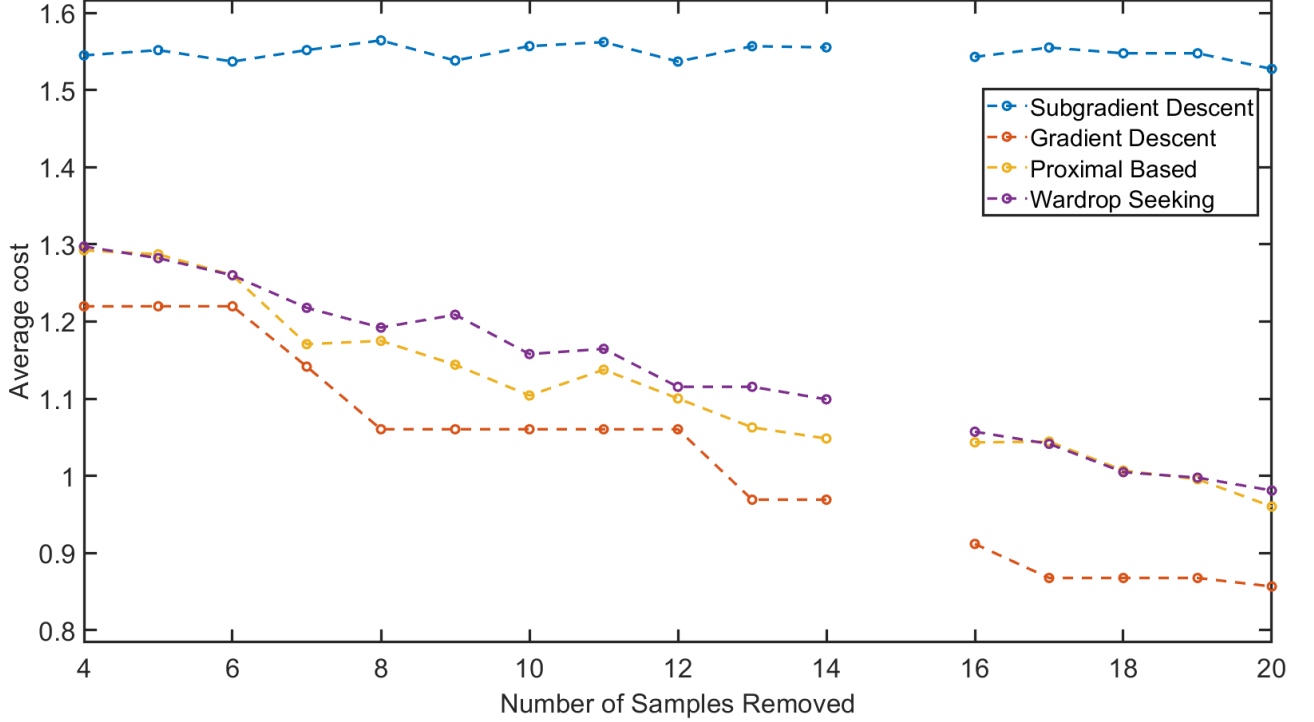


Figure 5.13 provides bounds for the violation rate similar to those in Figure 5.5. The a priori result we obtain does not appear to be of significant use, this could be expected since it does not account for the sample removal, and further the bound we use of $(n + 1)N$ relies on the assumption of non-degeneracy which may not necessarily hold true here, it is possible that this bound is satisfied and that our empirical violation rate is too loose and this may be an interesting avenue for further exploration. In particular, we find that the compression set cardinality found by checking \hat{y} is often significantly larger than $(n + 1)N$, resulting in the violation probability predicted by the a posteriori bound (making use of this compression set cardinality) being larger than that predicted by the a priori bound (using $(n + 1)N$). Also of little use is the violation rate prediction using (4.8), we used $d = 60$

for the number of decision variables this time and the resulting bound is incredibly loose, remaining at a trivial upper bound of 1 for any number of samples removed. Finally, the a posteriori bound which makes use of the predicted compression cardinality is in fact fairly reasonable, providing an upperbound on our empirical violation rate which holds and is non-trivial, unfortunately this does require some computation of the compression set cardinality but is nevertheless a very useful bound.

Figure 5.14: Comparison of different algorithms' cost improvements as samples are removed (gaps occur where the centralised solution has failed due to a timeout in an iteration)



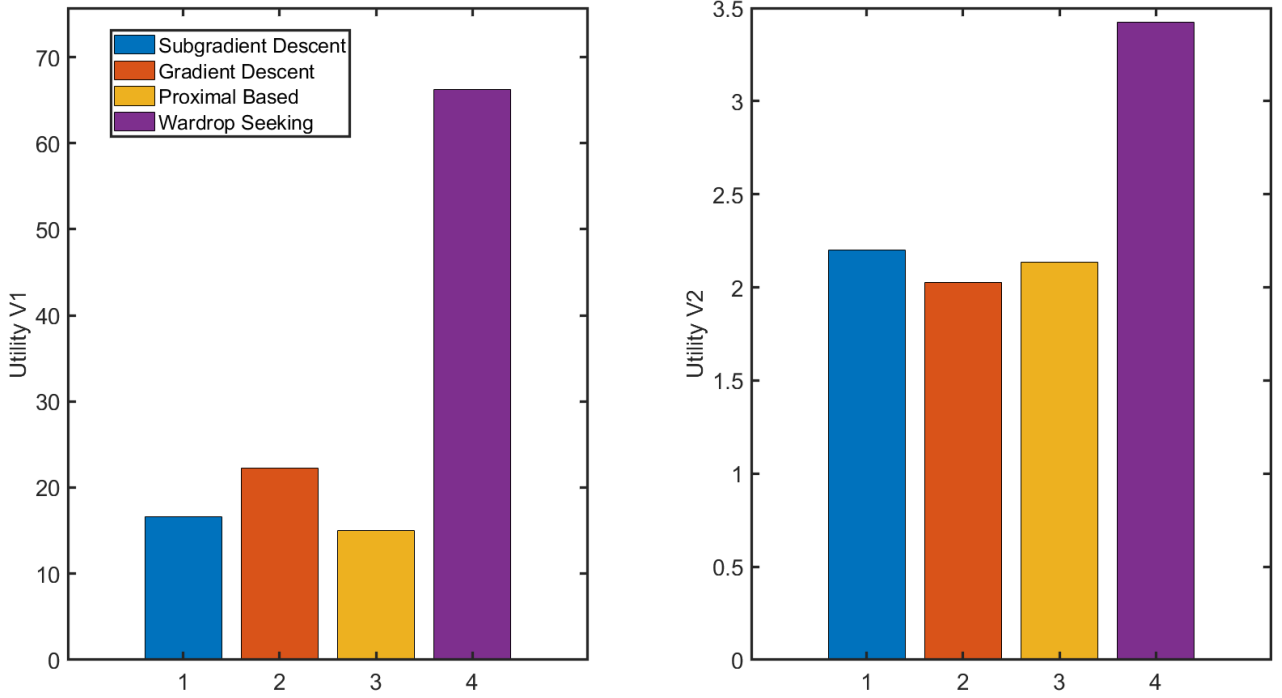
We can now see how different algorithms compare when samples are removed in Figure 5.14. Now that the Wardrop equilibrium seeking algorithm does include some optimisation of the scenarios we see that it behaves similarly to both other algorithms using the augmented game. In comparison, the subgradient descent algorithm is very ineffective in sample removal, since it has no implicit weighting of the different samples then it is more difficult to decide which samples to remove, evidently this has led to no real improvement in cost here.

This concludes our investigation into scenario removal for $\hat{\mathcal{G}}$ and we have seen that we are still able to offer some bounds on violation rate. We also noted that it is still possible to use the sampling-and-discarding approach for most algorithms, although deciding on which sample to remove is not obvious for the subgradient descent algorithm.

5.3.3 Alternative Cost Functions

We now move on to compare the different utility functions previously defined to see if the results found for $V_3(x)$ still hold, and again to gain a deeper insight into the operation of the algorithms by comparing $V_1(x)$ and $V_2(x)$.

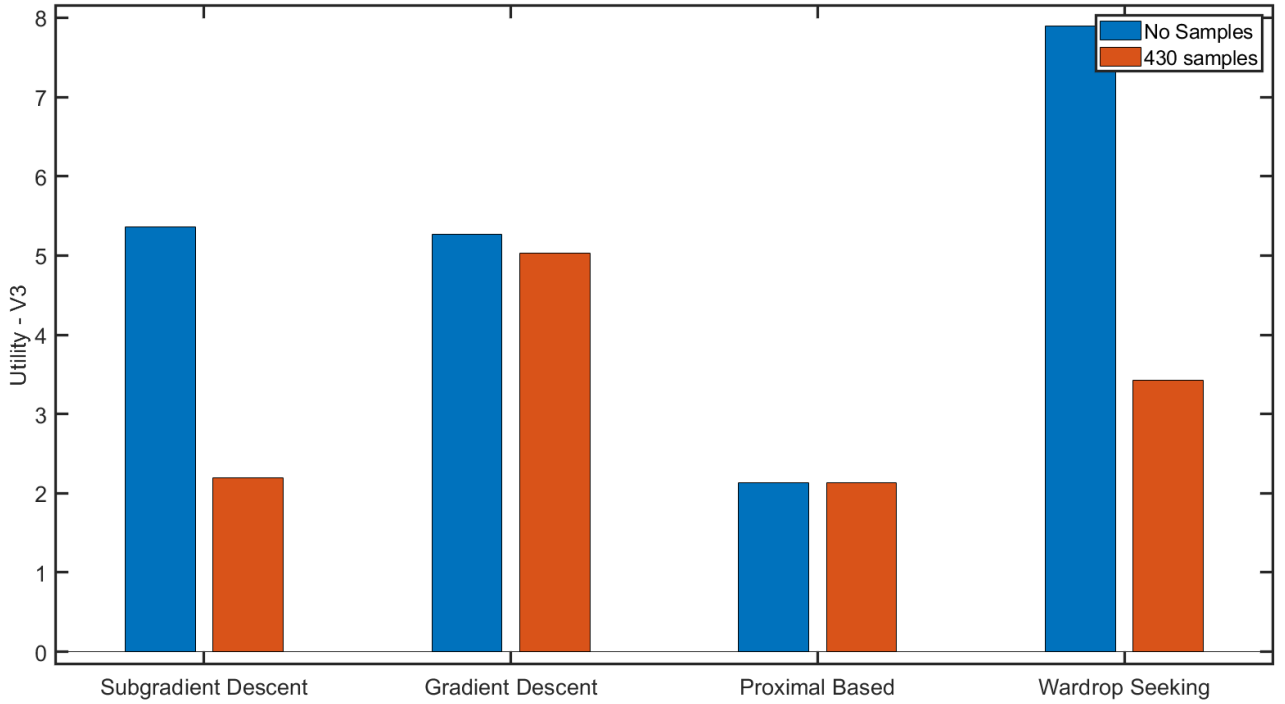
Figure 5.15: Comparison of two utility functions



In Figure 5.15 we can once again see that the Wardrop equilibrium seeking and gradient descent algorithms perform better when measured using $V_2(x)$. This suggests that even with the ability to optimise over the samples the Wardrop seeking equilibrium still slightly optimises more over the deterministic part of the cost, and similarly the gradient descent algorithm does not show a significant difference in behaviour with the introduction of a heterogeneous penalty.

We can also see from Figure 5.16 that the introduction of samples is generally able to match improvement to a more complex cost function $V_3(x)$. Once again the proximal based algorithm does not necessarily appear to follow this same improvement (and the gradient descent algorithm only shows slight improvement). Yet this does show some indication that the game is suitable to approximate a soft constraint on total usage.

Figure 5.16: $V_3(x)$ utility with and without samples



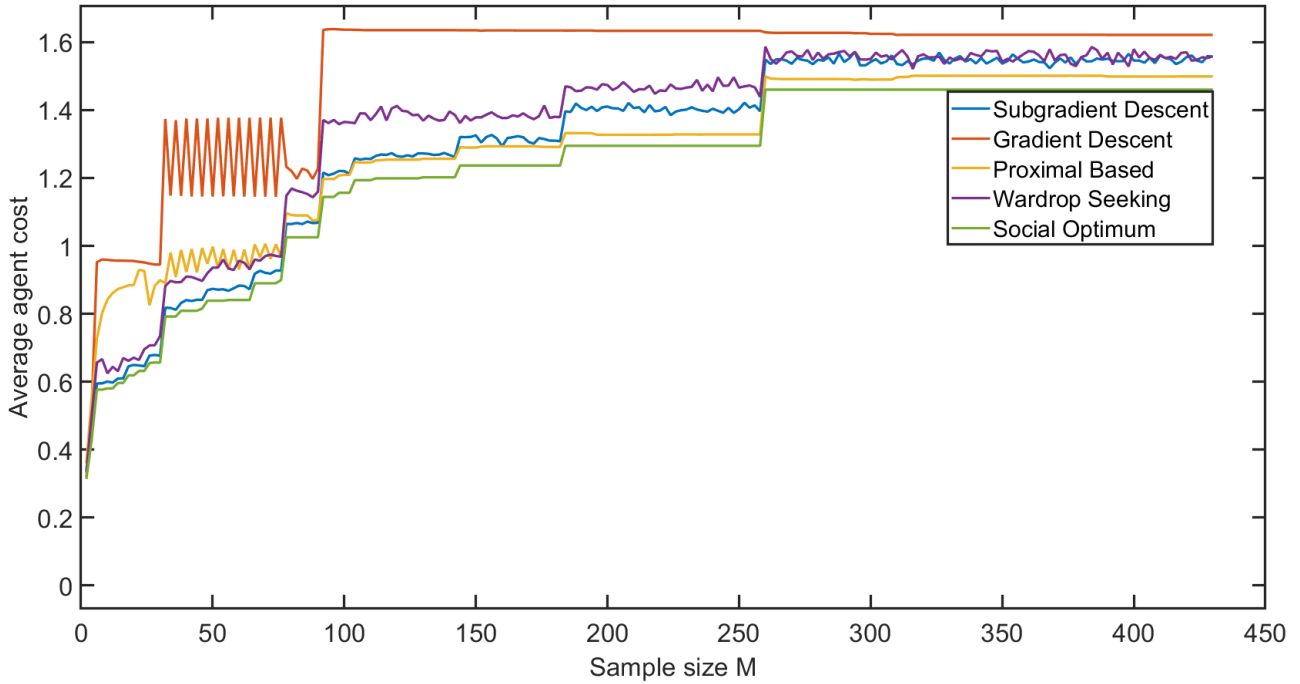
The results in this section show that, similar to in \mathcal{G} , we can use different utility functions to compare results and gain further insight into the operation of the various algorithms.

5.3.4 Incremental Scenario Addition

To conclude our numerical results we shall now investigate the behaviour of $\hat{\mathcal{G}}$ as samples are incrementally uncovered. This is a useful simulation to observe how the algorithms may behave in a realistic scenario, as well as allowing us to see how the addition of new samples affects equilibria.

We run the simulation almost identically to that of the previous section, with the minor change that we do not run to the full 500 samples since most algorithms take significantly longer to converge in this game. We have also used the same samples to run this so that a more direct comparison can be made with the results for \mathcal{G} .

Figure 5.17: Average agent cost as new scenarios are uncovered



From Figure 5.17 it is evident that the centralised game behaves quite similarly in terms of the locations and values of cost increase, once again reaching a reasonably steady value from around sample 260. The gradient descent algorithm once again reaches a steady state significantly before this. Interestingly, there are some slight decreases in the gradient descent cost as more samples are uncovered, perhaps because some samples now lead to a greater gradient up to the optimum. In any case, these cost improvements are relatively minor and so we can be reasonably confident that this is fairly robust to new samples. We can also see that the Wardrop and subgradient algorithms are very noisy and rarely seem to terminate in a constant position, this likely indicates that the convergence tolerances were not sufficient here. It is still possible to see where they jump as new samples affect the equilibria but both algorithms do jump around significantly more in this game than in our previous game. As might be expected we also see an increased average cost pretty much across the board, since we now take a maximum for each agent the cost will necessarily be greater than (or equal to) the cost when taking a maximum of the sum.

We have now briefly explored how algorithms react as scenarios are uncovered each day, with some contrast to how they were uncovered in a game with a common penalty term.

Chapter 6

Conclusion

In this chapter we will summarise our main results, including a comparison of the available algorithms, and then move on to provide some potential areas for future research. The code used to run the various simulations provided in Chapter 5 is available at <https://github.com/lukearcus/4YP>.

6.1 Summary

In this report we have introduced four algorithms for solving the non-cooperative games \mathcal{G} and $\hat{\mathcal{G}}$, these games are themselves reformulations of an electric vehicle charging problem. We have also investigated the possibility of applying a sampling-and-discarding approach to improve upon the costs found. The results presented in Chapter 5 provide some insight into how each of the various algorithms perform in different scenarios, and we will now have a look at the various advantages and disadvantages for each of the four algorithms.

Table 6.1: Subgradient Descent

Advantages	Disadvantages
<ul style="list-style-type: none">• Appears to tend toward the social optimum• Fastest algorithm per iteration• Works well on both games analysed• Central authority only required to aggregate schedules	<ul style="list-style-type: none">• No easy way to evaluate the compression set (and hence no easy to remove samples)• Performs worse per iteration than the proximal based algorithm

Table 6.2: Gradient Descent

Advantages	Disadvantages
<ul style="list-style-type: none"> • Straightforward computation of compression set • Generally more robust to new samples • Reasonably fast (only requires quadratic programs for y vectors) 	<ul style="list-style-type: none"> • Generally performs poorly compared to other algorithms • Slower than subgradient descent (especially for heterogeneous penalty term) • Takes many iterations to converge • Central authority required to aggregate schedules and optimise y vector(s)

Table 6.3: Proximal Based

Advantages	Disadvantages
<ul style="list-style-type: none"> • Performs better than any other algorithm per iteration • Easy to remove samples and evaluate a compression set • Has theoretical guarantees on convergence for \mathcal{G} 	<ul style="list-style-type: none"> • One of the slowest algorithms • Does not converge well in $\hat{\mathcal{G}}$ (although still performs best in terms of efficiency) • Requires a large number of quadratic programs on each iteration so is inherently slow • Central authority required to aggregate schedules and optimise y vector(s)

Table 6.4: Wardrop Equilibrium Seeking

Advantages	Disadvantages
<ul style="list-style-type: none"> • Does not require any information about samples for common penalty game • Can perform reasonably well with heterogeneous penalty term • Central authority only required to aggregate schedules in \mathcal{G} 	<ul style="list-style-type: none"> • Slowest to converge to a fixed threshold • Does not perform any optimisation over samples in \mathcal{G} • Can require many iterations to converge • Central authority required to aggregate schedules and optimise y vectors in $\hat{\mathcal{G}}$

We have also seen that the sampling-and-discarding method works well to further optimise our solutions, leading to a decrease in average agent cost. In taking this approach however, we do lose some robustness to new samples. This robustness is evaluated in the prediction of a violation rate (that is the probability that a new sample will violate our equilibrium) and we present a number of different methods for doing so. The a posteriori bound presented in [4] appears to be the best bound available in general but the underlying assumptions may mean that it could fail as a large number of samples are removed. The bound provided in [26] seems to be reasonably conservative to hold almost always, but we found that using the full number of optimisation variables lead to a bound that was too loose. Finally, we have found that the best method for removing samples is to remove the entire compression set (where possible) since this method does not appear to have any significant detriments in terms of the amount the cost is decreased by nor in terms of the increase in violation rate.

6.2 Future Work

There are a wide range of avenues available for future work. Firstly, we have found empirically that our algorithms converge for $\hat{\mathcal{G}}$, however we do not have theoretical proofs of this convergence for any of our algorithms since such proofs lie beyond the scope of our paper. However, this could be a very interesting area of future research. Further theoretical proofs concerning the results obtained by the subgradient descent algorithm could also be very useful since it appears that, at least in the case of the game with a common penalty, this algorithm can reach the social optimum (certainly with enough iterations it moves closer to the social optimum than the Nash equilibrium uncovered by the proximal

based algorithm).

Another interesting area of research concerns the prospect of dishonest agents who could lie about their charging intentions in order to obtain a better individual cost at the time of charging (likely at the detriment of other agents). Preliminary investigations indicate that if an agent was to lie about their usage plans they could very slightly improve upon their cost at the time of charging (although this required a request for energy of approximately 2 orders of magnitude larger than any agent would use in any timeslot usually). Further work to investigate if this is possible with a more intelligent agent could be beneficial in developing methods to prevent such dishonest agents.

Furthermore, in our work we have generally assumed agents are only selfish to the extent allowed by the various algorithms. Further considerations for truly selfish agents who simply play a best response at each timestep to fully optimise with respect to their objective could be interesting. One such setup could consider the effects of increasing numbers of truly selfish agents on a population who are otherwise playing according to one of the algorithms identified.

Finally, work in developing tighter bounds on the violation rate when samples are removed could also prove beneficial in developing strategies which allow for a better optimum. In addition, work investigating how a game in which a sample is removed whenever a new sample is added to the multisample (i.e. on the violation rate and average agent cost) could be beneficial for practical scenarios, maintaining a fixed multisample size whilst also uncovering new samples each day.

Bibliography

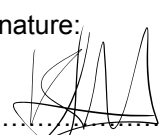
- [1] Z. Ma, D. S. Callaway, and I. A. Hiskens. Decentralized charging control of large populations of plug-in electric vehicles. *IEEE Transactions on Control Systems Technology*, 21(1):67–78, 2013.
- [2] F. Parise, M. Colombino, S. Grammatico, and J. Lygeros. Mean field constrained charging policy for large populations of Plug-in Electric Vehicles. *Proceedings of the IEEE Conference on Decision and Control*, 2015-February(February):5101–5106, 2014.
- [3] L. Deori, K. Margellos, and M. Prandini. On the connection between Nash equilibria and social optima in electric vehicle charging control games. *IFAC-PapersOnLine*, 50(1):14320–14325, 2017.
- [4] F. Fele and K. Margellos. Probably approximately correct nash equilibrium learning. *arXiv*, 2019.
- [5] D. Paccagnan, B. Gentile, F. Parise, M. Kamgarpour, and J. Lygeros. Nash and Wardrop Equilibria in Aggregative Games with Coupling Constraints. *IEEE Transactions on Automatic Control*, 64(4):1373–1388, 2019.
- [6] J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, jan 1950.
- [7] T. Baar and G. Olsder. *Dynamic Noncooperative Game Theory, 2nd Edition*. Society for Industrial and Applied Mathematics, 1998.
- [8] J. G. Wardrop. Road paper. some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers*, 1(3):325–362, 1952.
- [9] P. Couchman, B. Kouvaritakis, M. Cannon, and F. Prashad. Gaming strategy for electric power with random demand. *IEEE Transactions on Power Systems*, 20(3):1283–1292, 2005.
- [10] V. V. Singh, O. Jouini, and A. Lisser. Existence of Nash equilibrium for chance-constrained games. *Operations Research Letters*, 44(5):640–644, 2016.

- [11] J. Pang, S. Sen, and U. Shanbhag. Two-stage non-cooperative games with risk-averse players. *Mathematical Programming*, 165, 04 2017.
- [12] U. Ravat and U. Shanbhag. On the characterization of solution sets of smooth and nonsmooth convex stochastic nash games. *SIAM Journal on Optimization*, 21:1168–1199, 07 2011.
- [13] J. Koshal, A. Nedic, and U. V. Shanbhag. Regularized iterative stochastic approximation methods for stochastic variational inequality problems. *IEEE Transactions on Automatic Control*, 58(3):594–609, 2013.
- [14] H. Xu and D. Zhang. Stochastic Nash equilibrium problems: Sample average approximation and applications. *Computational Optimization and Applications*, 55(3):597–645, 2013.
- [15] C. Yu, M. van der Schaar, and A. H. Sayed. Distributed learning for stochastic generalized nash equilibrium problems. *IEEE Transactions on Signal Processing*, 65(15):3893–3908, 2017.
- [16] J. Lei and U. V. Shanbhag. A randomized inexact proximal best-response scheme for potential stochastic nash games. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1646–1651, 2017.
- [17] M. Aghassi and D. Bertsimas. Robust game theory. *Mathematical Programming*, 107(1-2):231–273, 2006.
- [18] R. Nishimura, S. Hayashi, and M. Fukushima. Robust nash equilibria in n-person non-cooperative games: Uniqueness and reformulation. 5, 05 2009.
- [19] J. Zazo, S. Zazo, and S. Valcarcel Macua. Robust worst-case analysis of demand-side management in smart grids. *IEEE Transactions on Smart Grid*, 8:1–1, 04 2016.
- [20] M. Hu and M. Fukushima. Existence, uniqueness, and computation of robust nash equilibria in a class of multi-leader-follower games. *SIAM Journal on Optimization*, 23(2):894–916, 2013.
- [21] S. Floyd and M. Warmuth. Sample Compression, Learnability, and the Vapnik-Chervonenkis Dimension. *Machine Learning*, 21(3):269–304, 1995.
- [22] T. Alamo, R. Tempo, and E. F. Camacho. Randomized strategies for probabilistic solutions of uncertain feasibility and optimization problems. *IEEE Transactions on Automatic Control*, 54(11):2545–2559, 2009.

- [23] K. Margellos, M. Prandini, and J. Lygeros. On the connection between compression learning and scenario based single-stage and cascading optimization problems. *IEEE Transactions on Automatic Control*, 60(10):2716–2721, 2015.
- [24] G. C. Calafiore and M. C. Campi. The scenario approach to robust control design. *IEEE Transactions on Automatic Control*, 51(5):742–753, 2006.
- [25] M. C. Campi, S. Garatti, and F. A. Ramponi. A General Scenario Theory for Nonconvex Optimization and Decision Making. *IEEE Transactions on Automatic Control*, 63(12):4067–4078, 2018.
- [26] M. C. Campi and S. Garatti. A Sampling-and-Discarding Approach to Chance-Constrained Optimization: Feasibility and Optimality. *Journal of Optimization Theory and Applications*, 148(2):257–280, 2011.
- [27] National Grid Historical Demand Data. <https://www.nationalgrideso.com/data-explorer>. Accessed: 2021-04-11.
- [28] G. Scutari, F. Facchinei, J. S. Pang, and D. P. Palomar. Real and complex monotone communication games. *IEEE Transactions on Information Theory*, 60(7):4197–4231, 2014.
- [29] M. Zinkevich. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. *Proceedings, Twentieth International Conference on Machine Learning*, 2:928–935, 2003.
- [30] F. Facchinei and J. Pang. *Finite-dimensional variational inequalities and complementarity problems*. Springer Science & Business Media, 2007.
- [31] N. Z. Shor. *Minimization Methods for Non-Differentiable Functions*. Springer-Verlag Berlin Heidelberg, 1985.
- [32] M. C. Campi and S. Garatti. The exact feasibility of randomized solutions of uncertain convex programs. *SIAM Journal on Optimization*, 19(3):1211–1230, 2008.
- [33] M. C. Campi and S. Garatti. Wait-and-judge scenario optimization. *Mathematical Programming*, 167, 07 2016.
- [34] M. C. Campi and S. Garatti. *Introduction to the Scenario Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2018.

Factor	Answer	Things to Consider	Record details here
Has the checklist covered all the problems that may arise from working with the VDU?	<input checked="" type="checkbox"/> Yes <input type="checkbox"/> No		
Are you free from experiencing any fatigue, stress, discomfort or other symptoms which you attribute to working with the VDU or work environment?	<input checked="" type="checkbox"/> Yes <input type="checkbox"/> No	Any aches, pains or sensory loss (tingling or pins and needles) in your neck, back shoulders or upper limbs. Do you experience restricted joint movement, impaired finger movements, grip or other disability, temporary or permanently	
Do you take adequate breaks when working at the VDU?	<input checked="" type="checkbox"/> Yes <input type="checkbox"/> No	Periods of two minutes looking away from the screen taken every 20 minutes and longer periods every 2 hours Natural breaks for taking a drink and moving around the office answering the phone etc.	
How many hours per day do you spend working with this computer?	<input type="checkbox"/> 1-2 <input type="checkbox"/> 3-4 <input checked="" type="checkbox"/> 5-7 <input type="checkbox"/> 8 or more		
How many days per week do you spend working with this computer?	<input type="checkbox"/> 1-2 <input type="checkbox"/> 3-5 <input checked="" type="checkbox"/> 6-7		
Please describe your typical computer usage pattern	I typically will spend the mornings using my laptop to watch any pre-recorded lectures, and afternoon doing coding and/or virtual classes		

Student Declaration and Academic Approval

<p><u>Student Declaration:</u></p> <p>I have completed the DSE Workstation Checklist and the Supplementary Questions for my computer-related risk assessment for 4YP Project Number indicated below:</p> <p>4YP Project Number: <u>12207</u></p> <p>4YP Student's Name (please print) <u>LUKE RICKARD</u></p> <p>4YP Student's Signature: <u>L Rickard</u></p>	<p><u>Academic Approval</u></p> <p>I confirm my approval of this 4YP DSE Risk Assessment.</p> <p>Academic Supervisor's Name: (please print)</p> <p><u>Kostas Margellos</u></p> <p>Academic Supervisor's Signature: </p>
--	--