Optimal scheduling of distributed generation to achieve linear aggregate response

Paul Beagon, Miguel D. Bustamante, Mel T. Devine, Susan Fennell, Jonathan Grant-Peters, Cameron Hall, Róisín Hill, Taulant Kerci, Gary O'Keefe

This problem was presented by Captured Carbon at the 141st European Study Group with Industry
University College Dublin, Ireland, 25-29 June 2018

Abstract

Captured Carbon supply power to Eirgrid by using many small power generators (Single Generators) as if their combined power were a unique large power station (Virtual Generator). When Eirgrid order power from a power station, they assume that power output increases linearly over a period of time (ramp up time), after which the station generates power at full capacity. The main question we tackle is: How can we control the start up time of many small generators in order to get as close as possible to a linear combined ramp behaviour? In this report, we provide 5 different approaches (gradient descent, mixed integer linear programming, bin packing, systematic adjustment and ramp tracking simulation) to the problem using sample data from 20 single generators provided by Captured Carbon. We also discuss briefly the "equity" challenge, consisting of the possibility to have optimal solutions for which not always the same single generator starts first. We conclude with some suggestions for future work.

1. Introduction

In Ireland, the electricity transmission system operator is Eirgrid. They are responsible for ensuring that electricity is transferred from generators to consumers in addition to forecasting energy demand in order to ensure that power meets demand. In recent years, the uncertainty associated with renewable sources such as wind and solar have made it difficult to ensure demand balances with supply. Consequently, Eirgrid co-ordinate with generators and instruct them to increase or decrease power output as required.

To ensure such a balance, Eirgrid seek electricity consumers, typically industrial, who can provide power back to the grid. Consumers may do so in two ways: either through their own micro-generators or by reducing their demand. Captured Carbon are an energy service provider that provide electricity to the grid by coordinating a number of such consumer sites (henceforth known as sites). As a result, Captured Carbon are considered as a virtual

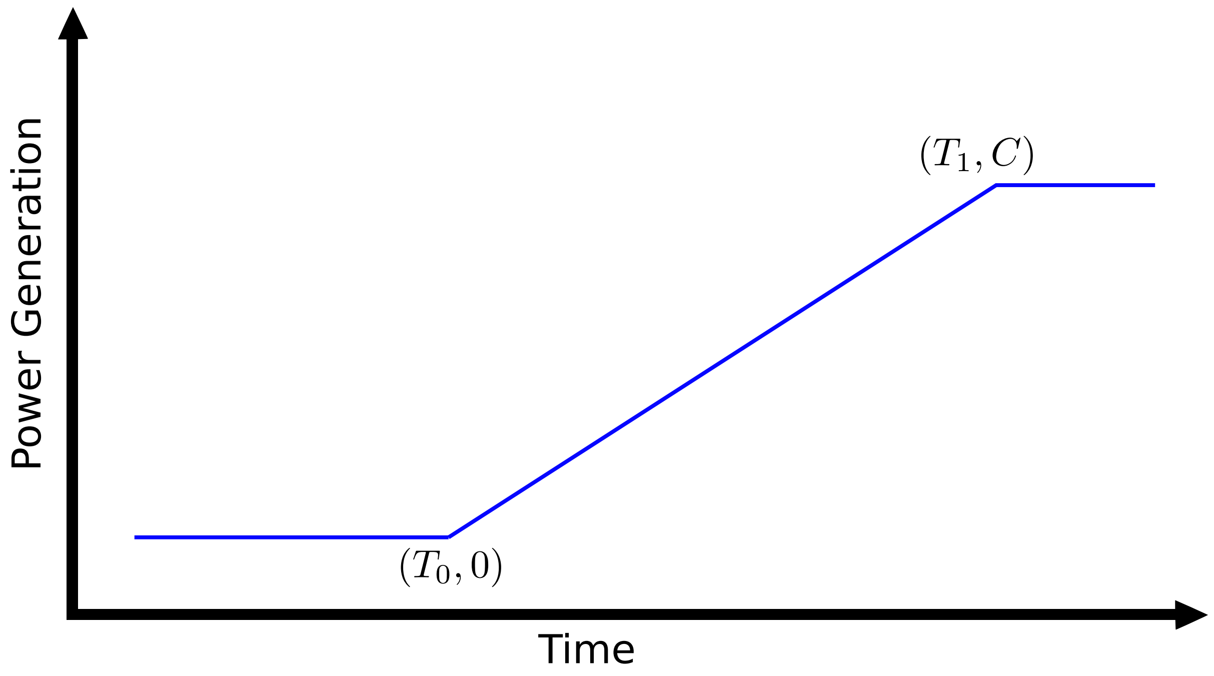


Figure 1: The above plot shows the production of a Power Plant when Eirgrid instructs it to change state from minimum production to maximum production.

generator or a virtual power plant.

When operating the grid, Eirgird require information on how each generator (real or virtual) will operate and assume that each generator behaves like a single generator. For conventional power plants (e.g., gas or coal-fired) this is not a problem. However, for virtual generators made out of multiple sites such as Captured Carbon, this can be challenging.

Single generators operate in one of three states: minimum production, maximum production, and ramping production. When the generator operates in either minimum or maximum production state, it produces energy at a constant rate. Defining the power produced as the rate of change of the energy produced, the power produced when operating at minimum production is 0 MW and the power produced when operating at maximum production is C MW, where C is known as the capacity of the generator. Now, when the generator operates in the ramping production state, the behaviour is slightly different. In this case, the power produced is not a constant; rather, it increases linearly with time, so *the rate of change of the power produced* is constant: this rate is known as the *ramping rate*. In Figure 1, the generator enters this state at time T_0.

For Captured Carbon, each of their generating sites has different characteristics. For example, some sites may have a capacity (i.e., maximum power production) of 0.1 MW while some could have a capacity of 5 MW. In addition, ramping rates may vary between sites. For each site, the ramping rate is known, fixed and linear, as in Figure 1. However, because each site is different, Captured Carbon's aggregated ramping rate may be non-linear; see Section 1.2 for further details.

Herein lies the challenge for Captured Carbon: because Eirgird expect Captured Carbon to behave like a single generator with a linear ramping rate, Captured Carbon's non-linear ramping rate means there are times when they provide excess power, compared with what Eirgird expect them to provide. In these cases, Captured Carbon do not receive any payments for the excess power. Moreover, there are also times when Captured Carbon provide insufficient power and consequently face fines for the shortfall.

However, when coordinating the consumer sites, Captured Carbon have flexibility in when to instruct each site to start providing power. This provides them an opportunity to minimise their excess power and fines. Captured Carbon posed the following task to the 141st European Study Group with Industry (ESGI141):

- Find ways to schedule Captured Carbon's sites such that their aggregate ramping rate is as close to linear as possible.

This report details how the participants of ESGI141 tackled the problem. Captured Carbon provided the group with sample data for 20 sites, each with different characteristics (see Appendix A). A number of different methods were considered:

1. Direct Methods:
    - Gradient Descent
    - Mixed Integer Linear Programming
2. Heuristic Methods:
    - Systematic Adjustment
    - Bin Packing (with simulated annealing)
    - Simulated Annealing on the direct problem
    - Ramp Tracking Simulation

The report is structured as follows: the Introduction continues with a description of notation and objective functions proposed. Section 2 describes each of the different methodologies: the Gradient Descent Method in Section 2.1, the Mixed Integer Linear Programming Method in Section 2.2, the Systematic Adjustment Method in Section 2.3, the Bin Packing Method in Section 2.4, Simulate Annealing directly applied to the original problem in Section 2.5, and the Ramp Tracking Simulation Method in Section 2.6. Finally, Section 3 provides conclusions and ideas for future work.

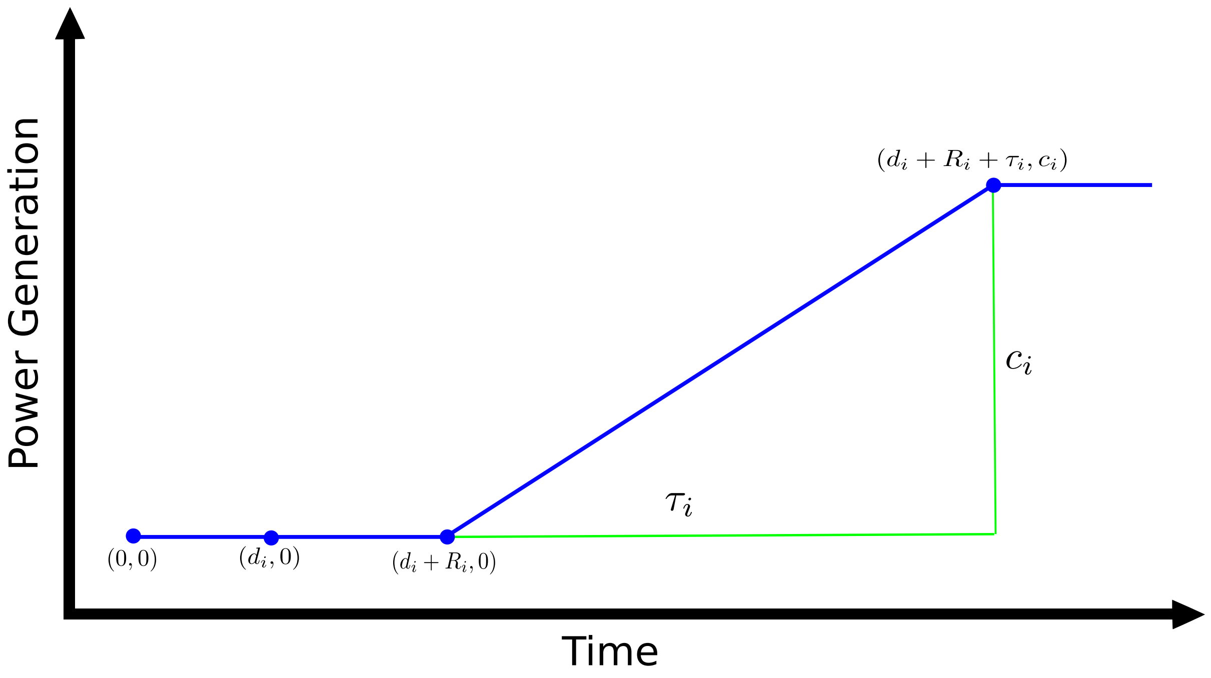


Figure 2: The above plot shows the production of a single small power generator. The qualitative behaviour is the same as that of a Power Plant.

1.1. Notation for a single site

We now describe notation for the sites that Captured Carbon control in order to change their effective power production. Consider Figure 2. This plot shows the energy produced by a single site i when instructed to transition from the state of minimum production to the state of maximum production.

Qualitatively, Figures 1 and 2 are the same. We use the quantitative distinction between these plots as an opportunity to introduce some notation. In Figure 2, which plots power production p versus time t, there are four points (t,p) of interest: $(0,0),(d_i,0),(d_i+R_i,0)$ and $(d_i+R_i+\tau_i,C_i)$. We explain the significance of these points below:

- $(0,0)$: Time 0 refers to the time at which Eirgrid instruct Captured Carbon to transition towards a state of maximum production. At this time Captured Carbon are in a state of minimum production which corresponds to 0 MW being produced.
- $(d_i,0)$: The time d_i refers to how long Captured Carbon delay before passing on to the i-th site the instruction to transition towards a state of maximum production. As no instructions have been given to the site thus far, production is still at 0 MW.
- $(d_i+R_i,0)$: For each site, there is a time frame which it requires in order to respond to the instruction from Captured Carbon. We call this the response time and denote

it by R_i. The time $d_i + R_i$ refers to the time at which the i-th site transitions from a state of minimum production to ramping production. At this instant production levels are still 0 MW as the site has still been in a state of minimum production thus far.

- $(d_i+R_i+\tau_i, C_i)$: For each site, there is a period of time required during which the power production grows linearly in time (i.e. the rate of change of the power production is constant), going from 0 MW to its maximum possible value C_i MW (C_i is the so-called capacity of the i-th site). By definition, throughout this time the site is in ramping production state. We call this time τ_i. Once this time is completed, the site's power production continues to be C_i MW, its full capacity. The time $d_i + R_i + \tau_i$ is the time at which the i-th site transitions from the state of ramping production to the state of maximum production.

Given that C is the maximum output of the virtual power plant which Captured Carbon represent, and C_i is the maximum output of each of their individual sites, it follows that $C = \sum_{i=1}^{n} C_i$, where n is the total number of sites.

1.2. Behaviour of a collection of sites

Let n refer to the number of sites which Captured Carbon have at their disposal. Further define the *ramping function* f by:

$$f(x) := \begin{cases} 0 & x < 0, \\ x & 0 \leq x < 1, \\ 1 & 1 < x. \end{cases} \tag{1}$$

With f defined, we can write an expression for $P(t)$, the power generated by a single Power Plant at time t, and $F(\boldsymbol{d}, t)$, the total power generated by all of Captured Carbon's sites at time t given delays $\boldsymbol{d}$.

$$P(T_0, T_1, t) = Cf\left(\frac{t - T_0}{T_1 - T_0}\right). \tag{2}$$

$$F(\boldsymbol{d}, t) := \sum_{i=1}^{n} C_i f\left(\frac{t - d_i - R_i}{\tau_i}\right). \tag{3}$$

We define also the difference between these:

$$E(\boldsymbol{d}, T_0, T_1, t) := F(\boldsymbol{d}, t) - P(T_0, T_1, t) \tag{4}$$

The function F is piecewise linear. It further has the properties that if $t < \min\limits_{1\leq i\leq n}(d_i + R_i)$,

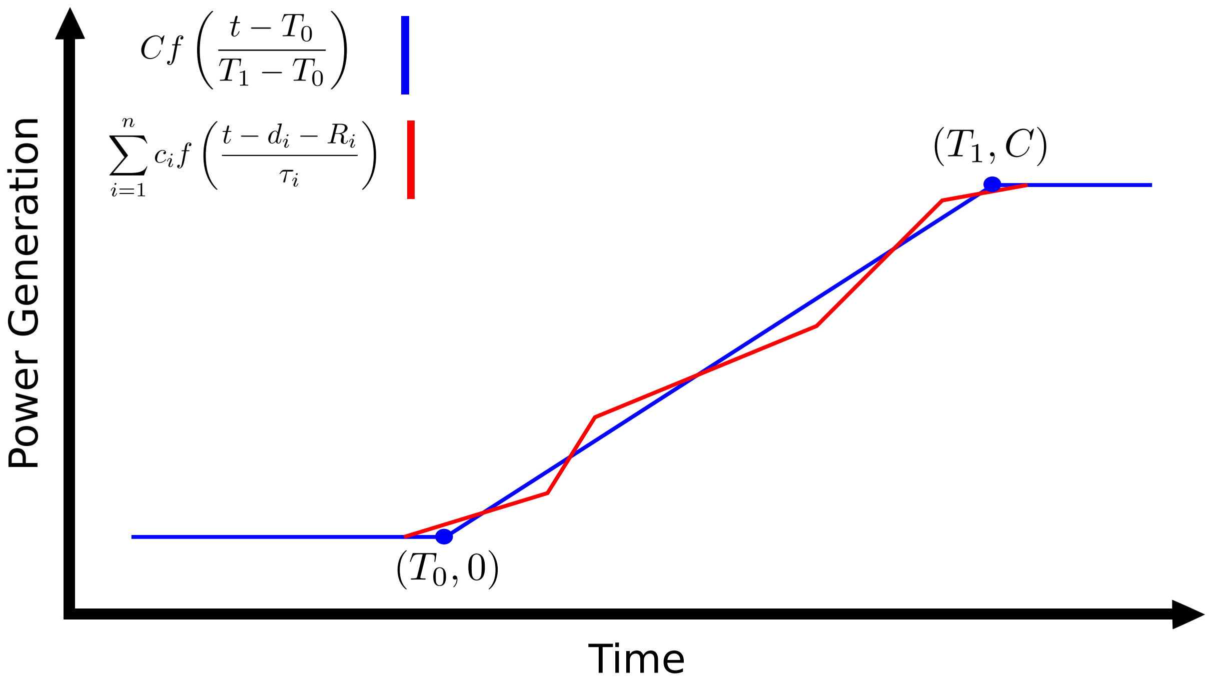


Figure 3: We plot above the production of both a single large Power Plant, and a collection of small generators which have the same maximum output capability.

then $F(\boldsymbol{d}, t) = 0$ and if $t > \max_{1\le i\le n}(d_i + R_i + \tau_i)$, then $F(\boldsymbol{d}, t) = C$. In so far as this, F, the total output of all our small generators, behaves in the same fashion as P, the output of a single large generator. However, in the region $\min_{1\le i\le n}(d_i + R_i) < t < \max_{1\le i\le n}(d_i + R_i + \tau_i)$, F is a piecewise linear function, instead of a linear function like P.

1.3. Objective functions

How do Captured Carbon make money? Eirgrid pay them an amount based on the power generated by the virtual Power Plant which Captured Carbon emulate. That is, revenue depends on $P(T_0, T_1, t)$ and is independent of $F(\boldsymbol{d}, t)$. Given this, the means by which Captured Carbon maximise profits is by minimising costs. There are three sources of costs at play here:

- Fines. If $F(\boldsymbol{d}, t) < P(T_0, T_1, t)$, then Captured Carbon are not delivering the power to Eirgrid that they are contractually obliged to provide. When this happens they are liable to be fined.
- Unpaid Power. Captured Carbon are paid for generating exactly $P(T_0, T_1, t)$ at time t. If $F(\boldsymbol{d}, t) > P(T_0, T_1, t)$, then they are supplying free energy to the grid.
- Costs of sites. Each site that Captured Carbon coordinates has a cost per unit of energy associated with it. For site i, we denote this cost by γ_i. It is more efficient for

Captured Carbon to activate cheap generators first in order to reduce this cost. The accumulated cost at a late enough reference time T_{ref} (i.e., such that $T_{\text{ref}} > d_i + R_i + \tau_i$) of running a site i whose power production started ramping at time $t = d_i + R_i$ until reaching full capacity C_i at time $t = d_i + R_i + \tau_i$, is equal to γ_i times the area under the power production curve, namely

$$Cost_i = \gamma_i C_i \left(\frac{1}{2}\tau_i + T_{\text{ref}} - d_i - R_i - \tau_i \right) = \gamma_i C_i \left(T_{\text{ref}} - d_i - R_i - \frac{1}{2}\tau_i \right) . \tag{5}$$

With these sources of costs in mind, we identify the following objective functions.

$$J_1(\boldsymbol{d}, T_0, T_1) := \int\limits_{V_0} |E(\boldsymbol{d}, T_0, T_1, t)| dt \tag{6}$$

$$J_2(\boldsymbol{d}, T_0, T_1) := \int\limits_{V_0} G\left(E(\boldsymbol{d}, T_0, T_1, t)\right) dt \tag{7}$$

$$J_3(\boldsymbol{d}, T_0, T_1) := \sum_{i=1}^{n} Cost_i - K \int\limits_{V_0} E(\boldsymbol{d}, T_0, T_1, t)\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right) dt \tag{8}$$

where $E(\boldsymbol{d}, T_0, T_1, t) = F(\boldsymbol{d}, t) - P(T_0, T_1, t)$, $G(x) = x\,\Theta(x) - K\,x\,\Theta(-x)$ with Θ being the Heaviside function ($\Theta(x) = 1$ if $x > 0$ and $\Theta(x) = 0$ if $x < 0$), and K is a positive constant representing the fines. The costs $Cost_i$ are defined in equation (5). Finally, V_0 is an appropriately chosen time interval in which the integrand is nonzero. Notice that we know the integrand is non-zero on the following finite closed interval:

$$\left[\min\left(T_0, \min_i(d_i + R_i)\right), \max\left(T_1, \max_i(d_i + R_i + \tau_i)\right)\right] ,$$

so we must choose V_0 so that $\left[\min\left(T_0, \min_i(d_i + R_i)\right), \max\left(T_1, \max_i(d_i + R_i + \tau_i)\right)\right] \subseteq V_0$. For example, if we impose the constraints $T_0 < \min_i(d_i + R_i)$ and $T_1 \geq \max_i(d_i + R_i + \tau_i)$, then we may use the interval $V_0 = [T_0, T_1]$.

A further simplification is to be noted. From the fact that the cost $Cost_i$ is linear in the variable d_i, we can replace J_3 with the following equivalent functional:

$$\widetilde{J}_3(\boldsymbol{d}, T_0, T_1) := -\sum_{i=1}^{n} \gamma_i C_i d_i - K \int\limits_{V_0} E(\boldsymbol{d}, T_0, T_1, t)\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right) dt\,. \tag{9}$$

This differs from the actual total cost by a constant that depends on the parameters of the problem and the late reference time only:

$$\widetilde{J}_3(\boldsymbol{d}, T_0, T_1) = J_3(\boldsymbol{d}, T_0, T_1) - \sum_{i=1}^{n} \gamma_i C_i \left(T_{\text{ref}} - R_i - \frac{1}{2}\tau_i \right) .$$

1.4. Practical computation of the objective function

At first glance, the objective functions shown in Equations (6), (7), (8) and (9) seem computationally daunting due to the integration required. However, we first note that both $F(\boldsymbol{d}, t)$ and $P(T_0, T_1, t)$ are piecewise linear functions, which implies that so is their difference. Moreover, we know the points at which each of these functions are nonsmooth. $F(\boldsymbol{d}, t)$ is nonsmooth at $t = d_i + R_i$ and at $t = d_i + R_i + \tau_i$ for any i, while $P(T_0, T_1, t)$ is nonsmooth at $t = T_0$ and $t = T_1$.

Therefore, all we need to do in order to compute these integrals exactly is to find the set of points at which the integrand is nonsmooth. Then we use these points as our grid for the Trapezoidal Rule, which then computes the integral exactly.

1.5. Modelling Options

The values of the constants $\boldsymbol{C}, \boldsymbol{R}, \boldsymbol{\tau}$ and $\boldsymbol{\gamma}$ depend on the qualities of the generators which Captured Carbon have access to. However, the values of T_0 and T_1 are more difficult to define. They may be treated as constants, but unless Eirgrid require that their power plants start up in a single precise time, then we forfeit a degree of freedom unnecessarily. On the other hand, there does exist a constraint on T_1 in that Eirgrid expect Captured Carbon to reach full capacity at a finite time after they have been notified.

Some options as to how to deal with T_0 and T_1 include:

- Treat them both as constants. One appropriate choice of constants (leading to a minimum total time to full power, and a minimal amount of time spent in the ramping process) is $T_0 = \max_i(R_i + \tau_i) - \max_i(\tau_i)$ and $T_1 = \max_i(R_i + \tau_i)$.
- Define $T_0 = \min_i(d_i + R_i)$ and $T_1 = \max_i(d_i + R_i + \tau_i)$.
- Let both T_0 and T_1 be variables such that T_1 is bounded from above.
- Treat T_0 as a constant and let T_1 be variable.

2. Methods

The following tables summarise the notation to be used in the methods described in this Section.

$i \in I$	Sites/generators
$t \in \tilde{T}$	time
$n \in N$	iterations

Table 1: Indices and sets.

C_i	Maximum generation capacity at site i
C	Total capacity of all sites i
R_i	Response time for site i
τ_i	Ramp time for site i
γ_i	Cost per unit energy associated with site i
$\overline{\gamma}$	Mean cost
K	Multipler describing fines

Table 2: Parameters (model inputs).

$Cost_i$	Cost of site i
$J_1(\boldsymbol{d}), J_2(\boldsymbol{d}), J_3(\boldsymbol{d}), \widetilde{J}_3(\boldsymbol{d})$	Objective functions
$f(x)$	Ramping function
$F(\boldsymbol{d}, t)$	Total power generated
$E(\boldsymbol{d}, t)$	Error function
$P(T_0, T_1, t)$	Virtual power generated
$\Theta(x)$	Heaviside function
$G(x)$	Function describing asymmetric error
$\delta(x)$	Dirac delta function
$\delta_{i,j}$	Kronecker delta function
$U(x)$	PDF of uniform distribution
$\|\|x\|\|_l$	l-norm

Table 3: Functions.

T_0	time when ramping begins
T_1	time when ramping stops
d_i	delay time for site i
$\boldsymbol{d}$	vector of delay times
$gen_{i,t}$	generation from site i at time t
$below_t$	distance below target curve at time t
$above_t$	distance above target curve at time t
$b_{i,t}$	binary variable indicating whether or not site i is generating at time t
$\overline{b_{i,t}}$	binary variable indicating whether or not site i is at maximum capacity at time t
κ	Gradient descent multiplier
κ_n	Gradient descent step value
α_n	Simulated annealing step value

Table 4: Variables (model outputs).

2.1. Gradient Descent Method

2.1.1. Objective Functions

The gradient descent method is a classical method that can be applied to optimise an objective function provided this objective function is twice differentiable with respect to the independent variables, and provided the minima of the objective function are local.

In our problem, the independent variables are the n delay times $d_i,\ i = 1, \ldots, n$ along with the initial and final times T_0, T_1. We will focus our analyses on two of the objective functions: $J_2(\boldsymbol{d}, T_0, T_1)$ and $\widetilde{J}_3(\boldsymbol{d}, T_0, T_1)$, defined in equations (7) and (9) respectively, which we summarise below:

$$J_2(\boldsymbol{d}, T_0, T_1) = \int\limits_{V_0} E(\boldsymbol{d}, T_0, T_1, t) \left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right] dt\,,$$

$$\widetilde{J}_3(\boldsymbol{d}, T_0, T_1) = -\sum_{i=1}^{n} \gamma_i C_i d_i - K \int\limits_{V_0} E(\boldsymbol{d}, T_0, T_1, t)\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right) dt\,,$$

where

$$E(\boldsymbol{d}, T_0, T_1, t) = \sum_{i=1}^{n} C_i f\left(\frac{t - d_i - R_i}{\tau_i}\right) - Cf\left(\frac{t - T_0}{T_1 - T_0}\right), \tag{10}$$

where f is the ramping function defined in equation (1), and V_0 can be chosen as $V_0 = \left[\min\left(T_0, \min_i(d_i + R_i)\right), \max\left(T_1, \max_i(d_i + R_i + \tau_i)\right)\right]$.

As explained in the Introduction, the integrand is a piecewise linear function of time. This, along with the fact that the dependence of the integrand on the variables $\boldsymbol{d}$, T_0 and T_1 is via a parameterisation within the ramping function f, implies that it is possible to calculate explicitly derivatives of the objective function. We will be interested in calculating the gradient of each of the objective functions $J_2(\boldsymbol{d}, T_0, T_1)$ and $\widetilde{J}_3(\boldsymbol{d}, T_0, T_1)$, namely the set of partial derivatives of these functions with respect to the independent variables.

Objective Function $J_2(\boldsymbol{d}, T_0, T_1)$. Assuming without loss of generality that T_0 is fixed, we now calculate $\frac{\partial}{\partial d_i} J_2(\boldsymbol{d}, T_0, T_1),\ i = 1, \ldots, n$ and $\frac{\partial}{\partial T_1} J_2(\boldsymbol{d}, T_0, T_1)$. We have, first,

$$\frac{\partial}{\partial d_j} J_2(\boldsymbol{d}, T_0, T_1) = \int\limits_{V_0} \left[\frac{\partial}{\partial d_j} E(\boldsymbol{d}, T_0, T_1, t)\right] G'\left(E(\boldsymbol{d}, T_0, T_1, t)\right) dt\,, \quad j = 1, \ldots, n\,,$$

where the derivatives of the time integration limits with respect to d_j do not contribute because $E(\boldsymbol{d}, T_0, T_1, t) = 0$ at those times. Now, using formula (10) for E we readily obtain

$$\frac{\partial}{\partial d_j} E(\boldsymbol{d}, T_0, T_1, t) = -\frac{C_j}{\tau_j} f'\left(\frac{t - d_j - R_j}{\tau_j}\right) ,$$

and notice that $f'(x) = 1$ for $0 < x < 1$ and $f'(x) = 0$ otherwise. Also, we have explicitly $G'(x) = \Theta(x) - K\Theta(-x)$. Thus we obtain

$$\frac{\partial}{\partial d_j} J_2(\boldsymbol{d}, T_0, T_1) = -\frac{C_j}{\tau_j} \int_{d_j+R_j}^{d_j+R_j+\tau_j} \left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right] dt , \quad j = 1, \ldots, n . \tag{11}$$

Second, we calculate the derivative of the objective function with respect to T_1. In a similar way to the previous derivatives we obtain:

$$\frac{\partial}{\partial T_1} J_2(\boldsymbol{d}, T_0, T_1) = \int\limits_{V_0} \left[\frac{\partial}{\partial T_1} E(\boldsymbol{d}, T_0, T_1, t)\right] G'\left(E(\boldsymbol{d}, T_0, T_1, t)\right) dt .$$

Using the above formula (10) for E we obtain

$$\frac{\partial}{\partial T_1} E(\boldsymbol{d}, T_0, T_1, t) = C \frac{t - T_0}{(T_1 - T_0)^2} f'\left(\frac{t - T_0}{T_1 - T_0}\right) .$$

Thus, using the previously discussed property of $f'(x)$ we get:

$$\frac{\partial}{\partial T_1} J_2(\boldsymbol{d}, T_0, T_1) = \frac{C}{(T_1 - T_0)^2} \int_{T_0}^{T_1} (t - T_0) \left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right] dt . \tag{12}$$

Objective Function $\widetilde{J}_3(\boldsymbol{d}, T_0, T_1)$. As for the gradient of the objective function $\widetilde{J}_3(\boldsymbol{d}, T_0, T_1)$, the calculation is quite similar. We omit the derivation and simply provide the results for the components of the gradient:

$$\frac{\partial}{\partial d_j} \widetilde{J}_3(\boldsymbol{d}, T_0, T_1) = -\gamma_j C_j + \frac{K C_j}{\tau_j} \int_{d_j+R_j}^{d_j+R_j+\tau_j} \Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right) dt , \quad j = 1, \ldots, n \tag{13}$$

and

$$\frac{\partial}{\partial T_1} \widetilde{J}_3(\boldsymbol{d}, T_0, T_1) = -\frac{K C}{(T_1 - T_0)^2} \int_{T_0}^{T_1} (t - T_0) \Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right) dt . \tag{14}$$

2.1.2. Local Extrema

A local extremum of a given objective function $J(d_1,\ldots,d_n,T_1)$ is a choice of the $n+1$ variables $(d_1,\ldots,d_n,T_1)$ such that the $n+1$ equations $\frac{\partial}{\partial d_j}J(\boldsymbol{d},T_1)=0$, $j=1,\ldots,n$ and $\frac{\partial}{\partial T_1}J(\boldsymbol{d},T_1)=0$ are satisfied simultaneously. There may be many local extrema, i.e. many different choices of these variables $(d_1,\ldots,d_n,T_1)$ so that the equations are satisfied. Sometimes, however, there are no local extrema because the objective function takes minimum values at a kind of boundary in the $(d_1,\ldots,d_n,T_1)$ space. One expects, for the type of problems we are dealing with, that the relevant local extrema are so-called local minima, i.e. that the objective function evaluated at points that are close to the chosen point $(d_1,\ldots,d_n,T_1)$ is larger than the objective function evaluated at the chosen point. Notice that the actual value of the objective function is of interest, as different local minima will provide different values for the objective functions at the minima, and *we are interested in finding low values of the objective function because that is how the company can make savings.* It is the goal of all of the methods presented in this report to find these local minima for the chosen objective function in a quick way and to find the lowest minimum within a given search range.

For the objective function $\widetilde{J}_3(\boldsymbol{d},T_0,T_1)$, we have a problem though. It is possible to show that its minima are not local. To see this, look at the equation $\frac{\partial}{\partial T_1}\widetilde{J}_3(\boldsymbol{d},T_0,T_1)=0$:

$$-\frac{KC}{(T_1-T_0)^2}\int_{T_0}^{T_1}(t-T_0)\Theta\left(-E(\boldsymbol{d},T_0,T_1,t)\right)dt=0\,.$$

The left-hand side is sign-definite: the only way it is zero is if $E(\boldsymbol{d},T_0,T_1,t)\geq 0$ for all $t\in V_0$. In other words, there can be no fines and the actual power production curve lies strictly above the linear virtual power plant ramping curve. This gives us, however, a nice start: assuming $(d_1,\ldots,d_n)$ are given, one can choose the time T_1 to be the shortest one so that $E(\boldsymbol{d},T_0,T_1,t)\geq 0$ for all $t\in V_0$: in graphical terms, this implies that the actual power production curve touches the linear virtual power plant ramping curve at least at one point. In any case, the fact that $E(\boldsymbol{d},T_0,T_1,t)\geq 0$ for all $t\in V_0$ implies that the other components of the gradient are given by:

$$\frac{\partial}{\partial d_j}\widetilde{J}_3(\boldsymbol{d},T_0,T_1)=-\gamma_j C_j,\qquad j=1,\ldots,n\,,$$

which are all non-zero. In conclusion, there are no local extrema for this problem. To apply the method in this case, therefore, one has to change the strategy by for example fixing T_1 and considering the objective function as a function of $\boldsymbol{d}$ only. In such a case, only equations (13) are needed in order to determine the extrema so we would have the following n equations for the n unknowns $(d_1, \ldots, d_n)$:

$$\int_{d_j+R_j}^{d_j+R_j+\tau_j} \Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right) dt = \frac{\gamma_j \tau_j}{K}\,, \quad j = 1, \ldots, n\,,$$

which implies that there must be fines (there are times at which the actual power production curve goes below the linear virtual power plant ramping curve), and these fines are bounded from above.

For the objective function $J_2(\boldsymbol{d}, T_0, T_1)$ as a function of the $n+1$ variables $(d_1, \ldots, d_n, T_1)$ we have a different situation. From equations (11), (12), the local minima solve the system

$$\begin{aligned}
\int_{d_j+R_j}^{d_j+R_j+\tau_j} \left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right] dt &= 0\,, \quad j = 1, \ldots, n\,, \\
\int_{T_0}^{T_1} (t - T_0)\left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right] dt &= 0\,.
\end{aligned}$$

In contrast to the previous case, these equations imply a balance between contributions coming from the time intervals where $E > 0$ and the time intervals where $E < 0$. Therefore, there must be fines as well as over-production: the actual power production curve must cross the linear virtual power plant ramping curve at least in one point.

2.1.3. Gradient Descent Algorithm: Barzilai-Borwein method

The Barzilai-Borwein gradient descent method (Barzilai and Borwein, 1988) for a given objective function $J(\boldsymbol{x})$ (here $\boldsymbol{x} = \boldsymbol{d}$ in the case $J = \widetilde{J}_3$ and $\boldsymbol{x} = (\boldsymbol{d}, T_1)$ in the case $J = J_2$) consists of starting from a seed point $\boldsymbol{x} = \boldsymbol{x}^{(0)}$ and a second point $\boldsymbol{x} = \boldsymbol{x}^{(1)}$ in the space of variables, and iteratively going from there along the directions of steepest descent of the objective function, until reaching a local minimum (or a boundary). The directions of steepest descent are obtained by calculating the gradient of the objective function, namely the vector ∇J with components $\left(\frac{\partial}{\partial d_1} J(\boldsymbol{d}, T_0, T_1), \ldots, \frac{\partial}{\partial d_n} J(\boldsymbol{d}, T_0, T_1), \frac{\partial}{\partial T_1} J(\boldsymbol{d}, T_0, T_1)\right)$. We define iteratively a sequence of points by the 2-point recursion relation

$$\boldsymbol{x}^{(m+1)} = \boldsymbol{x}^{(m)} - \kappa_m \nabla J(\boldsymbol{x}^{(m)}),$$

where

$$\kappa_m = \frac{(\boldsymbol{x}^{(m)} - \boldsymbol{x}^{(m-1)})^T (\nabla J(\boldsymbol{x}^{(m)}) - \nabla J(\boldsymbol{x}^{(m-1)}))}{\|\nabla J(\boldsymbol{x}^{(m)}) - \nabla J(\boldsymbol{x}^{(m-1)})\|^2}\,.$$

For this method to converge, a necessary condition is that the Hessian of the objective function J, namely the matrix of second derivatives of J with respect to the independent variables, be finite. It is possible to show that this condition holds for both J_2 and $\widetilde{J}_3$. The calculations are lengthy but straightforward: we do the calculation of these derivatives explicitly for J_2 in Appendix B.

2.1.4. Implementation and Results

In this study we apply the above gradient descent algorithm to the functional $J_2(\boldsymbol{d}, T_0, T_1)$ considered as a function of $\boldsymbol{d}$ only, i.e. we prescribe T_0 and T_1 to take nominal values. With this restriction, the gradient of $J_2(\boldsymbol{d}, T_0, T_1)$ has only n components, namely $\frac{\partial J_2}{\partial d_i}$, $i = 1, \ldots, n$, where n is the total number of generators.

In order to implement this algorithm we wrote a *Mathematica* program (details of which can be found in Appendix C) which takes an initial seed $\boldsymbol{d}^{(0)}$, calculates the next seed $\boldsymbol{d}^{(1)}$ using a classical line-search algorithm (that takes 0.5 seconds) and then applies the Barzilai-Borwein gradient descent method (Barzilai and Borwein, 1988) to complete up to 200 iterations $\boldsymbol{d}^{(2)}, \ldots, \boldsymbol{d}^{(200)}$ (this takes 3 seconds). The calculations were done on a MacBook Air with a 1.7 GHz Intel Core i7 processor, with 8 GB 1600 MHz DDR3 of RAM.

Parameter values. We use the parameters from 20 sites that were provided by Captured Carbon (see Appendix A). We set the constant representing fines to $K = 0.1$. This means the cost of producing more power than the straight ramp line of the Virtual Power Plant is 10 times higher than the fines due to producing less power than this reference ramp. Initial and final times for this reference Virtual Power Plant ramp are set to $T_0 = 0, T_1 = 60$ min.

Sensitivity to choice of seed. We found that the method is quite sensitive to the choice of initial seed. This shows that there are a lot of local minima, which is good from the point of view of the "equity" challenge. On the other hand, for some initial seeds the convergence is bad, and for many initial seeds the method does not provide a monotonous decrease of the objective function via iterations; rather, the cost takes transient jumps that can last for up to 100 iterations before entering a monotonous decay phase. Due to this sensitivity we performed a search over 30 initial seeds chosen randomly, so that for each seed the algorithm would be applied, leading to 200 iterations in each case. In this way we could obtain a good indication of the different behaviours. In figure 4 we show the best case we found, which achieved the minimum cost (72 euro) over the 30 cases studied.

To illustrate the overall behaviour amongst the 30 cases studied, we calculated the optimum configuration for each case and made a histogram of the achieved minimum cost of those optimal configurations (see figure 5). This cost is defined as the objective function J_2 (with units of energy), evaluated at the local optimum, multiplied by the sum of the prices per unit energy of the 20 sites considered. We can see that 23 out of the 30 cases have a minimum cost below 350 euro: these 23 cases produce iterations that converge to a local minimum, while the remaining 7 cases produce non-convergent iterations. For example, the outlier at

474 euro is actually a case where the method did not converge: it achieved a minimum at the 89-th iteration and failed to converge at subsequent iterations. This is shown in figure 6.

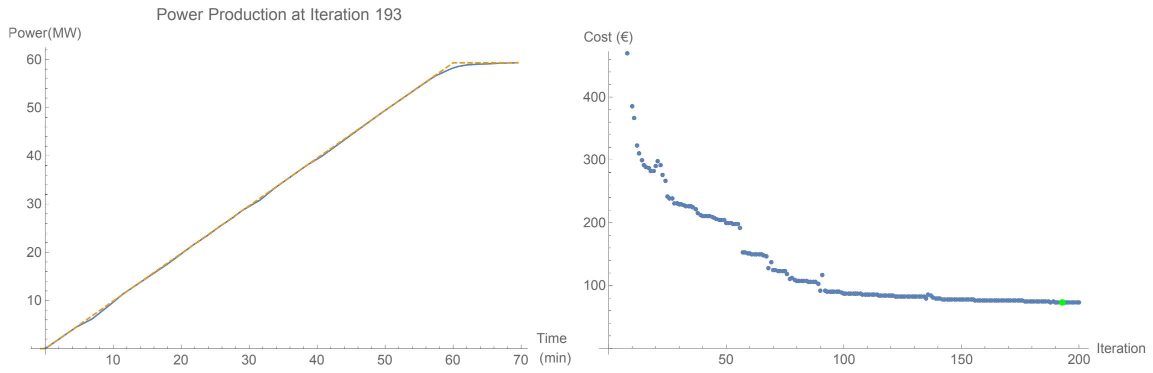


Figure 4: Left Panel: Virtual Power Station production power (actual in blue; ideal in orange) at iteration 193 of the **best minimum found over 30 seeds (cost: 72 euro).** Right Panel: Cost as a function of the iteration number. The green dot denotes the cost at the iteration 193.

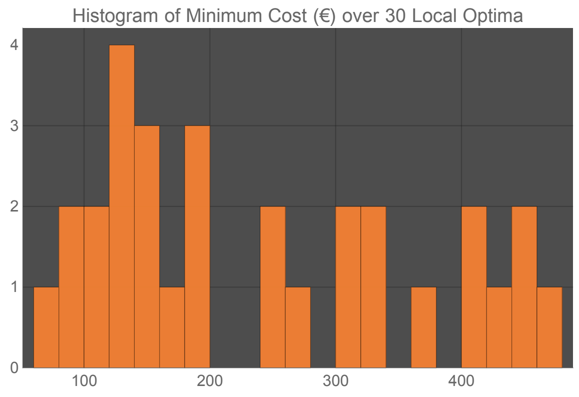


Figure 5: Histogram, over the 30 local optima found, of the cost associated with each local optimum. This cost is defined as the objective function J_2 (with units of energy), evaluated at the local optimum, multiplied by the sum of the prices per unit energy of the 20 sites considered.

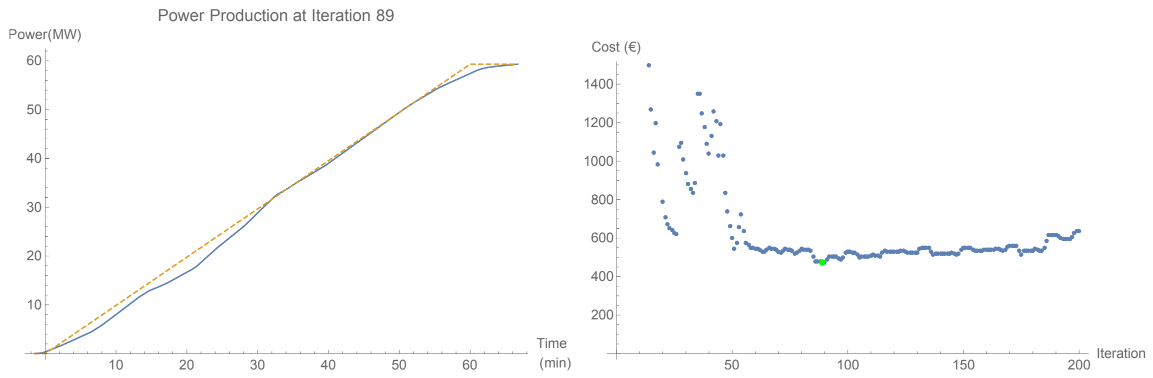


Figure 6: Left Panel: Virtual Power Station production power (actual in blue; ideal in orange) at iteration 89 of the **worst minimum found over 30 seeds (cost: 472 euro).** Right Panel: Cost as a function of the iteration number. The green dot denotes the cost at the iteration 89.

Equity. We briefly consider the equity challenge. Figure 7 shows 20 histograms, one for each generator site, of the starting times over the 23 different optimal configurations found where the method converged: these configurations are local minima with costs below 350 euro. We can see that there are optimal configurations for which any given site starts as late as 25 minutes after the first site starts. Moreover, there is a good spread of possible starting times for many sites (except perhaps for sites 1 and 15, which typically start during the first 10 minutes). Notice that due to the fact that some of the sites have long ramping times (over 50 minutes for site 14), this spread of starting times implies that the total time spent from non-zero actual production to reaching maximum capacity will exceed the nominal 60 minutes of ideal Virtual Power Plant ramping time. Figure 8 shows a histogram, over the 23 optimal solutions found, of these actual times spent going from zero to maximum capacity.

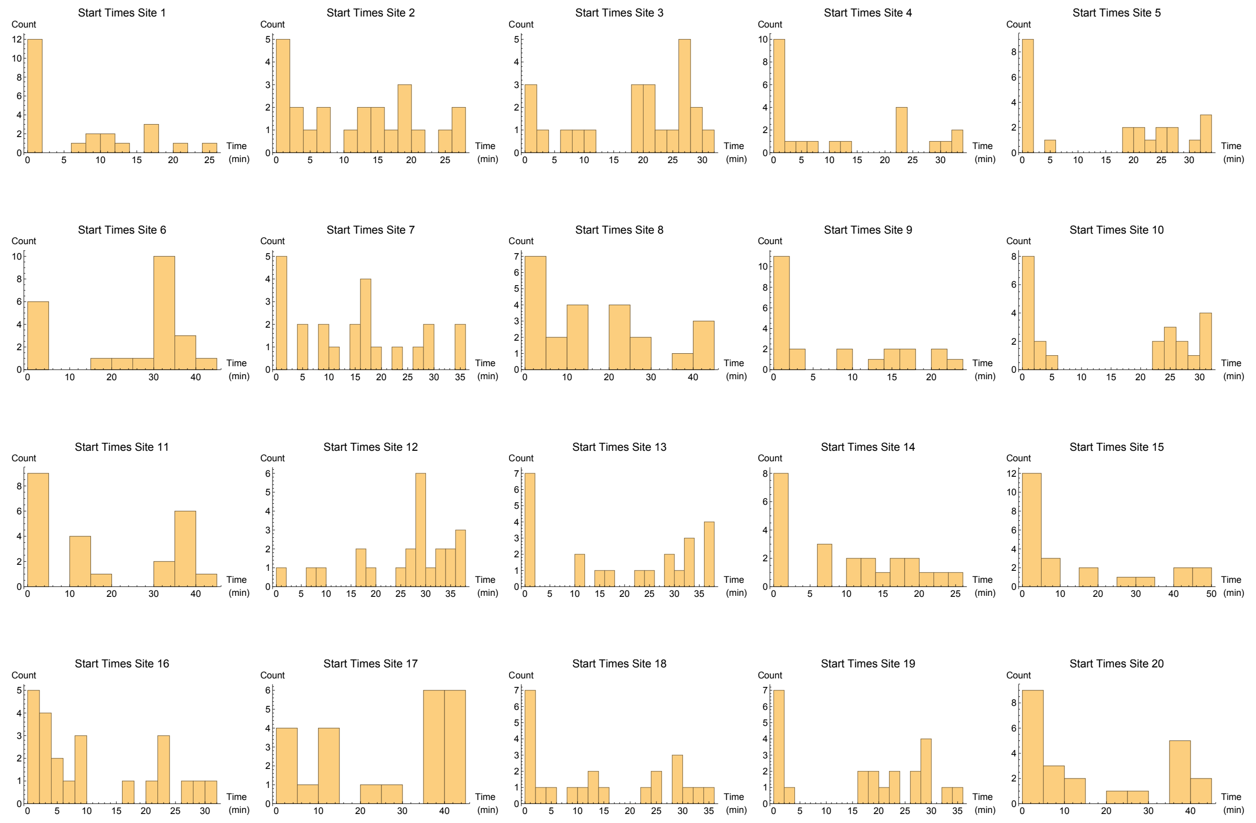


Figure 7: Addressing the equity challenge using the gradient descent method. Histograms, for each of the 20 different generator sites in the problem, of the starting times over the 23 different optimal solutions found that produced minimum costs lower than 350 euro.

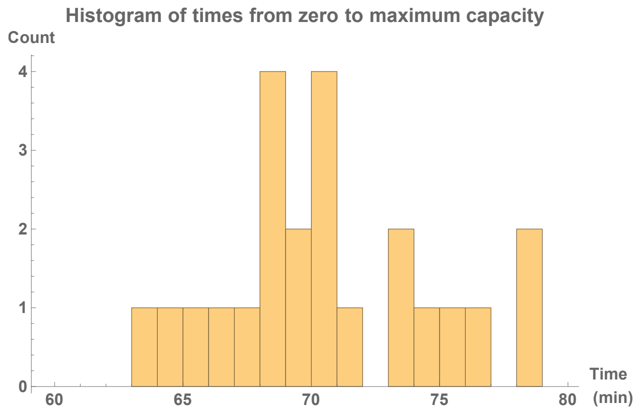


Figure 8: Addressing the equity challenge using the gradient descent method. Histogram, over the 23 different optimal solutions found that produced minimum costs lower than 350 euro, of the time spent ramping up the individual sites: this is measured from the time when the earliest site starts ramping up production until the time when the latest site reaches its maximum capacity. Notice that the ideal ramping time of the Virtual Power Plant is set to 60 minutes.

2.2. Mixed Integer Linear Programming

In this section we describe the Mixed Integer Linear Programming (MILP) approach used to solve the problem presented by Captured Carbon. MILP is an optimisation technique that decides the level of different variables so as to find the optimum value of a specified function, known as the *objective function*. Typically MILPs include constraints that place limits on the values the decision variables may take. Furthermore, some variables are limited such that they may only take integer variables while other variables may be continuous. A MILP is considered to be linear when both the objective function and each of the constraints are linear.

For this workshop we used MILP to determine how to *optimally* schedule each generating site. The methodology is in contrast to the other approaches considered in this report which all seek *near-optimal* solutions. We consider two different objectives to optimise using MILP. Firstly, we minimise the sum of the distances between the actual summed ramping rates and the target ramping rate line:

$$\min \sum_t \big(above_t + K below_t\big), \tag{15}$$

where $above_t$ and $upper_t$ are continuous decision variables representing the distances above and below the target line at time t, respectively. The parameter K represents the penalty multiplier for when the actual ramping rate is below line, relative to when it is above, i.e., being below the line is K times worse that being above the line. In Figure 9, we consider a value of $K = 10$.

The second objective minimises was the total cost associated with consumer generating:

$$\min \sum_{i,t} \gamma_i gen_{i,t} + \overline{\gamma} \sum_t \big(above_t + K below_t\big), \tag{16}$$

where $gen_{i,t}$ is the continuous decision variable representing generation from site i at time t. The parameters γ_i and $\overline{\gamma}$ represent the cost associated with site i generating and the average of those costs, respectively.

When seeking to optimise both equations (15) and (16), the following constraints must be observed:

$$\begin{aligned}
gen_{i,t} &\leq C_i, \ \forall i, t, && \text{(17a)}\\
gen_{i,t} &= gen_{i,t-1} + R_i\big(b_{i,t} - \overline{b_{i,t}}\big), \ \forall i, t, && \text{(17b)}\\
b_{i,t} &\geq b_{i,t-1}, \ \forall i, t, && \text{(17c)}\\
\overline{b_{i,t}} &\geq \overline{b_{i,t-1}}, \ \forall i, t, && \text{(17d)}\\
\sum_t (b_{i,t} - \overline{b_{i,t}}) &= \tau_i, \ \forall i, && \text{(17e)}\\
\sum_i gen_{i,t} + below_t - above_t &= \frac{t \sum_i C_i}{|\tilde{T}|}, \ \forall t, && \text{(17f)}\\
b_{i,t}, \overline{b_{i,t}} &\in \{0, 1\}, \ \forall i, t, && \text{(17g)}\\
gen_{i,t}, above_t, below_t &\geq 0, \ \forall i, t. && \text{(17h)}
\end{aligned}$$

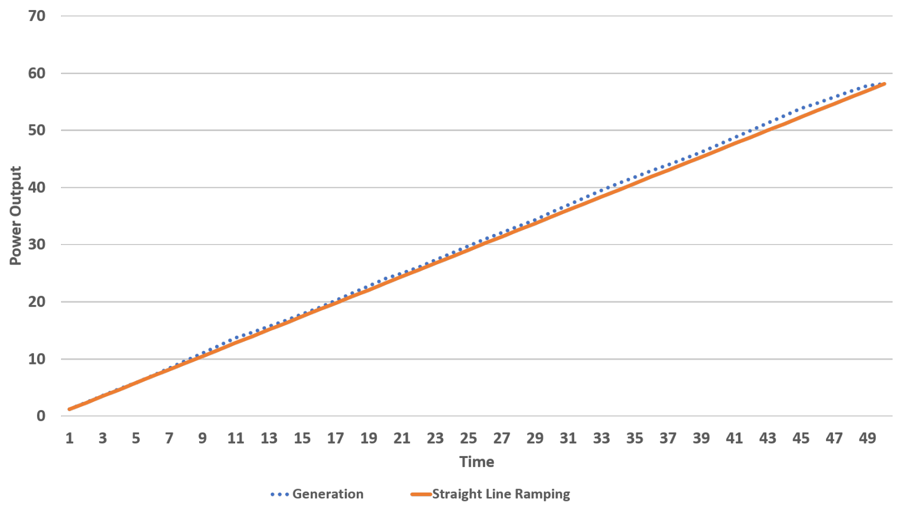
The binary decision variable $b_{i,t}$ indicates whether site i is generating at time t ($b_{i,t} = 1$) or not ($b_{i,t} = 0$). Similarly, the binary decision variable $\overline{b_{i,t}}$ indicates whether site i is generating at its maximum capacity at time t ($\overline{b_{i,t}} = 1$) or not ($\overline{b_{i,t}} = 0$). Constraint (17a) ensures site i cannot generate more than its maximum capacity. Constraint (17b) ensures, that if site i is generating at time t, its generation must equal its generation from the previous time step plus its ramping rate (R_i). However, when site i is at its maximum capacity, constraint (17b) also ensures it stays generating at that maximum capacity. Constraint (17c) ensures that if site i is generating at time $t-1$, then it must also generate at time t, i.e., once it begins ramping, it must continue. Similarly, constraint (17d) ensures that if site i is generating at its maximum capacity at time $t-1$, then it must also do so at time t.

The difference $b_{i,t} - \overline{b_{i,t}}$ is one if site i is generating, but not at its maximum capacity, at time t, and zero otherwise. Hence, the sum of these differences must equal site i's ramp time (τ_i), as indicated by equation (17e). Finally, constraint (17f) ensures that the sum of generation across the sites plus/minus any errors below/above must equal the target ramping line ($\frac{t \sum_i C_i}{|\tilde{T}|}$), where $|\tilde{T}|$ represents the total number of timesteps considered.

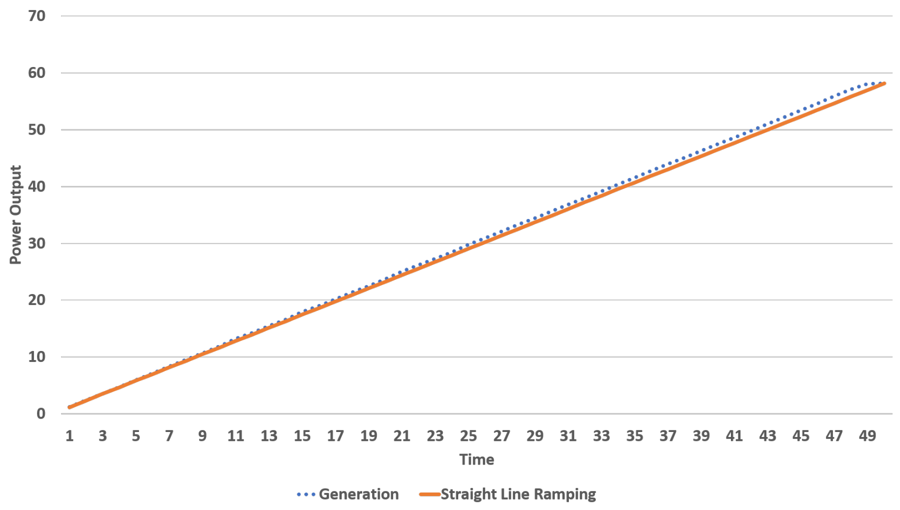
Figure 9 displays the results and shows that the MILP approach can schedule the generators such that the output ramp rate is linear. In Figure 9a, objective function (16) is minimised and the problem solves in 17 seconds. In Figure 9b, objective function (15) is minimised and the problem solves in $\approx$9 hours. In Figure 9c, objective function (15) is again minimised. However, in this case, the algorithm ceases after 60 seconds and the solution obtained at that point is displayed, showing a near linear ramp rate. For these results, the problem was solved using the programming language GAMS[1] and were performed on a 3.4 Ghz, 16GB RAM desktop PC.

In Appendix D, we provide the GAMS and Python code used to solve the problem when equation (16) is the objective function.

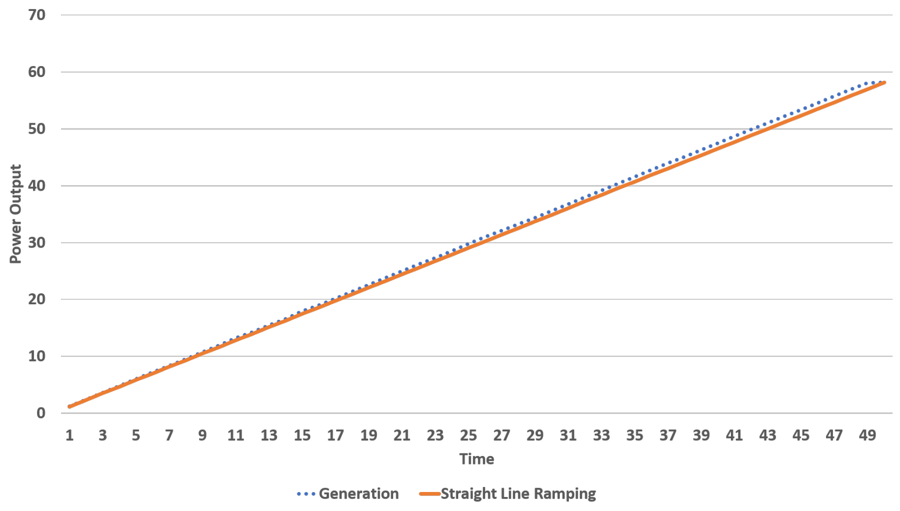
[1] `www.gams.com`



(a) Objective function (16).



(b) Objective function (15).



(c) Objective function (15) with 60 second time limit.

Figure 9: Power generation results from MILP approach.

2.3. Systematic Adjustment

2.3.1. Overview

The systematic adjustment algorithm is a heuristic approach where we start from the extreme solution where every site starts ramping as late as possible, and then sequentially make minimum-cost adjustments to this until it can be ensured that the total power generated from the sites is greater than the optimum power output throughout. The systematic adjustment algorithm was implemented in a discrete-time framework, but could be adapted to continuous time, although this would be more computationally expensive.

2.3.2. Method

Our aim is to ensure that the target power output is always achieved, while minimising any excess output. Our initial step is to set the start of the ramp-up time, $t_i = d_i + R_i$, for each site so that they all achieved their full capacity C_i at the same time T_1, as shown in Figure 10 and Listing 1 Then, we set the earliest of these start up times as T_0. We assumed that $[T_0, T_1]$ was the optimal range over which to achieve the company's ramp-up time.

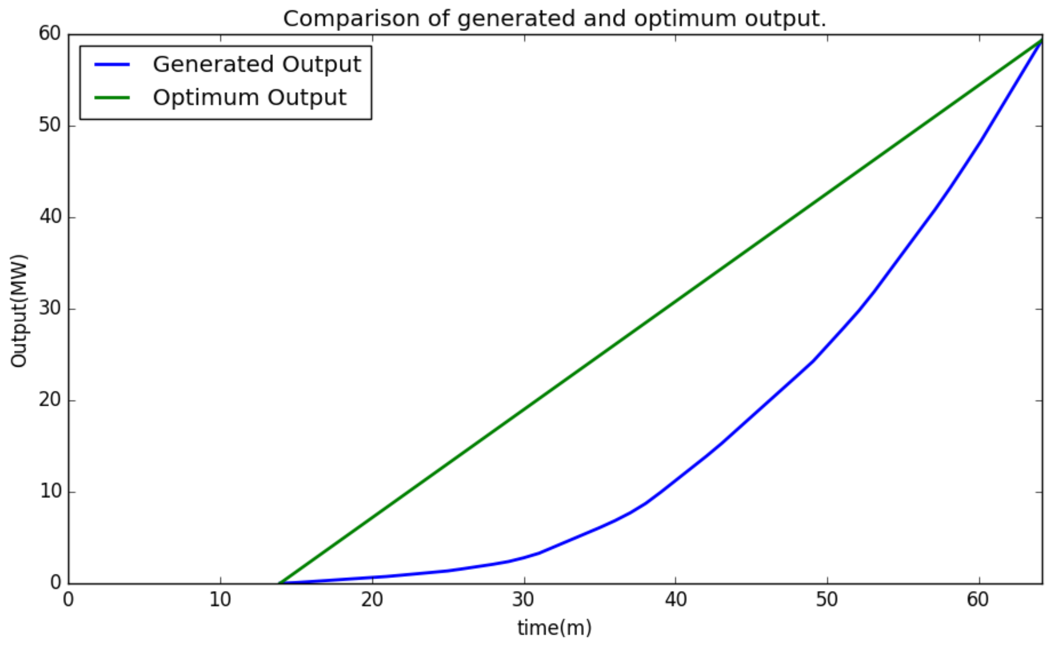


Figure 10: Initial configuration.

We discretise time between T_0 and T_1 into M segments. Let u_k represent the k-th time in the discretisation, so that $u_0 = T_0$ and $u_M = T_1$.

```
init_t_i = T_1-ramp_up_norms
```

Listing 1: Python code extract: set initial ramp-up start times

We begin by comparing the output generated by the initial configuration at u_0 with the optimal output at u_0. These match, since both are zero. Having established that there is no discrepancy at $t = u_0$, we move to $t = u_1$, where we will generally find that the power generated by the initial configuration is less than the optimal output. In order to have sufficient power generated at u_1, we will need to change at least one of the t_i values to be less than u_1.

We consider all of the sites where $t_i > u_0$, and identify the site where t_i can be shifted to u_0 at least cost. Since the ramp-up time is already included in the cost the additional cost, for each site will be $C_i(t_i - u_0) \times$ (unit price).

```
while Output[t] < (T_1-T_0)*time_line[t]:
        # Find the start time of the site to move
        for i in range(0, no_of_sites):
                if (init_t_i[i] - time_line[t-1]) < 1e-10:
                        init_t_i[i] = 100 # ignores sites that are contributing to the
                            power output at the start of this time-step
        for i in range(0, no_of_sites):
                if init_t_i[i] < 100:
                        find_min[i] = max((init_t_i[i] - time_line[t-1])*capacity_norms[
                            i]*prices[i]/60,0.0) #hose the site which will incur the
                            least increase in cost
                else:
                        find_min[i] = 100 # ignores sites that are contributing to the
                            power output at the start of this time-step
        min_cost = np.argmin(find_min) # index of site to move

        # Move the start time of the site chosen to the previous time-step
        #  and amend the relevant information in the arrays
        for t2 in range(1, time_steps+1):
                if time_line[t2]>init_t_i[min_cost] and time_line[t] <=T_1:
                        Output[t2] -= (time_line[t2] - init_t_i[min_cost])*
                            capacity_norms[min_cost]/ramp_up_norms[min_cost]
                        Output_detailed[t2][min_cost] -= (time_line[t2] - init_t_i[
                            min_cost])*capacity_norms[min_cost]/ramp_up_norms[min_cost]
        move_t_i = init_t_i[min_cost] - time_line[t-1]
        init_t_i[min_cost] = time_line[t-1]
        actual_t_i[min_cost] = time_line[t-1]
        for t2 in range(1, time_steps+1):
                if time_line[t2] > init_t_i[min_cost] and time_line[t2] <=(T_1 -
                    move_t_i):
                        Output[t2] +=  (time_line[t2] - init_t_i[min_cost])*
                            capacity_norms[min_cost]/ramp_up_norms[min_cost]
                        Output_detailed[t2][min_cost] +=  (time_line[t2] - init_t_i[
                            min_cost])*capacity_norms[min_cost]/ramp_up_norms[min_cost]
                elif time_line[t2] > (T_1 - move_t_i) and time_line[t2]<=T_1:
                        Output[t2] += capacity_norms[min_cost]
                        Output_detailed[t2][min_cost] += capacity_norms[min_cost]
```

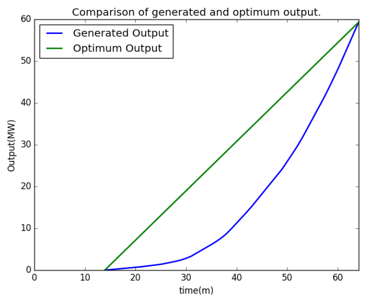
Listing 2: Python code extract: move site with lowest increase in cost

Having changed one t_i value, see Listing 2, we assess the new configuration, see sub-figure 11a. If the power generated at $t = u_1$ is still below the optimal level, we again consider all sites where $t_i > u_0$ and identify the site where t_i can be shifted to u_0 at least cost; this process can be repeated until we obtain a configuration where the power generated at u_1 is sufficient, see sub-figures 11b to 11l. Once sufficient power at u_1 is obtained, we consider sites with $t_i < u_1$ and whose power output is less than the excess power being generated at $t = u_1$; from these sites we select the site with the largest power output at u_1 and move its t_i back to its latest possible value, $T_0 - \tau_i$, see sub-figure 11m and Listing 3, and repeat if necessary.

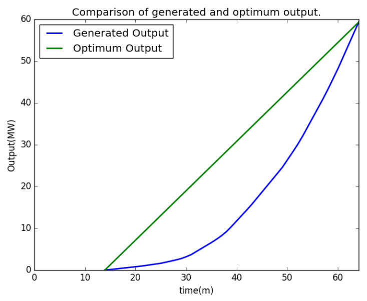
```
# When the minimum requirements have been attained
# check if any site can be moved back to it's original start time
# while still achieving the overall minimum requirements, then from these
# move the site that gives the largest contibution to the current
# time-step, and update the arrays
push_back = 0
if  Output[t] > (T_1 - T_0)*time_line[t]:
        for i in range(1, no_of_sites):
                if Output_detailed[t][i] <  (Output[t] - (T_1-T_0)*time_line[t]) and
                    Output_detailed[t][i]>0:
                        if Output_detailed[t][i] > push_back:
                                push_back = Output_detailed[t][i]
                                push_back_index = i
                if push_back > 0:
                        for t3 in range(0, t):
                                if Output_detailed[t3][push_back_index] <  (Output[t3] -
                                    (T_1 - T_0)*time_line[t3]):
                                        push_back = 0
if push_back > 0:
        move_t_i = T_1 - ramp_up_norms[push_back_index]
        for t2 in range(1, time_steps+1):
                Output[t2] =  Output[t2] - Output_detailed[t2][push_back_index]
                Output_detailed[t2][push_back_index] = 0
        actual_t_i[push_back_index] = T_1 - ramp_up_norms[push_back_index]
        for t2 in range (0, time_steps+1):
                if time_line[t2] > actual_t_i[push_back_index] and time_line[t2] <= T_1:
                        Output[t2] +=  (time_line[t2] - actual_t_i[push_back_index])*
                            capacity_norms[push_back_index]/ramp_up_norms[
                            push_back_index]
                        Output_detailed[t2][push_back_index] = (time_line[t2] -
                            actual_t_i[push_back_index])*capacity_norms[push_back_index
                            ]/ramp_up_norms[push_back_index]
```

Listing 3: Python code extract: move relevant site back to initial start time

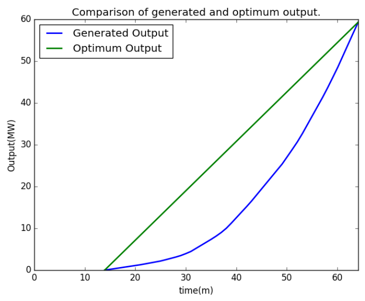
Now that sufficient power is generated at u_1 and we have moved back any sites that are not necessary to achieve this, we step forward in time to identify the next time-step where the power generation is insufficient, $t = u_P$. We repeat the same process as before; we consider all of the sites where $t_i > u_{P-1}$ and identify the site where t_i can be shifted to u_{P-1} at least cost. We change this t_i value and assess the new configuration, changing more t_i values if necessary, see sub-figures 11n to 11p. We repeat the process until sufficient power is generated at all times between T_0 and T_1, see sub-figures 11q and 11r.



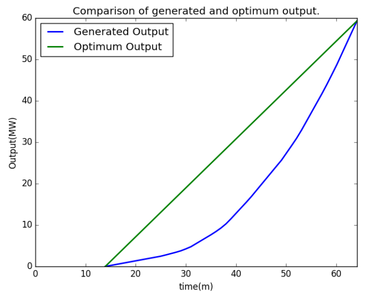
(a) Move site 2 forward.



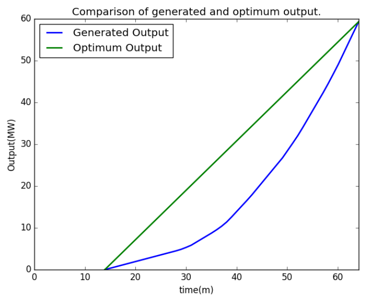
(b) Move site 18 forward.



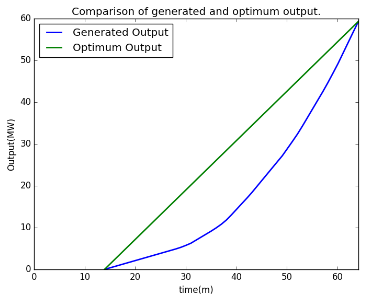
(c) Move site 4 forward.



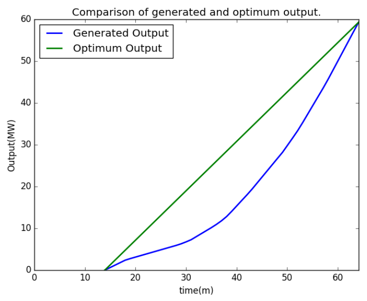
(d) Move site 7 forward.



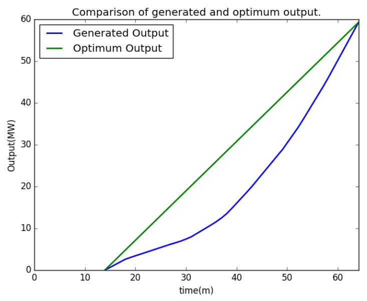
(e) Move site 3 forward.



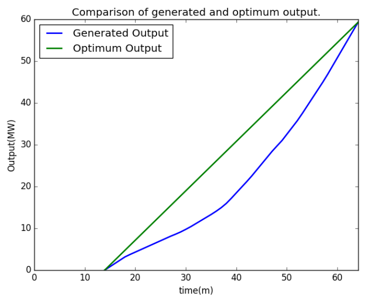
(f) Move site 19 forward.



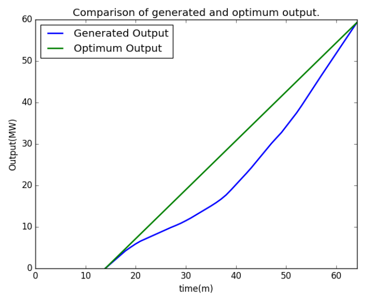
(g) Move site 9 forward.



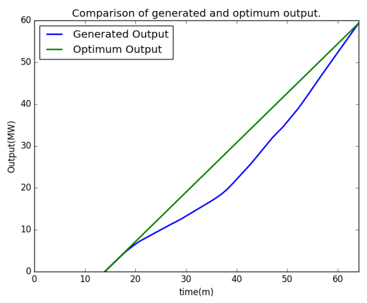
(h) Move site 1 forward.



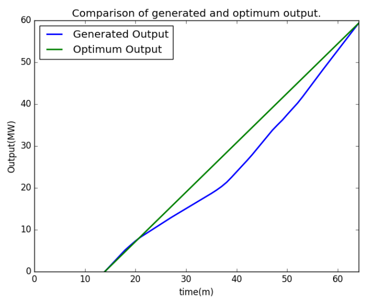
(i) Move site 10 forward.



(j) Move site 16 forward.

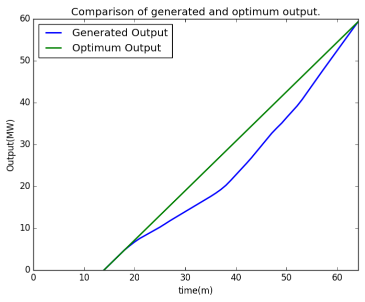


(k) Move site 12 forward.

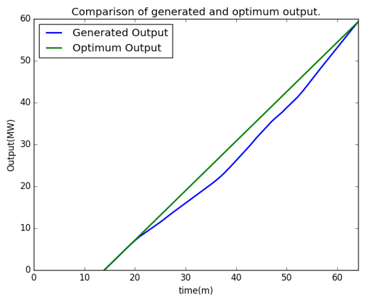


(l) Move site 5 forward.

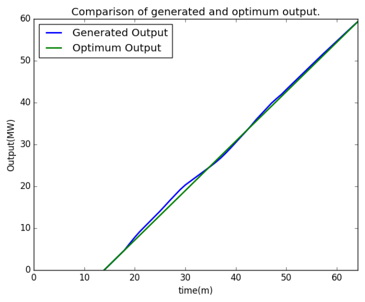
Figure 11: Step by step results when moving site start-up times.



(m) Move site 3 backward.

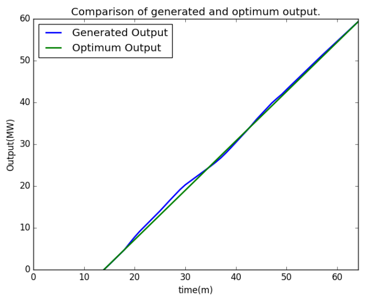


(n) Move site 13 forward.

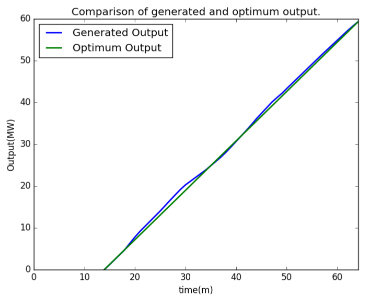


(o) Move site 15 forward.

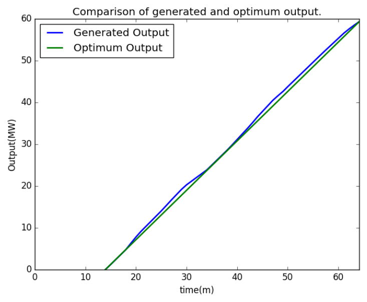
Figure 11: Step by step results when moving site start-up times.



(p) Move site 2 backward.



(q) Move site 6 forward.



(r) Move site 11 forward.

Figure 11: Step by step results when moving site start-up times (cont.).

2.3.3. Results

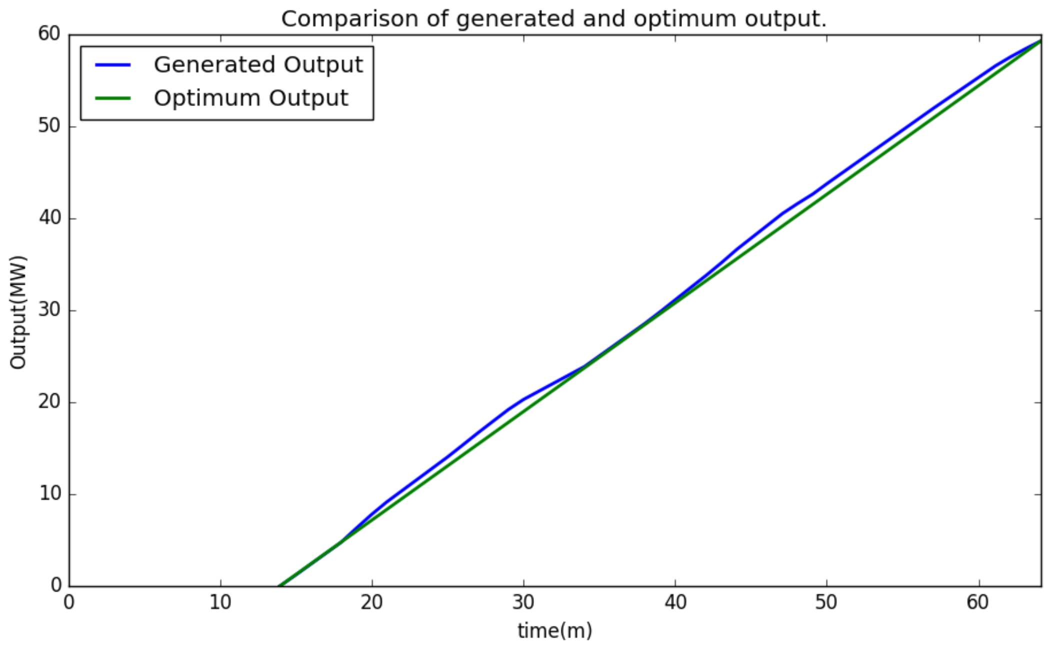


Figure 12: Final configuration.

Our results, based on the sample data given, show a generated power output that achieves the target output throughout, as shown in Figure 12, while generating very little excess power, ensuring no penalties are incurred. The power generated is broken down by site in Figure 13, where we can see that many sites are started at T_0, with a small number starting later to achieve our solution.

This method uses a heuristic approach and moves the start time of some sites, based on minimum additional cost, to ensure the target output is achieved throughout. Thereby, we ensure no penalties are incurred. Good results are achieved for the sample data given, with a total cost of €4,478, a ramp-up time of 50.24 minutes and an overall time of 64.15 minutes including the response and delay times. The method runs quickly and should scale well. The processing time for the sample data supplied, using fifty time-steps, is less that one-hundredth of a second.

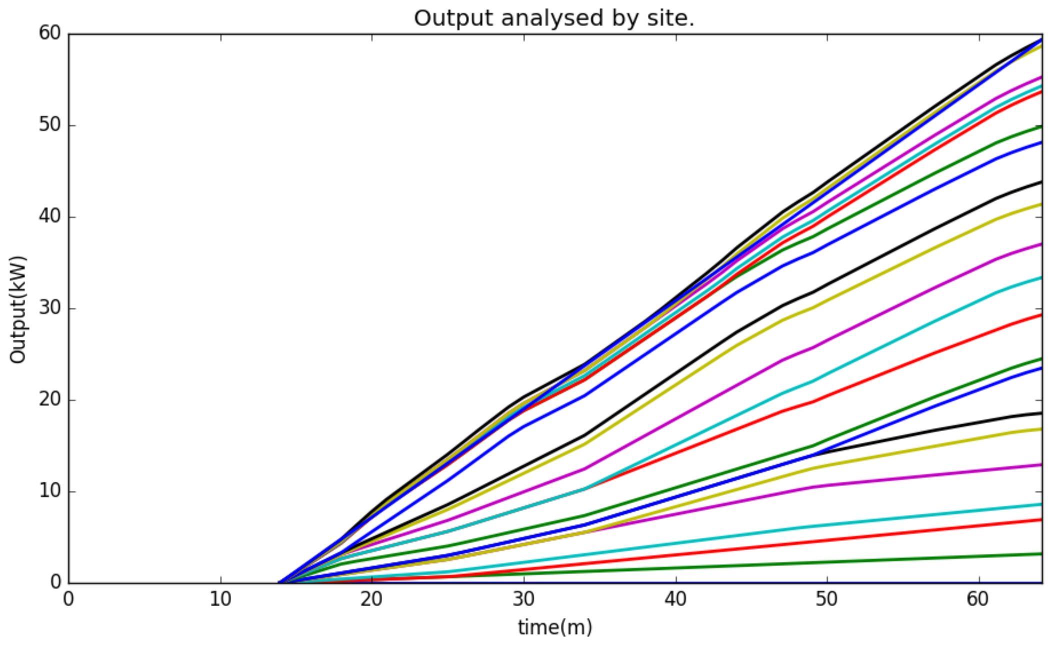


Figure 13: Breakdown by site.

2.3.4. Algorithm variations

This algorithm is deterministic and will always produce the same output. The t_i value to move could be decided on using different criteria, for example:

i. the t_i value that needs to be moved the least to contribute as required, this alternative is already an option in the python file created;

ii. the t_i value that has the least cost per MW;

iii. weighting the sites based on their location in the previous setup;

iv. the time a site takes to start ramping-up; and

v. the time a site takes ramping-up.

These could all produce different arrangements of site start-up times, while still ensuring sufficient power is generated throughout.

2.4. Bin packing

2.4.1. Overview

This bin-packing algorithm is a simulated annealing approach to solving this problem in a discrete-time framework. Some aspects of the algorithm are similar to the simulated annealing approach to the direct problem described in Section 2.5. The main difference between the two approaches is that the approach described in Section 2.5 treats the original problem directly (taking advantage of functions for simulated annealing functionality available in Matlab), while the approach described in this section takes advantage of the fact that the original problem can be transformed into a bin-packing problem that is easier to solve.

The classical bin-packing problem involves packing a list of items of different sizes into the smallest number of bins, each of which has a maximum capacity (Rao and Iyengar, 1994). We propose a slight modification to the classical formulation of the bin-packing problem. More specifically, recall that each power generator has a specific ramping profile (as shown in Figure 14 (a) where the ramping profile is represented by the function $f(x)$). We note that the rate of change of the ramping profile with respect to time, $f'(x)$—shown in Figure 14 (b)—is zero everywhere except while the generator is ramping (at which point it is equal to the ramping rate, i.e., C_i/τ_i). We let B_i (the block associated with each generator) be represented by a rectangle of width τ_i and height C_i/τ_i. We then try and pack each block as efficiently as possible into a bin of width $T_1 - T_0$ and a maximum height of $\sum_{i=1}^{I} C_i/\tau_i$.

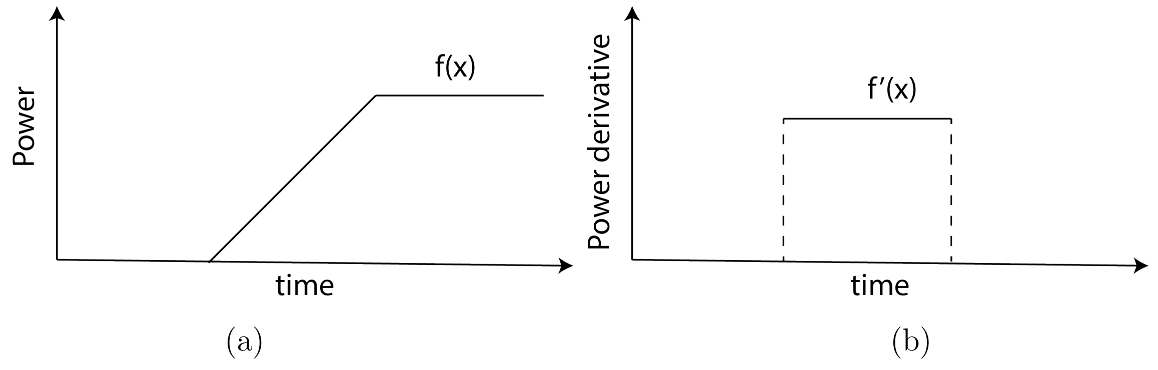


Figure 14: (a) Power ramping up versus time and (b) derivative of power ramping up over time.

We arrange the blocks in some arbitrary packing order and fit each block inside the bin one at a time until they have all been packed. This specific bin-packing algorithm is deterministic, however changing the packing order of the blocks can change the solution. Since there are I generators, we can arrange the blocks in $I!$ possible orders. As I increases, it becomes computationally prohibitive to test every possible solution, therefore we choose to apply an annealing process to search the parameter space more efficiently.

2.4.2. Method

The bin packing algorithm is designed as follows:

1. We begin by discretising the bin via an $N \times M$ rectangular mesh, i.e., $T_1 - T_0$ is discretised into N equal segments, and $\sum_{i=1}^{I} C_i/\tau_i$ is discretised into M equal segments.

2. Next, we let O be an ordered set containing each block in some arbitrary order.

3. The deterministic bin-packing process is as follows:

   (a) We pick the first block in the ordered set O. This will be the first block that we sort.

   (b) We search the bin until we find a free space to place the current block that we are trying to place. At which point, we place the block inside the bin. Note that the search routine is as follows:

      i. Begin the search at the position $n = m = 1$.

      ii. Check if the bin is currently empty in the rectangular region given by $n \in [n, n + \tau_i]$ and $m \in [m, m + C_i/\tau_i]$. If it does fit, the search is over.

      iii. If $n = N - \tau_i$, let $m = m + 1$, $n = 1$, and return to step ii.

      iv. If $n < N - \tau_i$, let $n = n + 1$, and return to step ii.

   (c) If we have not yet placed each block in O, choose the next block in the ordered set and return to step (b).

   (d) Evaluate the cost function score associated with starting the generators using this solution.

4. Pick a new candidate set, O', at random from all neighbours of the existing set O. One way to pick a neighbouring set is to randomly choose two elements in O, and then reverse the order of these elements. Then repeat steps 3 (a) – (d). The solution generated by O' might be better or worse than the solution generated by O.

5. If O' performs better than O, accept it as the new O.

6. If O' performs worse than O, accept it with some probability. The probability of accepting an inferior solution is a function of the temperature of the annealing process. A higher temperature makes you more likely to accept O'.

7. Go back to step 4 and repeat many times, lowering the temperature a bit at each iteration, until you get to a low temperature and arrive at your (hopefully global,

possibly local) minimum. If you are not sufficiently satisfied with the result, try the process again, perhaps with a different temperature cooling schedule.

The key to the simulated annealing method is in step 6: we still sometimes accept a worse solution, because it might be the stepping stone that gets us out of a local minimum and ultimately closer to the global minimum. The temperature is higher at the beginning of the annealing process, so that initially we accept more solutions. As the temperature decreases, we decrease the likelihood of accepting bad solutions.

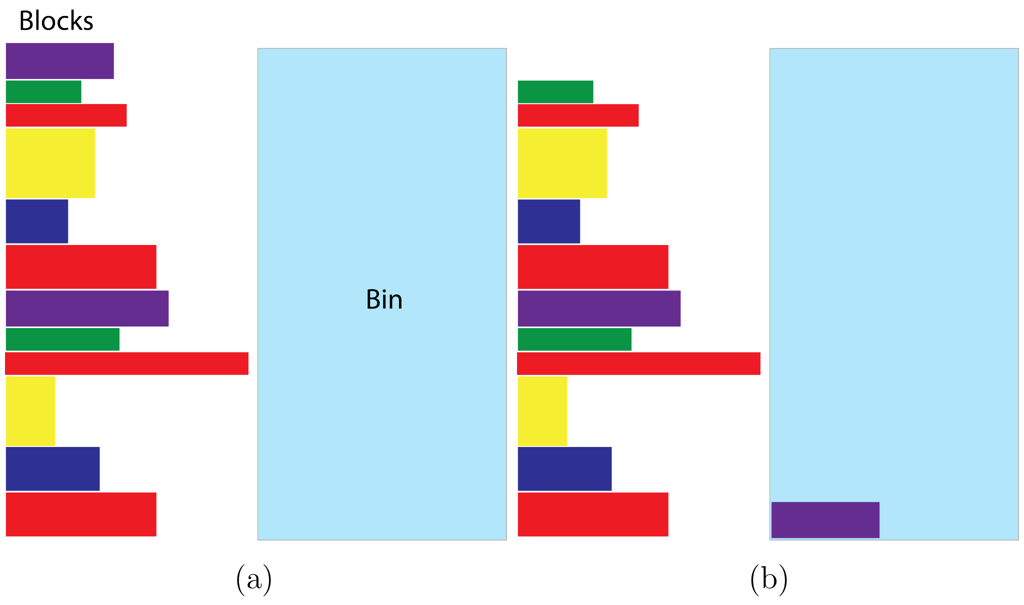
2.4.3. Algorithm variations

There are several reasonable variations to this algorithm.

- **Naïve hill climbing:** Similar to the simulated annealing approach except step 6 is ignored. This can increase the likelihood of the solution converging at a local minimum rather than a global minimum, however, this algorithm can also converge faster than the simulated annealing algorithm.
- **Alternative neighbour choice:** In step 4 we suggested that one way to pick a neighbouring set is to randomly choose two elements in O, and then reverse the order of these elements. This is a reasonable way to choose a neighbouring set, however it is not the only way to choose a neighbouring set. The number of elements to swap could be a function of the temperature or number of generators.

2.4.4. Results bin packing

Figure 15 provides an illustrated demonstration of the bin-packing process. In Figure (a) the blocks are sorted in some arbitrary order, ready to pack. We note that the height of the bin is equal to the sum of the heights of the blocks and the width of the bin is equal to the maximum block width (i.e., the width of the red block in position 9). In Figure (b) the first block is packed and it is placed in the bottom left of the bin. We note that the first item will always be placed in this position since the searching method always starts in the bottom left of the bin and at the beginning of the process the bin is empty. In Figure (c) five blocks have been stacked and the algorithm is trying to find a place for the red block in position 6, and in figure (d) all of the blocks have been stacked.



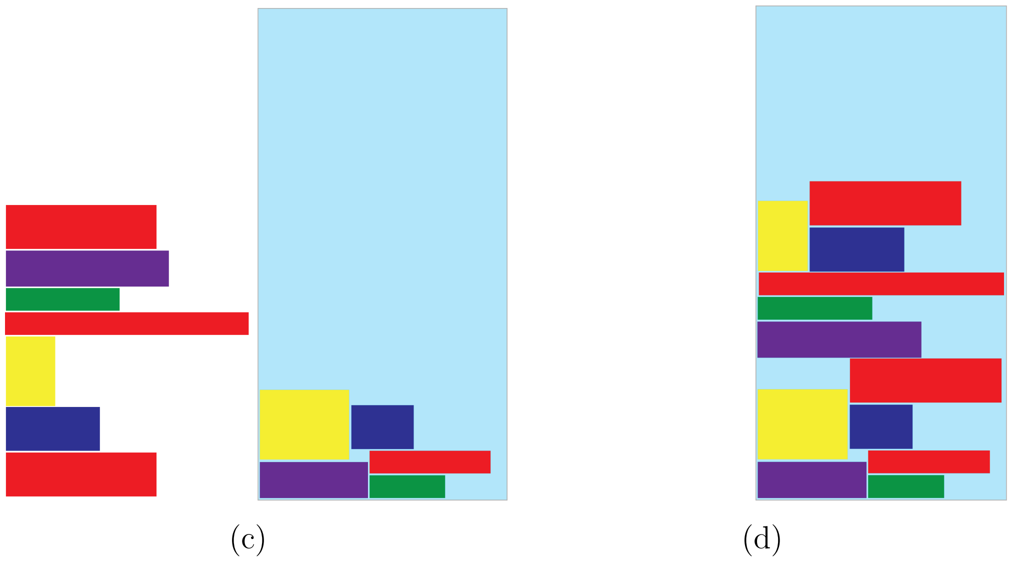


Figure 15: Graphical example of bin-packing algorithm.

2.5. Simulated Annealing

As described in Section 2.4, simulated annealing is an iterated technique for finding an optimal solution to a problem. At each iteration, a random modification is made to the current guess of the optimal solution. If this leads to an improvement (a lower cost), the modification is accepted and becomes the new guess of the optimal solution. If there is no improvement (the modification leads to a solution with equal or higher cost), the modification is accepted with probability p. This probability of acceptance when there is no improvement is taken to depend on a variable, T, referred to as the temperature; higher values of T correspond to higher values of p.

The key to simulated annealing is that the temperature is decreased over the course of the search for an optimal solution. The high initial temperature means that there can be large movements around the space of possible solutions in the early stages, and then as the temperature decreases the guesses will move more consistently towards a minimum-cost solution. The purpose of the temperature is to make the method unlikely to settle on a local minimum; the exploration of the parameter space in the early stages makes it more likely for a global minimum solution to be achieved.

A number of software packages exist for running simulated annealing. Typically, such packages require a cost function (ideally one that is computationally efficient, since it will be evaluated frequently over the course of the search for an optimal solution) and a domain in which to seek solutions. The Matlab optimisation toolbox includes such a simulated annealing function. One way of approaching the problem is simply to use simulated annealing directly on the original problem, evaluating the cost based on the function $\tilde{J}_3$.

It is straightforward to determine the domain on which solutions should be sought when T_0 and T_1 are fixed at the outset. Given the total time to reach full power T_1 and the earliest time to begin ramping T_0, there is only a limited range of notification times, d_i, that are possible. Specifically, the fact that $d_i + R_i + \tau_i \leq T_1$ and that $d_i + R_i \geq T_0$ implies that $d_i \in [T_0 - R_i, T_1 - R_i - \tau_i]$. If there are n sites, this gives us an n-dimensional hyperrectangle (an n-orthotope) of possible solutions. Since the bounds of an n-orthotope are easily expressed in Cartesian coordinates, the implementation of these bounds is straightforward.

In Appendix E, we give commented code for implementing simulated annealing for this problem using Matlab's optimisation toolbox. The important feature of this code that makes it practical is that the code for evaluating the cost function has been written to be quick and efficient. Example output of this process is shown in Figure 16.

Simulated annealing applied directly to the original problem as described here has a random element. This is useful for the "equity" problem, since multiple runs of simulated annealing will lead to different solutions. Simulated annealing also depends on the starting

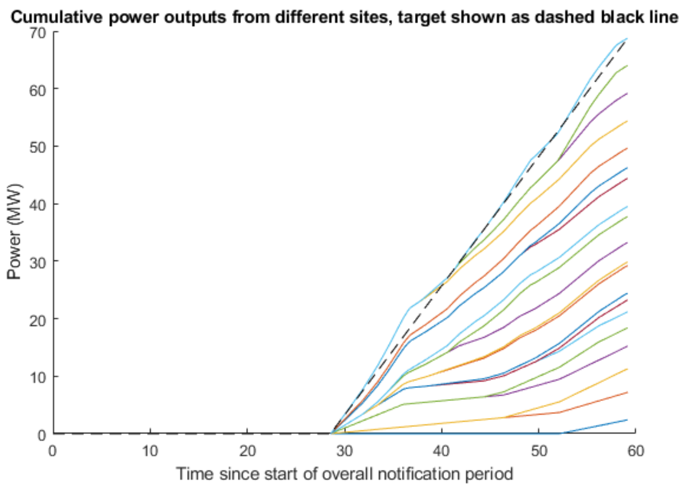


Figure 16: An example of a solution for twenty sites obtained using simulated annealing directly applied to the cost function $\tilde{J}_3$.

point (although to a lesser extent than methods such as gradient descent that will move directly to a local minimum). Hence, simulated annealing could be used as part of a multistep process. A crude optimisation method could be used to obtain a good starting point for simulated annealing, or (more likely to be useful), the output of simulated annealing could be refined by using it as the starting point for a method such as gradient descent.

There are extensions to the simulated annealing problem that are yet to be considered but could be done as further mathematical work. The main extension would be to consider modifications to the definition of the cost function. The present work all depends on using $\tilde{J}_3$ as the cost function and taking T_0 and T_1 as fixed. Alternatives would include using J_1 or J_2 as the cost function (which would simplify the problem), or performing the optimisation of T_0 and/or T_1 as well as $\mathbf{d}$. This modification, where T_0 and T_1 are not treated as fixed, would be a substantial and interesting change to the problem, and could lead to new insights into how to specify the best "virtual power plant" that will be equivalent to a given set of sites with known properties.

2.6. Ramp Tracking Simulation

As discussed in the Introduction, an *ideal* virtual power plant (VPP) replicates the operation of a single large power plant. A characteristic of ideal VPP operation is a linear and constant ramp to generation capacity (C). In financial terms, ramping avoids penalty fines by achieving contracted power capacity. On the other hand, excessive ramping, especially at ramp commencement, incurs costs of unpaid power. Fines and cost of unpaid power are discussed in the Introduction, with fines generally more expensive.

Given the risks of fines and costs, the Ramp tracking technique schedules VPP generators for two goals: 1.) track an ideal VPP ramp, and 2.) reduce excessive ramping above the ideal VPP ramp. Both goals are prioritised in early ramping in order to provide a margin of safety from fines and limit unpaid power.

Ramp tracking does not guarantee the optimal solution but is operationally intuitive and easier to code without expensive mathematical software. The implementation in code resembles a time domain simulator by checking the ramp value at fixed time steps.

2.6.1. Methodology

The ramp tracking algorithm schedules the VPP generators to maintain a specified difference above the ideal ramp rate when possible. Three algorithm parameters determine the scheduling of the VPP generators, led by the "Ideal VPP ramp time" (Table 5). Ideal VPP ramp time is the preferred duration in minutes from the start of the first generator(s) ramp to the completion of all generator ramps. Equation (18) expresses this parameter in notation from the Introduction. The same parameter in equation (19) calculates the ideal ramp rate in algorithm Phase 1. The algorithm phases are: 1) scheduling of first generators and 2) scheduling of additional generators as Phase 1 generators complete ramping.

Table 5: Parameters of ramp tracking algorithm

Parameter	Meaning	Phase
Ideal VPP ramp time	Ramp time of all VPP generators	1
minDiff	Minimum difference between actual and ideal ramp *before* scheduling inactive generator	1 & 2
addGenDiff	Preferred difference between actual and ideal ramp *after* scheduling inactive generator	2

$$\text{VPP Ramp Time} = T_1 - T_0 \qquad [mins] \tag{18}$$

$$\text{Ideal VPP Ramp Rate} = \frac{C}{VPP\,RampTime} \qquad [MW/min] \tag{19}$$

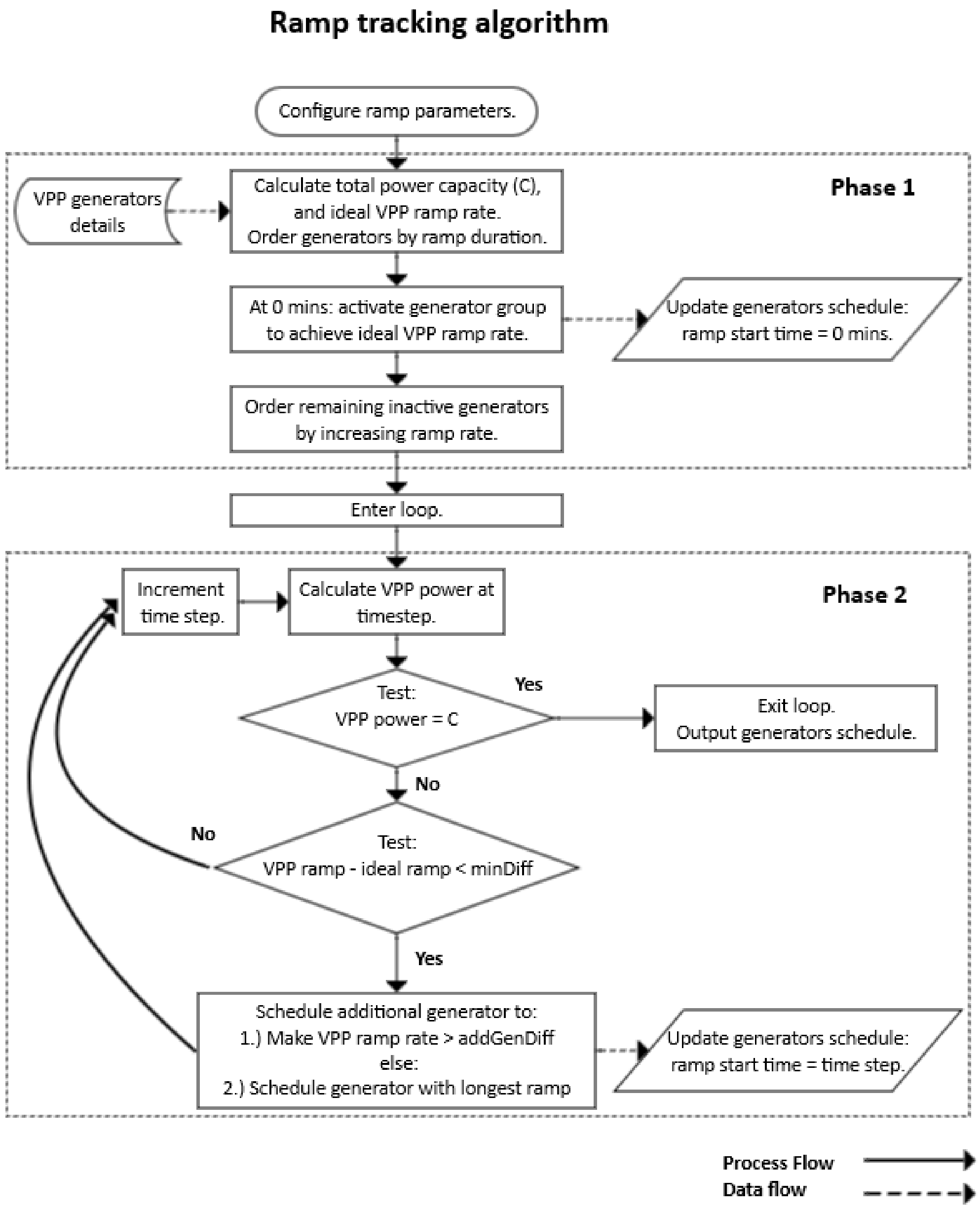

Ramp tracking algorithm
Configure ramp parameters.
Phase 1
VPP generators details
Calculate total power capacity (C), and ideal VPP ramp rate. Order generators by ramp duration.
At 0 mins: activate generator group to achieve ideal VPP ramp rate.
Update generators schedule: ramp start time = 0 mins.
Order remaining inactive generators by increasing ramp rate.
Enter loop.
Phase 2
Increment time step.
Calculate VPP power at timestep.
Test: VPP power = C
Yes
Exit loop. Output generators schedule.
No
Test: VPP ramp - ideal ramp < minDiff
No
Yes
Schedule additional generator to: 1.) Make VPP ramp rate > addGenDiff else: 2.) Schedule generator with longest ramp
Update generators schedule: ramp start time = time step.
Process Flow
Data flow

Figure 17: Ramp tracking algorithm comprising two phases and tests that schedule additional generators to track ideal ramp rate

As displayed in Figure 17, Phase 1 calculates VPP ramp rate and orders the generators by longest ramp duration. Subsequently the algorithm schedules the minimum number of generators at 0 minutes that exceed the ideal VPP ramp rate. Finally, Phase 1 orders the other inactive generators by their increasing ramp rate before entering a time step loop.

Phase 2 time steps are 0.1 minutes, matching the rounding level of the rampTime values in the provided generator details. At each time step the algorithm tests the total VPP power and, if necessary, the difference in VPP ramp rate and ideal ramp rate. The group of ramping generators is unchanged until the difference between the actual ramp and ideal ramp falls below an algorithm parameter. Such an event occurs as the Phase 1 generators complete their individual ramps. The critical parameter is the minimum difference between actual and ideal ramp, minDiff in Table 5.

One of the inactive generators is scheduled to now ramp at the current loop time step. The scheduled generator is selected for its ramp rate that will produce a VPP ramp difference greater than the parameter addGenDiff. If the addGenDiff value is impossible, the inactive generator with the longest ramp time is selected.

The time steps increment until the VPP power reaches its maximum capacity (C). The loop breaks and reports a complete generator schedule of ramp times and VPP power level throughout the its ramp. The power level report enables calculation of the power difference (Pd_t) at each time step (t). The difference exists between a time step's actual power (Pd_t) and its concurrent ideal power (Pideal_t) due to the ideal VPP ramp (Eqn. 20). Summation of purely positive or purely negative Pd_t and scaling by time step size, converts the Pd_t to two energies in MWh. The energies are the areas of ramp differences plotted over time above and below the ideal ramp, referred to as "over-ramp" (21) and "under-ramp" (22). Note that 0.1 is the time step in minutes.

$$Pd_t = Pact_t - Pideal_t \quad \text{t: time step index} \qquad [MW] \tag{20}$$

$$\text{Over-ramp} = \sum Pd_t \times 0.1 \times \frac{1}{60} \quad where\, Pd_t > 0 \qquad [MWh] \tag{21}$$

$$\text{Under-ramp} = \sum Pd_t \times 0.1 \times \frac{1}{60} \quad where\, Pd_t < 0 \qquad [MWh] \tag{22}$$

2.6.2. Results

Combining two values of the three aforementioned parameters produces eight results (Table 6). Ramps 3 and 4 both track an ideal 60 minute ramp duration. The latter ramp displays two phase ramping due to increasing parameter minDiff from 0.02 to 0.1 (Figure 18, Ramp 4). The VPP ramp diverges from the ideal ramp during both phases.

Initially thirteen generators start ramping, without additional generators until 33.6 mins. A short dip occurs at 33.6 mins, when a second phase generator commences ramping. Increasing parameter minRampDiff=0.1 causes two drawbacks: higher generator cost by diverging from the ideal linear ramp and delayed ramp completion from 60.8 m to 64.3 mins.

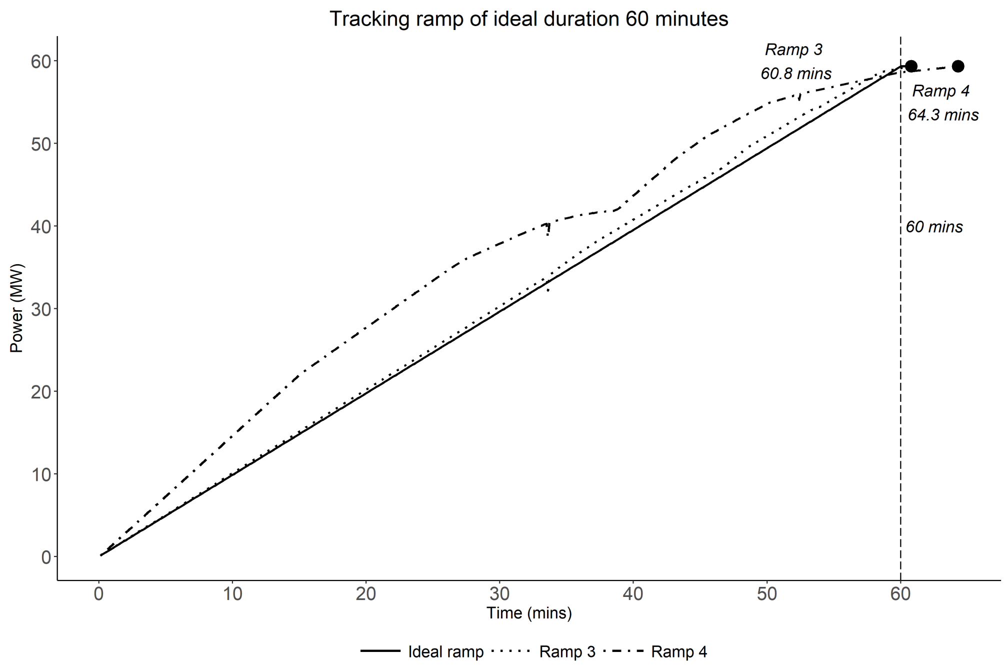


Figure 18: Ramps 3 and 4 of ideal duration 60 mins and minDiff of 0.02 and 0.1 respectively (Table 6)

Where the ideal VPP ramp time is 60 mins, increasing the addGenDiff from an initial 0.3 to 0.4, reduces ramping overruns (Table 6). As already displayed in Figure 18, Ramp 3 completes in 60.8 minutes; overunning by only 0.8 minutes. As expected, reducing the ideal VPP ramp time to 52.7 mins causes in larger overuns (Figure 19).

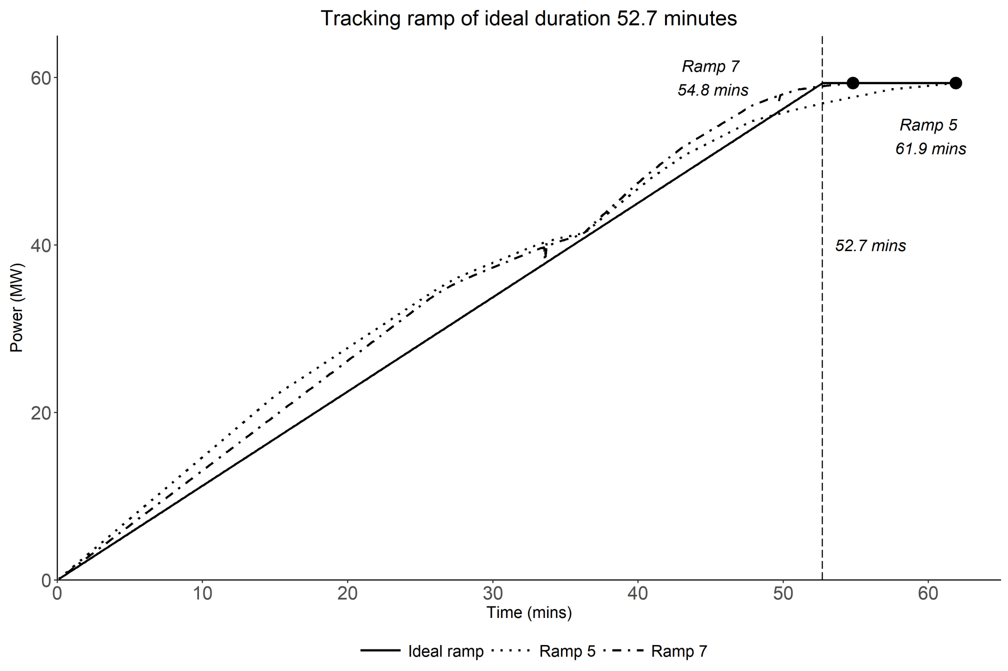


Figure 19: Ramps 5 and 7 of ideal duration 57.2 mins and addGenDiff of 0.3 and 0.4 respectively (Table 6)

In summary, the results favour a strategy of tracking the ideal ramp rate by sequencing the generators by descending ramp time. Parameter addGenDiff values above 0.4 achieve this intuitive sequence, when minDiff=0.02.

Table 6: Ramp tracking parameters and results, typical ideal time of 60 mins and minimum 52.7 mins

Ramp id	Ideal ramp time (mins)	minDiff (MW/min)	addGenDiff (MW/min)	Actual ramp time (mins)	Over-ramp (MWh)	Under-ramp (MWh)
1	60.0	0.02	0.30	65.75	0.56	-0.23
2	60.0	0.10	0.30	66.75	3.51	-0.12
3	60.0	0.02	0.40	60.79	0.70	0.00
4	60.0	0.10	0.40	64.25	5.26	-0.03
5	52.7	0.02	0.30	61.85	2.49	-0.22
6	52.7	0.10	0.30	58.14	3.18	-0.08
7	52.7	0.02	0.40	54.71	2.13	-0.01
8	52.7	0.10	0.40	58.81	4.68	-0.07

As expected, higher resolution fixed time steps increase computation time but achieve lower generator cost and VPP ramp time. The system.time() function, available in R statistics software (The R Foundation, 2018), reports the computation time using the provided details of twenty generators. A one minute time step computes in 0.27 s; better scheduling with a 0.1 minute time step still requires only 2.16 s. Computation times are measured on a 3.80 GHz, 28 GB RAM desktop PC. While no parallel computation is implemented, the “parallel” library function detectCores() finds 8 CPU cores.

2.6.3. Future Work

Future work would scale the number of VPP generators, for example, to sets one hundred or even one thousand and compare computation times. Larger sets of VPP generators may demonstrate more interesting variations in results due to parameters minDiff and addGenDiff.

The presented implementation of ramp tracking considers only the ramping characteristics of the VPP generators. Its algorithm assumes sufficient delay (d_i) has already occurred for the generator sites.

As a simulation, the ramp tracking technique enables user discretion (Lund et al., 2017). A qualified user could increase VPP over-ramp above the ideal ramp to protect again generator failure. Over-ramp reduces net costs where Eirgrid penalties exceed generator costs and generators underperform their ramping. Thereby, a user would be better informed if the algorithm converted the cost and fines caused by the over- and under-ramp in Table 6 to financial values.

The current calculation of energy differences due to over- or under- ramping are simple multiplications of a single power value by the 0.1 minute time step (21, 22). Numerical integration using the Trapezoidal Rule may increase accuracy (Weisstein, 2018); although any improvement is limited by the already small time step.

A R script implements the ramp tracking algorithm without expensive simulation software. Base R libraries suffice, although the tested code called libraries: “magrittr”, “dplyr” and “readr”. R’s “parallel” library resides in base packages, and provides the aforementioned detectCores() function (Gordon, 2015).

An implementation of ramp tracking in python promises faster simulations.

3. Conclusion and future work

We have provided six different methods to tackle the problem of optimising the scheduling of distributed generation to achieve linear aggregate response. The first method discussed (Gradient Descent) is theoretically very interesting but is unlikely to provide a global minimum due to the existence of many local minima. This, however, was exploited in Section 2.1 to address the equity challenge. Using the objective function J_2, the gradient descent method works very well and gives a number of different sensible optimal solutions. A disadvantage of this method is that it does not provide sensible results when the objective function does not have local optima, which is the situation with the objective function J_3.

The second method discussed (Mixed Integer Linear Programming) finds an optimal solution (subject to a time discretisation) due to the fact that it minimises the objective functions (15) and (16). Fast solution are obtained for 20 generators. However, there are questions to how well this approach will do with a large number of generators.

The four Heuristic Algorithms discussed work well with the objective function J_3 and provide good solutions with respect to the ‘eyeball norm’ (or quantitatively based on the computed $\tilde{J}_3$) very quickly. However, optimal solutions are unguaranteed. Their full potential should be evident if we combined these methods with the two direct methods. In particular, the simulated annealing method should combine well with the gradient descent method. Just as other Heuristic Algorithms could be used to specify initial conditions for a simulated annealing approach, the output of a simulated annealing method would work well as the initial condition for a gradient descent method.

As for future work, we recommend to: (i) Evaluate quantitatively the results from the heuristic and direct methods (also taking into account computational costs). (ii) Investigate local minima. Do these solve the equity challenge? (a partial positive answer was found and discussed for the Gradient Descent method). (iii) Combine techniques - for example, to take an initial seed for Gradient Descent from a heuristic result. (iv) Study the scalability of each method with respect to the number of sites.

References

Barzilai, J., Borwein, J., 1988. Two-point step size gradient methods. IMA Journal of Numerical Analysis 8, 141–148.

Gordon, M., 2015. How-to go parallel in R basics + tips.
URL `https://www.r-bloggers.com/how-to-go-parallel-in-r-basics-tips/`

Lund, H., Arler, F., Østergaard, P. A., Hvelplund, F., Connolly, D., Karnøe, P., 2017. Simulation versus Optimisation: Theoretical Positions in Energy Modelling. Energies 10 (7).
URL `http://www.dx.doi.org/10.3390/en10070840`

Rao, R., Iyengar, S., 1994. Bin-packing by simulated annealing. Computers and Mathematics with Applications 27 (5), 71–82.

The R Foundation, 2018. R: The R Project for Statistical Computing.
URL `http://www.r-project.org/`

Weisstein, E. W., 2018. Trapezoidal Rule. From MathWorld - A Wolfram Web Resource.
URL `http://mathworld.wolfram.com/TrapezoidalRule.html`

Appendix A. Sample data provided by Captured Carbon

Captured carbon provided sample data from 20 sites. The data from the i-th site consists of: capacity C_i (in MW), response time R_i (in minutes), ramping time τ_i (in minutes) and price in euro per unit MWh of energy γ_i.

The following table summarises these parameters.

Table A.7: Parameters provided by Captured Carbon.

Site number i	Capacity C_i (MW)	Response time R_i (min)	Ramping time τ_i (min)	Price γ_i (euro/MWh)
1	0.68	2.73	13.17	3.07227
2	3.16	6.13	49.61	2.93315
3	3.74	10.71	39.00	1.50213
4	1.68	6.91	34.51	1.35757
5	4.32	1.78	35.49	3.76508
6	3.89	11.34	28.52	4.24047
7	1.74	11.23	43.04	3.73992
8	4.92	9.00	15.14	2.85404
9	1.02	1.51	4.17	1.59351
10	4.80	9.48	33.20	2.46992
11	4.07	7.31	27.11	4.01328
12	3.66	13.69	33.50	3.89666
13	4.36	4.99	25.94	2.67428
14	2.42	7.21	50.21	3.71732
15	4.33	0.56	11.59	1.61575
16	1.74	5.32	6.59	3.02192
17	3.80	13.55	21.51	4.03933
18	0.62	8.22	27.74	2.31835
19	0.97	10.32	34.94	4.31203
20	3.40	13.94	25.76	3.84887

Appendix B. Calculation of the Hessian of objective function $J_2(\boldsymbol{d}, T_0, T_1)$

The Hessian of $J_2(\boldsymbol{d}, T_0, T_1)$**.** Regarding the Hessian of J_2, from equations (11)–(12) we readily calculate the second derivatives

$$\begin{aligned}\frac{\partial}{\partial d_i}\frac{\partial}{\partial d_j}J_2(\boldsymbol{d}, T_0, T_1) &= \frac{C_j}{\tau_j}\frac{\partial}{\partial d_i}\left[\int_{d_j+R_j}^{d_j+R_j+\tau_j}\left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right]dt\right]\\ &= \frac{C_j}{\tau_j}\delta_{ij}\left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right]\Big|_{t=d_j+R_j}^{t=d_j+R_j+\tau_j}\\ &+ \frac{C_j}{\tau_j}\int_{d_j+R_j}^{d_j+R_j+\tau_j}\frac{\partial}{\partial d_i}\left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right]dt\,, \quad i,j = 1,\dots,n\,.\end{aligned}$$

Now,

$$\frac{\partial}{\partial d_i}\left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right] = (1+K)\left[\frac{\partial}{\partial d_i}E(\boldsymbol{d}, T_0, T_1, t)\right]\delta\left(E(\boldsymbol{d}, T_0, T_1, t)\right)\,,$$

where $\delta(E)$ is the Dirac delta distribution. But we know $\frac{\partial}{\partial d_i}E(\boldsymbol{d}, T_0, T_1, t) = -\frac{C_i}{\tau_i}f'\left(\frac{t-d_i-R_i}{\tau_i}\right)$, so

$$\frac{\partial}{\partial d_i}\left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right] = -\frac{(1+K)C_i}{\tau_i}f'\left(\frac{t-d_i-R_i}{\tau_i}\right)\delta\left(E(\boldsymbol{d}, T_0, T_1, t)\right)\,,$$

which gives

$$\begin{aligned}\frac{\partial}{\partial d_i}\frac{\partial}{\partial d_j}J_2(\boldsymbol{d}, T_0, T_1) &= \frac{C_j}{\tau_j}\delta_{ij}\left[\Theta\left(E(\boldsymbol{d}, T_0, T_1, t)\right) - K\Theta\left(-E(\boldsymbol{d}, T_0, T_1, t)\right)\right]\Big|_{t=d_j+R_j}^{t=d_j+R_j+\tau_j}\\ &- \frac{(1+K)C_iC_j}{\tau_i\tau_j}\int_{d_j+R_j}^{d_j+R_j+\tau_j}f'\left(\frac{t-d_i-R_i}{\tau_i}\right)\delta\left(E(\boldsymbol{d}, T_0, T_1, t)\right)dt\,, \quad i,j = 1,\dots,n\,.\end{aligned}$$

Now, the latter integral can be simplified by first noticing that the support of $f'\left(\frac{t-d_i-R_i}{\tau_i}\right)$ is the interval $[d_i{+}R_i, d_i{+}R_i{+}\tau_i]$ where it takes the value 1. Defining $V_k = [d_k{+}R_k, d_k{+}R_k{+}\tau_k]$, we get then,

$$\int_{d_j+R_j}^{d_j+R_j+\tau_j}f'\left(\frac{t-d_i-R_i}{\tau_i}\right)\delta\left(E(\boldsymbol{d}, T_0, T_1, t)\right)dt = \int_{V_i\cap V_j}\delta\left(E(\boldsymbol{d}, T_0, T_1, t)\right)dt\,.$$

Now, the Dirac delta singles out the zeros of E. There is a formula that holds for functions with simple zeros: assume $E(\boldsymbol{d}, T_0, T_1, t)$ as a function of the variable t has simple zeros

$z_m,\ m = 1, \ldots, M$. Then

$$\delta(E(\boldsymbol{d}, T_0, T_1, t)) = \sum_{m=1}^{M} \frac{\delta(t - z_m)}{\left|\frac{\partial}{\partial t} E(\boldsymbol{d}, T_0, T_1, t)\right|_{t=z_m}} .$$

From equation (10), it is straightforward to compute the derivative of $E(\boldsymbol{d}, T_0, T_1, t)$ with respect to t. We get

$$\frac{\partial}{\partial t} E(\boldsymbol{d}, T_0, T_1, t) = \sum_{k=1}^{n} \frac{C_k}{\tau_k} f'\left(\frac{t - d_k - R_k}{\tau_k}\right) - \frac{C}{T_1 - T_0} f'\left(\frac{t - T_0}{T_1 - T_0}\right) .$$

These formulas establish that the Hessian is finite (although it may be discontinuous as a function of the variables), thus validating the gradient descent method and its basic algorithm.

We will not perform the explicit calculation of these integrals because there are several special cases that would need to be considered, such as: (i) when V_i or V_j is the first starting ramping interval or the last ending ramping interval; (ii) when the time derivative is discontinuous at one of the relevant zeroes of E (this can happen when the zero coincides with either a starting point of a ramp or an ending point of a ramp).

Appendix C. Code for Gradient Descent Method

Below is a Mathematica Notebook for the Gradient Descent Method.

CODE WITH L_1-TYPE COST FUNCTION AND FAST: GRADIENT DESCENT ALGORITHM THAT FIXES T_0 AND T_1
Written by Miguel D. Bustamante
University College Dublin, May 2019

```
In[1]:= ngen = 20

In[5]:= capacity = {0.68, 3.16, 3.74, 1.68, 4.32, 3.89, 1.74,
          4.92, 1.02, 4.8, 4.07, 3.66, 4.36, 2.42, 4.33, 1.74, 3.8, 0.62, 0.97, 3.4}

In[6]:= rampup = {13.17, 49.61, 39.00, 34.51, 35.49, 28.52 , 43.04, 15.14,
          4.17, 33.20, 27.11, 33.50, 25.94, 50.21, 11.59, 6.59, 21.51, 27.74, 34.94, 25.76}

In[7]:= notice = {2.73, 6.13, 10.71, 6.91, 1.78, 11.34, 11.23,
          9, 1.51, 9.48, 7.31, 13.69, 4.99, 7.21, 0.56, 5.32, 13.55, 8.22, 10.32, 13.94}

In[8]:= prices = {184.3363397, 175.9889665, 90.12779155, 81.45440258, 225.9046999, 254.4279989,
           224.3952205, 171.2422458, 95.61057852, 148.1954481, 240.7965139, 233.799561, 160.4570989,
           223.0390969, 96.94509111, 181.3151872, 242.3598063, 139.1008322, 258.7217771, 230.9322294} / 60

In[9]:= tstart[dvars_] := notice + dvars

In[10]:= tend[dvars_] := notice + rampup + dvars

In[11]:= framp[t_] := Piecewise[{{0, t ≤ 0}, {t, 0 < t ≤ 1}, {1, 1 < t}}]

In[12]:= cost[x_] := Piecewise[{{-1, x ≤ 0}, {kPenalty, x > 0}}]

In[13]:= costNew[x_] := Piecewise[{{-1, x ≤ 0}, {0, x > 0}}]

In[14]:= kPenalty = 0.1

In[16]:= Plot[framp[t], {t, -1, 2}]

In[17]:= Plot[cost[x], {x, -1, 1}]

         Clear[T0fixed, T1fixed]

In[837]:= T0fixed[dvars_] := Min[notice + dvars]

In[838]:= T1fixed[dvars_] := Max[notice + rampup + dvars]

In[15]:= SetAttributes[framp, Listable]
```

Error function:

```
In[25]:= error[dvars_, T0_, T1_, t_] := capacity.framp[(t - dvars - notice) / rampup] - Total[capacity] framp[(t - T0) / (T1 - T0)]
```

Vertices and errors contain the important points (x,y) = (vertex, error) where the error function changes slope:

```
In[48]:= vertices[dvars_, T0_, T1_] := Sort[Join[tstart[dvars], tend[dvars], {T0, T1}]]

In[74]:= errors[dvars_, T0_, T1_] :=
          Chop[Table[error[dvars, T0, T1, vertices[dvars, T0, T1][[ii]]], {ii, Length[vertices[dvars, T0, T1]]}]]
```

Zeros contain the times zn where the error function is zero: The end points are always zeros and the may be points in between:

```
In[210]:= zeros[dvars_, T0_, T1_] := Module[{verticesList, errorsList, zerosList},
            Sort[Join[{vertices[dvars, T0, T1][[1]], vertices[dvars, T0, T1][[-1]]}, verticesList = vertices[dvars, T0, T1];
              errorsList = errors[dvars, T0, T1];
              zerosList = {};
              Do[If[errorsList[[ii + 1]] errorsList[[ii]] < 0,
                AppendTo[zerosList, (-errorsList[[ii + 1]] verticesList[[ii]] + errorsList[[ii]] verticesList[[ii + 1]]) /
                  (errorsList[[ii]] - errorsList[[ii + 1]])]], {ii, 2, Length[verticesList] - 2}];
              zerosList]]]
```

Calculation of cost function J2

```
In[834]:= J2[dvars_, T0_, T1_] := Module[{verzererrList}, (verzererrList =
              Join[Transpose[{vertices[dvars, T0, T1], errors[dvars, T0, T1]}], Thread[{zeros[dvars, T0, T1], 0}]] // Union;
                                                  1
            Map[If[# < 0, -kPenalty * #, #] &, — Table[(verzererrList[[ii + 1, 2]] + verzererrList[[ii, 2]])
                                                  2
                  (verzererrList[[ii + 1, 1]] - verzererrList[[ii, 1]]),
                 {ii, Length[verzererrList] - 1}]] // Total) - 0 * dvars.(prices * capacity) / Total[prices]]
```

Calculation of gradient of cost function J2

```
In[835]:= gradJ2[dvars_, T0_, T1_] := Module[{zerosList, newVertexList, errorTemp}, (zerosList = zeros[dvars, T0, T1];
            Join[-0 * (prices * capacity) / Total[prices] -
              capacity / rampup * Table[newVertexList = Join[{(dvars + notice)[[ii]], (dvars + notice + rampup)[[ii]]},
                    Select[zerosList, (dvars + notice)[[ii]] ≤ # ≤ (dvars + notice + rampup)[[ii]] &]] // Union // Sort;
                                                          1
                Sum[errorTemp = error[dvars, T0, T1, — (newVertexList[[jj + 1]] + newVertexList[[jj]])];
                                                          2
                 (newVertexList[[jj + 1]] - newVertexList[[jj]]) (HeavisideTheta[errorTemp] -
                    kPenalty * HeavisideTheta[-errorTemp]), {jj, Length[newVertexList] - 1}], {ii, ngen}],
               Total[capacity]
             {———————————————— * (newVertexList = Join[{T0, T1}, Select[zerosList, T0 ≤ # ≤ T1 &]] // Union // Sort;
                (T1 - T0)^2
                                                          1
                Sum[errorTemp = error[dvars, T0, T1, — (newVertexList[[jj + 1]] + newVertexList[[jj]])];
                                                          2
                                                               1
                 (newVertexList[[jj + 1]] - newVertexList[[jj]]) (— (newVertexList[[jj + 1]] + newVertexList[[jj]]) - T0)
                                                               2
                  (HeavisideTheta[errorTemp] - kPenalty * HeavisideTheta[-errorTemp]), {jj, Length[newVertexList] - 1}])}])]
```

Must not do this (evaluating then replacing):

```
In[664]:= J2[dvarsTemp - α gradJ2Temp[[1 ;; -2]], T0temp, T1temp - α gradJ2Temp[[-1]]] /. α → 0.
```

Algorithm of gradient descent that does not change T_1, only **dvars** are changed:

Initial and end times (fixed):

```
In[999]:= T0temp = T0fixed[0]; T1temp = T1fixed[0];
```

```
In[1186]:= T0temp = 0; T1temp = 60;
```

Testing one random initial seed:

Initial seed:

```
dvarsTemp = 50 * Table[.5 - Random[], {i, ngen}]
```

Pre-step using line-search classical gradient method:

```
αmin = (ftemp = Interpolation[
      Table[{α, J2[dvarsTemp - α gradJ2[dvarsTemp, T0temp, T1temp][[1 ;; -2]], T0temp, T1temp]}, {α, -5, 5., .5}]];
   FindMinimum[{ftemp[α], -5 ≤ α ≤ 5}, {α, 0}][[2, 1, 2]]);
dvarsTempNew = dvarsTemp - αmin * gradJ2[dvarsTemp, T0temp, T1temp][[1 ;; -2]];
```

```
In[1001]:= Plot[{J2[dvarsTemp - α gradJ2[dvarsTemp, T0temp, T1temp][[1 ;; -2]], T0temp, T1temp], ftemp[α]},
            {α, (-5 + αmin) / 2, (5 + αmin) / 2}, PlotStyle → {Blue, Red}]
```

Barzilai-Borwein gradient descent method:

```
In[1017]:= AbsoluteTiming[niter = 100;
             dvarsSeries = Table[Table[0, {igen, ngen}], {in, niter - 2}];
             αSeriesNOT1 = Table[0, {in, niter - 2}];
             PrependTo[αSeriesNOT1, αmin];
             PrependTo[dvarsSeries, dvarsTempNew];
             PrependTo[dvarsSeries, dvarsTemp];
             gmTempOld = gradJ2[dvarsSeries[[1]], T0temp, T1temp][[1 ;; -2]];
             Do[gmTempNew = gradJ2[dvarsSeries[[im]], T0temp, T1temp][[1 ;; -2]];
              dgmTemp = gmTempNew - gmTempOld;
                                          (dvarsSeries[[im]] - dvarsSeries[[im - 1]]).dgmTemp
              αSeriesNOT1[[im]] = 1. * ──────────────────────────────────────────────────────;
                                                         Norm[dgmTemp]^2
              dvarsSeries[[im + 1]] = dvarsSeries[[im]] - αSeriesNOT1[[im]] * gmTempNew;
              gmTempOld = gmTempNew, {im, 2, niter - 1}]]

In[1028]:= costPlot = ListPlot[Table[Total[prices] * J2[dvarsSeries[[iin]], T0temp, T1temp] -
                1 * dvarsSeries[[iin]].(prices * capacity) + 1 * (prices * capacity).(T1temp - notice - 0.5 rampup),
               {iin, niter}], AxesLabel → {"Iteration", "Cost  (€)"}, ImageSize → Large];
           Manipulate[{Show[Plot[Evaluate[error[dvarsSeries[[in]], T0temp, T1temp, t]],
                {t, Min[vertices[dvarsSeries[[in]], T0temp, T1temp]], Max[vertices[dvarsSeries[[in]], T0temp, T1temp]]},
                PlotRange → All, AxesLabel → {"Time", "Power Error at Iteration " <> ToString[in]}], ListPlot[Transpose[
                  {vertices[dvarsSeries[[in]], T0temp, T1temp], errors[dvarsSeries[[in]], T0temp, T1temp]}], PlotStyle → Red],
               ListPlot[Thread[{zeros[dvarsSeries[[in]], T0temp, T1temp], 0}], PlotStyle → Green], ImageSize → Large],
             Plot[Evaluate[{error[dvarsSeries[[in]], T0temp, T1temp, t] + Total[capacity] framp[(t - T0temp) / (T1temp - T0temp)],
                 Total[capacity] framp[(t - T0temp) / (T1temp - T0temp)]}],
               {t, Min[vertices[dvarsSeries[[in]], T0temp, T1temp]], Max[vertices[dvarsSeries[[in]], T0temp, T1temp]]},
               ImageSize → Large, AxesLabel → {"Time", "Power at Iteration " <> ToString[in]}], Show[costPlot,
               ListPlot[{{in, Total[prices] * J2[dvarsSeries[[in]], T0temp, T1temp] - 1 * dvarsSeries[[in]].(prices * capacity) +
                   1 * (prices * capacity).(T1temp - notice - 0.5 rampup)}}, PlotStyle → Green]]}, {in, 1, niter, 1}]
```

Testing several initial seeds:

Initial seed table:

```
In[•]:= dvarsIniTable = Table[50 * Table[.5 - Random[], {i, ngen}], {jj, 30}]

        dvarsIniTable =
         {{22.22119129790904`, 3.3780201611528087`, 4.883296245632668`, 18.63977956390988`, -7.3534207709472446`,
           -11.683368764506419`, -9.968647732137036`, -11.85665042568087`, 0.49417700867071535`, -9.682040355497179`,
           -7.6983753135243305`, -9.15892536004092`, -21.86817119596261`, 18.765774626822143`, -13.031063340237715`,
           6.0110995142554104`, -20.409120467927583`, -15.593190336308915`, -1.4760985463690868`, 21.216247821483446`},
          {5.038764047336325`, -8.866680644350922`, -4.409515337534592`, -17.084294345495994`, 7.817572749427285`,
           12.755299194496267`, 15.70718841683274`, -10.724073909405874`, -9.829006479625468`, -0.5613320409973144`,
           0.6758361489697801`, -23.867423483725002`, 14.676816511703816`, -15.879291685500135`, -16.62578853750589`,
           10.29150187631592`, 11.544987707666426`, -9.645066312322282`, 21.405274802731824`, -20.71959763793949`},
          {6.954108175594007`, -19.051875976013367`, -2.1186266508990914`, -16.93584545942294`, -23.08465587174232`,
           14.814804668337555`, -22.7091113133645`, -24.851551113926945`, -5.902228621169603`, -22.940494526158712`,
           -13.41629973019724`, 10.872522795478925`, -21.073222141544136`, 2.620837514838606`, 10.907864120832983`,
           9.739946279203929`, -10.750038653247952`, -6.499870799661256`, 2.5336526583388723`, 24.448444402888008`},
          {2.7049736390856225`, -21.854804487338974`, 6.128377855607045`, 20.168042040827498`, 20.750865463491618`,
           22.197071488674393`, -16.752995493493867`, 12.103887500250435`, 18.835521335233935`, -17.61773317966317`,
           -19.043884180129368`, 11.955438614177377`, -0.26225004359646253`, -19.67723865350446`, 19.372415550067874`,
           -23.917084181301547`, -4.189027902052328`, 2.7019238316569356`, -16.53544857076511`, -8.657030460505478`},
          {-18.438989248804372`, -15.798205368681806`, 5.930898770896017`, -8.105474863393486`, 3.8560371121100054`,
           -18.943400881342832`, 24.802520915288973`, -3.2735169042209797`, 8.10517164861839`, -16.14047237001722`,
           16.555516408782836`, 9.622595595528585`, 14.269650313384455`, -23.522739190354052`, 10.599400588912204`,
           22.667156981351205`, -10.468099643019086`, 21.154499463150405`, 16.226985038844333`, 21.584241162652752`},
          {18.720928259033244`, -6.547424368506532`, 7.762433609609443`, 5.241271623158231`, 12.159917507837617`,
           -15.749218999824727`, -23.168465161286576`, -11.653253513448286`, -16.69611960427239`, -21.80581811848189`,
           -22.970986076575546`, 16.620263390772692`, 0.19870874710922193`, 19.33465425153533`, -14.526502485358384`,
           -18.002332204755888`, 10.929058433724768`, 17.857393441889386`, -0.12590307427058356`, -15.669489186107095`},
          {-3.6028419232561504`, 21.70289397873898`, 8.647111886885082`, -12.25373034875985`, 2.676229817710607`,
           3.250318347245515`, -24.11532172272436`, 7.504998028081921`, 15.516312309872992`, -6.000462652929761`,
           24.053143438562216`, -5.841748458469792`, 7.212431914145381`, -9.194644534447871`, 22.024129515137762`,
           2.5379881507575113`, -17.986276832963842`, -3.5292987859831983`, 11.550632000496144`, -4.459679644486597`},
          {-3.915335266688608`, 3.6133077721274165`, -13.323464925233269`, -13.790190458379502`, 24.687506656567543`,
 6.910413793388434`, 3.0294231878816467`, 23.463539890380346`, -2.9887231611430645`, -21.33990455385708`,
 2.1447449106060086`, -9.041458137701575`, 6.494964528983943`, 9.66055809907268`, 3.0916014720437923`,
 21.800290320768216`, 24.282532614838566`, -6.144797366479448`, 6.067471956906031`, -5.737697829989297`},
{17.268809447802404`, 22.38450141950375`, 19.516839956409886`, 23.721981814497305`, -3.81585528550899`,
 -6.228806352623667`, 7.840304881643156`, 12.512172272876807`, -3.5033619420765327`, 11.860779853987898`,
 -20.18911830623849`, 14.048632382496459`, 24.485361219066533`, 8.20068440784498`, 2.666136783155501`,
 -1.9099094798019634`, -7.009603309917411`, 23.540126308772297`, 24.57453531111171`, 1.2898001994298163`},
{-6.292135924755976`, 4.684923675251748`, -6.492936645794323`, -17.97250197058089`, 1.4390546274416187`,
 7.300422255747999`, -1.0097766022042098`, -16.69448378507819`, -19.745090087049388`, -11.470771391628338`,
 16.149918516152635`, -4.206656057955`, 8.758271855027141`, 1.668448754383764`, 11.339036822391124`,
 6.744711559548541`, 9.27291063596061`, 18.467764346538786`, -16.32709996076438`, -16.34537896064949`},
{-8.717486054121975`, 19.92763803776649`, -15.901635271876085`, 7.364820839920691`, 22.574649870634`,
 -9.757285637485257`, 15.591301373918236`, 0.33732281050158075`, -3.8644047568076214`, 7.9422921067667405`,
 -8.398922023877548`, -7.9681934044202265`, -9.119314669758227`, -5.586936501604923`, 0.45115945996981877`,
 21.238462653534768`, 7.122413475214628`, 17.744614744011315`, 14.112122637578695`, -10.50624890601377`},
{22.84950283925402`, 24.27685039747253`, 5.439222598343071`, -19.160869945364283`, 6.566988893375994`,
 -20.65078764029396`, -3.6591421297808404`, -1.5256907852849722`, 8.992339022741994`, 14.106497997191303`,
 5.749556496300917`, 23.13698640421345`, -12.143256220450388`, -18.83579410957544`, -10.85152147982153`,
 6.105179808633676`, 21.976058449307843`, 11.751142392029482`, 13.69731906020865`, 9.866717155098906`},
{-10.14635502590679`, 19.006527648018167`, 24.585196422629956`, -4.627033938887321`, -7.995857865160804`,
 19.72967725054564`, -5.854026175713117`, -10.466163993523036`, 10.437153241463202`, 15.3804648908396`,
 22.805115954067727`, 16.059526791761936`, -23.55518578127879`, -23.7260331063517`, -7.9444405422331945`,
 17.922540387548487`, 13.588070439171595`, 20.109761003223735`, -22.09291906241166`, -13.18263942108519`},
{16.612011989863756`, -16.641381388805744`, -10.79023812262031`, 1.9506434238159036`, 1.7583670157705429`,
 -10.647909036823911`, -10.375434545250267`, -18.42232263729678`, -15.24577511906865`, -5.377586287369551`,
 20.47859163046285`, 17.04384135622626`, -0.6829283605318537`, 4.241948821790848`, 22.673475676395125`,
 -24.015685435535673`, -2.127742579253061`, 2.9679819281425517`, 5.6179162186283165`, -16.938225823084156`},
{9.28418698157534`, 7.8582209249188155`, 2.710835281039978`, 21.24441359800103`, 17.672174991711586`,
 -0.5003976862754389`, -11.49892659633971`, -5.706229825814874`, -9.08619202405896`, -14.852488649451528`,
 23.876507948910557`, -12.283907188518095`, -18.840416904990303`, 15.525097637918025`, -21.60208368155229`,
 -4.327748544744359`, 6.842511455541548`, -13.716851183872825`, -19.27555935794742`, -5.312063109208687`},
{-16.02974596520539`, 8.315166887984624`, 0.10652442342426738`, -13.373837286124523`, -0.31393294678072614`,
 -24.54305403693419`, 22.39568914238429`, -9.618250884125557`, 7.013892061507687`, 0.9573436493412496`,
 8.894615738724`, 21.087978941689318`, -8.899915914433354`, -9.190167701207224`, 10.01810778981344`,
 8.371886130207418`, -15.05949900944305`, 0.2847346608747525`, 6.620191471365732`, -12.300365325048224`},
{3.0979895350154023`, -10.998414155252423`, 0.8957508293131522`, 18.011697784160464`, -5.872264499779211`,
 5.686418956762951`, -24.210773594111117`, 6.385535070284987`, 19.44166844700152`, 5.229472993697143`,
 -21.606462736495402`, -8.996214045589458`, -12.572223614506168`, -20.72787065564411`, -5.501078475219401`,
 -5.084192987278779`, 21.327692299927186`, 13.462297045563115`, 9.480813734967159`, 11.543920882513802`},
{11.387191309370237`, -11.822437615311637`, -22.139377736398576`, -1.1557137924379735`, -16.710798225645163`,
 24.175976539940784`, 1.9648714342882734`, 5.832588423401564`, 14.161466274134044`, -6.5104424168221655`,
 1.1756450283993884`, 24.44705335311658`, 19.719797827132524`, 13.260084589480691`, -2.2178922351052064`,
 8.443267398706036`, 7.292021441638691`, 8.987955245124802`, -21.716813759885806`, -11.472539614015181`},
{10.964329141711504`, 20.525658199561686`, -6.197627494852965`, 1.9835395034710157`, 24.57713783234127`,
 7.348095814873326`, -9.058249758454389`, -21.86074670409101`, 16.28793605798643`, 8.172119274932541`,
 13.97687880725734`, -2.693335127492574`, -22.873530216147614`, -10.317438308245297`, -12.19876622114205`,
 -2.1403884806091567`, -17.593328043280138`, 1.422477102274014`, 15.019126013963158`, 14.416344120684807`},
{0.11465051508117163`, 17.43452185714921`, 11.735939773848965`, 0.8888837346999912`, 14.150321373369668`,
 21.908863657587524`, -7.066432731298066`, 23.905344231228977`, 14.573183541028401`, -10.4392321572858`,
 -23.008182972843684`, 20.766090935319987`, 23.28524748304197`, 6.38864856778166`, -11.98506178010102`,
 -1.5405739371874339`, 21.15877769918958`, -8.293913123973045`, -24.78629555895897`, -24.40018545657828`},
{13.752105742469723`, 15.283609773752943`, -14.805421572922128`, -13.816529577263086`, -11.36254477261145`,
 22.84908791660373`, -1.5413613467710974`, 10.294586688036922`, -0.5128661459811168`, -24.059775740983795`,
 -19.47492861547303`, 11.389242456807946`, 9.91395031299048`, 11.379456416302006`, -21.46674564262935`,
 15.62315152148796`, 11.628702829948512`, -20.009192151479656`, 15.518316137471672`, -7.836274541324606`},
{15.469925130758927`, 13.28472097249339`, 15.304611696430639`, -8.436089084746323`, -23.28218061171079`,
 23.00111119874045`, 5.110033269352771`, -19.619559507483235`, 13.08036416090066`, -24.84797671786328`,
 -18.348605383876134`, -4.914146195520158`, -11.406769693118225`, 24.211799023120516`, -23.873676768403108`,
 8.696611347671897`, 3.679279993891296`, -12.167657393181491`, 22.593068874226244`, 18.073459826183942`},
{17.050577163942783`, -17.158465241701837`, -17.925247263245424`, 0.9097343675085467`, -23.419347966816144`,
 -5.443186214195228`, -8.229858959676067`, -15.65417654774513`, 24.86283264489465`, -3.444297412935682`,
 11.660107770971164`, -21.034617040261892`, -13.217531516006009`, -3.5963206950724027`, 5.008713154847297`,
 8.87952915525826`, 23.189238177112212`, -2.80811971819292`, 3.8823899232504013`, -24.817082192413636`},
{-5.490041816779084`, -15.640462325011429`, 6.289321049024157`, -17.89054201859758`, 2.4593810192781342`,
 -23.48199708330959`, -0.7854316877304146`, 6.199723613893876`, 0.8787289860942793`, 6.961189130885634`,
 -17.555572728054354`, -3.146099838360994`, 1.0158963411996318`, -14.594513456178687`, -4.215680499025515`,
  -7.1114827980991`, -10.766572142794356`, 14.001807238893715`, 15.775606346127185`, 9.008988046642639`},
 {-8.95581031990657`, -8.190073042913365`, -13.106783577123215`, 8.826070239056275`, 21.534231496872515`,
  -17.54961071790194`, 5.603895373852627`, 1.7166122576538516`, -5.92514952240562`, -19.067613634592345`,
  -18.610672938416954`, 20.516888643759977`, 18.1961214915001`, -1.0288027654779763`, 23.944899789637393`,
  -1.3370115178790276`, -7.819774849699529`, -11.43428930929929`, 3.1605802886629117`, -19.225528719779934`},
 {-22.05320270690517`, -0.4360965481930046`, 12.384973942535723`, -3.2345167664225682`, 11.902607613001399`,
  -17.246023505279638`, 0.4917575196589391`, 12.939412994521154`, 15.368376116128884`, -24.696412787377696`,
  19.887862145806313`, -13.777199263132701`, -3.706474361465495`, 19.37120084721464`, 13.498535084223267`,
  -9.294087906892672`, 3.097404147034408`, -4.599996387307382`, 14.55363529458587`, 17.042923610986353`},
 {-14.082821003266066`, -18.165707078008086`, -13.606944994077041`, 11.268452330766287`, -17.029618296360894`,
  7.270389470184916`, -0.9919189366127623`, -10.497030902811144`, -3.932225909362291`, -0.4835870245354479`,
  23.516323543728294`, 1.5635561026677014`, 5.699397974508824`, -0.787174237157745`, -21.371538602078015`,
  -9.659244634199604`, -15.594127664025686`, 4.84162491562761`, -9.870073686301279`, 24.6348432726930 7`},
 {6.308468188939909`, -15.558378697065011`, 0.5762910191128512`, -17.408080338293285`, -4.608710807794026`,
  -22.39267161905692`, -10.816763986810107`, -3.6765326690595734`, -12.579092511433137`, -4.663061089241838`,
  15.175154949802655`, -18.17950176624843`, 16.353133397929152`, 20.820525935293606`, 16.65883140607436`,
  5.256942131083872`, -14.34626457657967`, -3.392299827548645`, 13.03037000815237`, -10.083813234716526`},
 {-23.752136912553983`, 16.766075256823743`, -2.0995563055463515`, -9.718656507409602`, -5.060605101493892`,
  7.324453953888755`, 22.324152675340798`, -17.310576169116317`, 24.548105706300134`, 4.717125572945677`,
  8.14091666215091`, 11.365956499943255`, 12.127198217733268`, -15.619813337812483`, 17.965761712348254`,
  4.545458266191682`, 20.774064819804114`, -11.440339273106092`, -23.693069693726105`, 24.28851613510781`},
 {10.12032939638378`, 16.951960554442554`, -11.723439701878474`, 9.372329369824337`, 8.872466308937769`,
  -24.81411470238119`, 15.376116603667878`, -5.909014122766065`, -11.066928589568336`, -7.138568656269944`,
  18.051963928327076`, -13.59843795364975`, -10.615034295868469`, 13.144305770784376`, -15.088952733823835`,
  0.035605546406994315`, 2.2577674863982597`, 3.7641191085968613`, -8.05471444617209`, 20.49014728021531`}};
```

Pre-step uses line-search classical gradient method; all subsequent steps use the Barzilai-Borwein gradient descent method:

```
In[∘]:= dvarsSeriesTable = {};
        αSeriesTable = {};
        Do[dvarsTemp = dvarsIniTable[[dini]];
         αmin = (ftemp = Interpolation[
              Table[{α, J2[dvarsTemp - α gradJ2[dvarsTemp, T0temp, T1temp][[1 ;; -2]], T0temp, T1temp]}, {α, -10, 10., .5}]];
            FindMinimum[{ftemp[α], -10 ≤ α ≤ 10}, {α, 0}][[2, 1, 2]]);
         dvarsTempNew = dvarsTemp - αmin * gradJ2[dvarsTemp, T0temp, T1temp][[1 ;; -2]];
         niter = 200;
         dvarsSeries = Table[Table[0, {igen, ngen}], {in, niter - 2}];
         αSeriesNOT1 = Table[0, {in, niter - 2}];
         PrependTo[αSeriesNOT1, αmin];
         PrependTo[dvarsSeries, dvarsTempNew];
         PrependTo[dvarsSeries, dvarsTemp];
         gmTempOld = gradJ2[dvarsSeries[[1]], T0temp, T1temp][[1 ;; -2]];
         Do[gmTempNew = gradJ2[dvarsSeries[[im]], T0temp, T1temp][[1 ;; -2]];
          dgmTemp = gmTempNew - gmTempOld;
                                    (dvarsSeries[[im]] - dvarsSeries[[im - 1]]).dgmTemp
          αSeriesNOT1[[im]] = 1. * ----------------------------------------------------;
                                                   Norm[dgmTemp]^2
          dvarsSeries[[im + 1]] = dvarsSeries[[im]] - αSeriesNOT1[[im]] * gmTempNew;
          gmTempOld = gmTempNew, {im, 2, niter - 1}];
         AppendTo[dvarsSeriesTable, dvarsSeries];
         AppendTo[αSeriesTable, αSeriesNOT1];, {dini, Length[dvarsIniTable]}]
```

Making table of minimum costs throughout the different searches:

```
In[∘]:= (mincostTable = Table[dvarsSeries = dvarsSeriesTable[[dini]];
            {dini, Table[Total[prices] * J2[dvarsSeries[[iin]], T0temp, T1temp] - 0 * dvarsSeries[[iin]].(prices * capacity) +
                0 * (prices * capacity).(Max[vertices[dvarsSeries[[iin]], T0temp, T1temp]] - notice - 0.5 rampup),
               {iin, niter}] // Min}, {dini, Length[dvarsSeriesTable]}]) // ListPlot
```

Creating histogram of lowest costs found:

```
In[∘]:= Histogram[mincostTable[[All, 2]], 18,
         PlotLabel → "Histogram of Minimum Cost (€) over 30 Local Optima",
         BaseStyle -> {FontSize → 14}, PlotTheme → "Marketing"]
```

Selecting those samples with low costs: must set up the threshold manually depending on the above plot:

```
In[•]:= threshold = 351; Select[mincostTable, #[[2]] < threshold &][[All, 1]]
```

Making histogram of times spent from 0 to maximum capacity:

```
In[•]:= Histogram[Table[Max[tend[dvarsSeriesTable[[dini, -1]]]] - Min[tstart[dvarsSeriesTable[[dini, -1]]]],
          {dini, Select[mincostTable, #[[2]] < threshold &][[All, 1]]}], 20,
         PlotRange → {{59.5, 80}, All}, PlotLabel → "Histogram of times from zero to maximum capacity",
         BaseStyle -> {FontWeight -> Bold, FontSize → 14}, AxesLabel → {"Time \n (min)", "Count"}, ImageSize → Medium]
```

Making plots of histograms of starting times for each of the sites:
(WARNING: these plots are saved directly on the chosen directory. Uncomment in order to save.)

```
In[•]:= (*SetDirectory["/Volumes/GoogleDrive/My Drive/Captured Carbon/Report/Figs"]*)
```

```
In[•]:= (*siteTimes=Table[tstart[dvarsSeriesTable[[dini,-1]]]-Min[tstart[dvarsSeriesTable[[dini,-1]]]],
            {dini,Select[mincostTable,#[[2]]<threshold&][[All,1]]}];
         Do[filename="site"<>If[jj<10,"0",""]<>ToString[jj]<>"hist.pdf";
          Export[filename,Histogram[siteTimes[[All,jj]],12,PlotLabel→"Start Times Site "<>ToString[jj],
            BaseStyle->{FontSize→14},AxesLabel→{"Time \n (min)","Count"},ImageSize→Medium]],{jj,ngen}]*)
```

Cost curves for each of the seeds tried:

```
In[•]:= costPlotTable = Table[dvarsSeries = dvarsSeriesTable[[dini]];
          ListPlot[Table[Total[prices] * J2[dvarsSeries[[iin]], T0temp, T1temp], {iin, niter}], AxesLabel → {"Iteration",
             "Cost (€)"}, ImageSize → Large, BaseStyle -> {FontSize → 14}], {dini, 1, Length[dvarsSeriesTable], 1}]
```

Manipulate command showing, per sample seed, per iteration, the following: (i) power production error (with its vertices and zeroes), (ii) power production, and (iii) cost curve and its convergence.

```
In[•]:= Manipulate[dvarsSeries = dvarsSeriesTable[[dini]];
         costPlot = ListPlot[Table[Total[prices] * J2[dvarsSeries[[iin]], T0temp, T1temp] - 0 * dvarsSeries[[iin]].(prices *
                 capacity) + 0 * (prices * capacity).(Max[vertices[dvarsSeries[[iin]], T0temp, T1temp]] - notice - 0.5 rampup),
            {iin, niter}], AxesLabel → {"Iteration", "Cost (€)"}, ImageSize → Large, BaseStyle -> {FontSize → 14}];
         Manipulate[{Show[Plot[Evaluate[error[dvarsSeries[[in]], T0temp, T1temp, t]], {t, Min[vertices[
                dvarsSeries[[in]], T0temp, T1temp]], Max[vertices[dvarsSeries[[in]], T0temp, T1temp]]}, PlotRange → All,
             AxesLabel → {"Time", "Power Production Error (MW) at Iteration " <> ToString[in]}, BaseStyle -> {FontSize → 14}],
            ListPlot[Transpose[{vertices[dvarsSeries[[in]], T0temp, T1temp], errors[dvarsSeries[[in]], T0temp, T1temp]}],
             PlotStyle → Red], ListPlot[Thread[{zeros[dvarsSeries[[in]], T0temp, T1temp], 0}], PlotStyle → Green],
            ImageSize → Large], ListPlot[{Table[{t, capacity.framp[(t - dvarsSeries[[in]] - notice) / rampup]}, {t, vertices[
                dvarsSeries[[in]], T0temp, T1temp]}], Table[{t, Total[capacity] framp[(t - T0temp) / (T1temp - T0temp)]},
              {t, vertices[dvarsSeries[[in]], T0temp, T1temp]}]}, Joined → {True, True}, PlotStyle → {Automatic, Dashed},
            ImageSize → Large, PlotLabel → "Power Production at Iteration " <> ToString[in],
            AxesLabel → {"Time \n (min)", "Power(MW)"}, BaseStyle -> {FontSize → 14}], Show[costPlot,
            ListPlot[{{in, Total[prices] * J2[dvarsSeries[[in]], T0temp, T1temp] - 0 * dvarsSeries[[in]].(prices * capacity) +
                0 * (prices * capacity).(Max[vertices[dvarsSeries[[in]], T0temp, T1temp]] - notice - 0.5 rampup)}},
             PlotStyle → Green]]}, {in, 1, niter, 1}], {dini, 1, Length[dvarsSeriesTable], 1}]
```

Appendix D. Code for MILP

Appendix D.1. GAMS code for MILP

```
sets
i sites /i1*i20/
t timesteps /t1*t50/

Parameters
C(i) Capacities,
Gamma(i) Costs,
Ramptime(i) Ramp time,
R(i) Ramp rate;

*import data
$call gdxxrw.exe inputs_CC.xlsx par=inputs rng='Sheet1'!a1:e21  MaxDupeErrors»
=10
parameter inputs
$GDXIN inputs_CC.gdx
$load inputs
$GDXIN
;

*populate data
C(i)=inputs(i,'dummy1');
R(i)=inputs(i,'dummy2');
Ramptime(i)=inputs(i,'dummy3');
Gamma(i)=inputs(i,'dummy4');

*define binary variables
Binary Variables
b(i,t) =1 if generating at time t and zero otherwise
b_tilda(i,t) =1 if generating at max capacity at time t and zero otherwise
;

Positive Variables
gen(i,t) generation from site i at time t
below(t) distance below line at time t
upper(t) distance above line at time t
;

Variables
o objective function value;

Equations
Obj objective function equation
Energy(i,t) energy constraint
MaxC(i,t) maximum capacity constraint
On(i,t) Once begining genearting cosntraint
On2(i,t) Once at max capacity constraint
Ramping(i) ramping time constraint
Line(t) Line cosntarint
;

Obj.. o=e= sum((i,t),gamma(i)*gen(i,t))
+(sum(i,gamma(i)/card(i) ))*sum(t,upper(t)+10*below(t));

Energy(i,t).. gen(i,t)=e=gen(i,t-1)+R(i)*(b(i,t) -b_tilda(i,t) );
MaxC(i,t).. gen(i,t)=l=C(i);
On(i,t)..  b(i,t)=g=b(i,t-1);
On2(i,t)..  b_tilda(i,t)=g=b_tilda(i,t-1);
Ramping(i).. Sum(t,b(i,t)-b_tilda(i,t))=e=floor(Ramptime(i));
Line(t).. sum(i,gen(i,t))+below(t)-upper(t)=e=ord(t)*sum(i,C(i))/card(t);

*define model
```

```
Model M /all/

*solver options
OPtion optcr=0;
OPtion optca=0;

*solve model
Solve M using mip minimising o;



```

Appendix D.2. Python code for MILP

In the implementation in Python, the condition 17e was replaced with the equivalent set of conditions

$$\overline{b_{i,t}} = b_{i,t-\tau_i}, \ \ \forall i, \forall t \geq \tau_i \tag{D.1}$$

$$\overline{b_{i,t}} = 0, \ \ \forall i, \forall t < \tau_i, \tag{D.2}$$

as this was more time efficient. There is no generation at time $t = 0$ for all generators.

$$gen_{i,0} = 0, \ \ \forall i. \tag{D.3}$$

Condition 17b was replaced with

$$gen_{i,t} = gen_{i,t-1} + R_i\big(b_{i,t} - \overline{b_{i,t}}\big) + (C_i - floor(\tau_i) * R_i)\big(\overline{b_{i,t}} - \overline{b_{i,t-1}}\big), \ \ \forall i, t. \tag{D.4}$$

The last term adds a small amount to $gen_{i,t}$ when generator i is almost at full capacity, but the addition of R_i would mean that $gen_{i,t} > C_i$. This is needed as time is measured in minutes in the Python code, while in reality this is not the case.

```
import pandas as pd
# Pulp package for MILP
from pulp import *
import numpy as np
import matplotlib.pyplot as plt
import time

data = pd.read_csv("C:/Users/Susan/Documents/SitesData.csv", index_col=0)

#Parameters
C = data.loc['Final_Expected_Output'].astype(float) #Capacity

RR = data.loc['Ramp_Rate:_Up'].astype(float) #Ramp rate

P = data.loc['Price'].astype(float) #Price

RT = data.loc['Notice_Period'].astype(float) #Response Time

tau = C/RR #Ramp time

#Need to use ceiling and floor of ramp time since we are using minutes but ramp time
    from data is in fraction of minutes.
tau_ceil = np.ceil(tau)

tau_floor = np.floor(tau)

#T = T1-T0. In this case T is the longest ramp time.
T = max(np.ceil(C/RR)).astype(int)
```

```
T_no_rounding = max(C/RR)

Num_gen = len(C); #Number of generators

Pnew = np.tile(P,T+1) #used in cost function

K = 10 #Factor by which underproducing is penalised relative to overproducing

t0 = time.time()
prob = LpProblem("GeneratorSchedule",LpMinimize)

gen_indices = np.arange(0,((T+1)*Num_gen))
time_indices = np.arange(0,T+1)

#Definition of variables and constraints on these variables.
#0<=gen_i,t<=C_i
gen = LpVariable.dicts("gen",gen_indices,0)
for t in np.arange(0,T+1):
    for i in np.arange(0,Num_gen):
        gen[t*Num_gen+i].lowBound = 0
        gen[t*Num_gen+i].upBound = C[i]
#below_t,above_t >=0
below = LpVariable.dicts("below",time_indices,0)
above = LpVariable.dicts("above",time_indices,0)
#bstart,bstop binary
bstart = LpVariable.dicts("bstart",gen_indices,cat=LpBinary)
bstop = LpVariable.dicts("bstop",gen_indices,cat=LpBinary)


#Cost function
prob +=sum([gen[i]*Pnew[i] for i in gen_indices]) + P.mean()*sum([K*below[i] + above[i]
     for i in time_indices])


for i in np.arange(0,Num_gen):
    #Need generator to stop ramping by time T
    prob += bstop[(T)*Num_gen + i] ==1
    #No generation at t=0
    prob += bstart[i]==0
    for t in np.arange(0,tau_floor[i]+1):
        #Generator i can't have stopped before time tau_i
        prob += bstop[t*Num_gen + i] == 0
    for t in np.arange(tau_floor[i],T+1):
        #bstop_t = bstart_(t-tau)
        prob += bstop[t*Num_gen+i] == bstart[(t-tau_floor[i])*Num_gen + i]
    for t in np.arange(np.floor(T_no_rounding-tau[i])+1,T+1):
            #Generator i needs to start by time T-tau_i to be at full capacity at time T
            prob += bstart[t*Num_gen+i]==1


#Generation at time t
for i in np.arange(0,Num_gen):
    for t in np.arange(1,T+1):
        prob += gen[t*Num_gen + i] == gen[(t-1)*Num_gen + i] + RR[i]*(bstart[t*Num_gen +
             i]-bstop[t*Num_gen + i]) +(C[i] - (tau_floor[i])*RR[i])*(bstop[t*Num_gen +
            i]-bstop[(t-1)*Num_gen + i])


#bstart, bstop 1 at t => 1 at t+1
for t in np.arange(1,T+1):
    for i in np.arange(0,Num_gen):
        prob += bstart[t*Num_gen + i] >=  bstart[(t-1)*Num_gen + i]
        prob += bstop[t*Num_gen + i] >=  bstop[(t-1)*Num_gen + i]

#gen + above + below is linear
for t in np.arange(0,T+1):
    prob += sum([gen[t*Num_gen+i] for i in np.arange(0,Num_gen)]) + below[t] - above[t]
        == (t)*sum(C)/T


#Output of each generator is 0 at t=0
for i in np.arange(0,Num_gen):
    prob+= gen[i]==0

prob.writeLP("GeneratorSchedule.lp")


prob.solve()


print("Status:", LpStatus[prob.status])

t1=time.time()
t1-t0

#Get optimal parameter values, start times
gen_opt = np.empty([Num_gen,T+1])
bstart_opt = np.empty([Num_gen,T+1])
bstop_opt = np.empty([Num_gen,T+1])
below_opt = np.empty(T+1)
above_opt = np.empty(T+1)

for t in np.arange(0,T+1):
    below_opt[t] = below[t].varValue
    above_opt[t] = above[t].varValue
    for i in np.arange(0,Num_gen):
        bstart_opt[i,t] = bstart[t*Num_gen+i].varValue
        gen_opt[i,t] = gen[t*Num_gen+i].varValue
        bstop_opt[i,t] = bstop[t*Num_gen+i].varValue

start_times = (T-np.sum(bstart_opt,axis=1)).astype(int)
```

Appendix E. Matlab codes for cost calculation and direct simulated annealing

Appendix E.1. Matlab code for cost of a given choice of start times

This code takes a set of inputs include the times for starting ramping (i.e. $d_i + R_i$) for each site and uses it to calculate the overall cost, $\tilde{J}_3$ that must be minimised for an optimal solution. This code can be used to compare results from different solution methods.

```
% costOfSchedule.m
% 26 June 2018

% Function to calculate cost of given schedule

function ...
[totalCost, ...                Total cost of energy (in euros)
penalisedEnergy, ...           Energy associated with power below promised value to eirgrid
     (in MWmin)
eventTimes, ...                Times at which changes in power gradient occur (useful for
    plotting)
powerOutputs] ...              Power outputs at given event times
= costOfSchedule(...
tStartRamps, ...               Times for starting ramp for each site (in minutes)
rampRates, ...                 Ramp rate for each site (in MW/min)
finalPowers, ...               Final power outputs for each site (in MW)
pricePerMwmin, ...             Price of energy from sites (in euros per MWmin)
penaltyPerMwmin, ...           Penalty for failing to deliver power to eirgrid (in euros
    per MWmin)
totalRampTime) ...             Total time for ramping system (in minutes)


% Number of sites in problem
numSites = numel(tStartRamps);

% Overall ramp rate
totalRampRate = sum(finalPowers)/totalRampTime;

% Quick error checks
if numel(rampRates) ~= numSites
error('Vector_inputs_must_all_have_numSites_elements');
end
if numel(finalPowers) ~= numSites
error('Vector_inputs_must_all_have_numSites_elements');
end
if numel(pricePerMwmin) ~= numSites
error('Vector_inputs_must_all_have_numSites_elements');
end

% Finish times for different ramps
rampTimes = finalPowers./rampRates;
tFinishRamps = tStartRamps + rampTimes;

% More quick error checks
if any(tStartRamps < 0)
error('Negative_time_for_starting_ramp_given');
end
```

```
% THIS HAS BEEN CREATING ISSUES WITH SIMULATED ANNEALING ALGORITHM
%if any(tFinishRamps > totalRampTime)
%    error('Time for finishing a ramp is greater than total ramp time');
%end

% Create vector of all times where events take place, deleting times
% corresponding to repeated events.
eventTimes = sort([0; tStartRamps; tFinishRamps; totalRampTime]);
eventTimes(diff(eventTimes)==0) = [];
numEventTimes = numel(eventTimes);

% Create numEventTimes-by-numSites arrays which indicate whether ramping is
% finshed (finishedRampMatrix) or the time since ramping began in the case
% where ramping is currently occurring (rampingMatrix). Using these,
% construct a vector that contains the power output at all event times.
repeatedStartTimesCol = (tStartRamps*ones(1,numEventTimes))';
repeatedFinishTimesCol = (tFinishRamps*ones(1,numEventTimes))';
repeatedCurrTimesRow = eventTimes*ones(1,numSites);
finishedRampMatrix = (repeatedCurrTimesRow>=repeatedFinishTimesCol);
rampingMatrix = repeatedCurrTimesRow-repeatedStartTimesCol;
rampingMatrix(rampingMatrix<0) = 0;
rampingMatrix(finishedRampMatrix) = 0;

% Vector of length eventTimes containing total power output at each time.
powerOutputs = rampingMatrix*rampRates + finishedRampMatrix*finalPowers;

% Difference between ideal power output and actual power output
powerDiscrepancies = eventTimes*totalRampRate - powerOutputs;

% gapClassification refers to pairs of adjacent terms in
% powerDiscrepancies. If these are both negative, they do not contribute to
% the cost of the power output being less than promised to eirgrid (class
% 0); if one is positive and the other is negative, then some maths needs
% to be done to determine the total energy that is less than that
% promised to eirgrid (class 1 or 2); if both are positive, then the energy
% contribution less than promised to eirgrid is easily calculated from the
% sum of the beginning and ending power values.
%
% For the tricky case, we get energy = 0.5 * (timegap) * (positivePower)^2
% / (positivePower + |negativePower|).

signsOfPowerDiscrepancies = (powerDiscrepancies>0);
gapClassification = ...
2*signsOfPowerDiscrepancies(1:numEventTimes-1) ...
+ signsOfPowerDiscrepancies(2:end);

positivePowerDiscrepancies = zeros(numEventTimes-1,1);
positivePowerDiscrepancies(gapClassification==1) = ...
powerDiscrepancies([false; gapClassification==1]).^2 ./ ...
(powerDiscrepancies([false; gapClassification==1]) - ...
powerDiscrepancies([gapClassification==1; false]));
positivePowerDiscrepancies(gapClassification==2) = ...
powerDiscrepancies([gapClassification==2; false]).^2 ./ ...
(powerDiscrepancies([gapClassification==2; false]) - ...
powerDiscrepancies([false; gapClassification==2]));
```

```
positivePowerDiscrepancies(gapClassification==3) = ...
powerDiscrepancies([gapClassification==3; false]) + ...
powerDiscrepancies([false; gapClassification==3]);

% Energy that will be penalised by eirgrid.
penalisedEnergy = (positivePowerDiscrepancies')*diff(eventTimes)/2;

% Total cost including only terms that can vary with tStartRamps
totalCost = penaltyPerMwmin*penalisedEnergy - (pricePerMwmin')*(finalPowers.*tStartRamps
    );

% True total cost (including standing cost of providing the energy)
baseCost = 1/2*(pricePerMwmin')*(finalPowers.*(2*totalRampTime-rampTimes));
totalCost = baseCost + totalCost;
```

Appendix E.2. Matlab code for simulated annealing

This code describes the use of Matlab's inbuilt simulated annealing functions to find an optimal solution where we take $T_0 = \max_i(R_i + \tau_i) - \max_i(\tau_i)$ and $T_1 = \max_i(R_i + \tau_i)$. This will give a heuristic solution that aims to have the minimal total time from notification to complete power-up, and the minimum time spent powering up.

```
% scriptSimulatedAnnealing.m
% 27 Jun 2018

% Script m-file to read data from SitesData.csv file (made from top rows of
% Excel spreadsheet sent by CapturedCarbon), translate into a form usable
% by Matlab optimisation functions, and then use Matlab's inbuilt simulated
% annealing to calculate an optimum solution.
%
% An important output is totalCost, which gives a euro amount that includes
% both the cost of using all of the power generators and the penalty cost
% of going below the power promised to eirgrid.

%% Parameter specification

% Read data from SitesData.csv.
allSitesData = csvread('SitesData.csv',1,1);

% Rows of allSitesData
noticePeriods = allSitesData(1,:)';
rampRates = allSitesData(2,:)';
finalPowers = allSitesData(3,:)';
pricePerMwh = allSitesData(4,:)';

% Input penalty per MWhour of energy provided below power promised to
% eirgrid.
penaltyPerMwh = mean(pricePerMwh)*10;

% Translating data to more useful form
pricePerMwmin = pricePerMwh/60;
penaltyPerMwmin = penaltyPerMwh/60;
rampTimes = finalPowers./rampRates;

% Use the longest ramp time and shortest possible notice period as the ramp
% time and notice period for seeking solutions.
totalRampTime = max(rampTimes);
totalNoticePeriod = max(rampTimes+noticePeriods)-totalRampTime;

%% Simulated annealing

% Vectors of earliest and latest start times for ramping. Note that all
% times here are measured from the start of the first ramp time.
earliestRampStartTimes = max(zeros(size(rampRates)),...
noticePeriods-totalNoticePeriod);
latestRampStartTimes = totalRampTime - rampTimes;
if any (earliestRampStartTimes > latestRampStartTimes)
error('There is a site for which no feasible solution exists within the constraint of 
    given overall ramp time and notice period')
```

```
end

% Use midpoint for initial guesses
tStartRampsInit = (earliestRampStartTimes + latestRampStartTimes)/2;

% Create anonymous function for the cost associated with a given
% tStartRamps
costForRampStartTimes = @(t) costOfSchedule(t,rampRates,finalPowers,pricePerMwmin,
    penaltyPerMwmin,totalRampTime);

% Use Matlab's simulated annealing to find an optimum
tStartRamps = simulannealbnd(costForRampStartTimes,tStartRampsInit,
    earliestRampStartTimes,latestRampStartTimes);

%% Process results

% Calculate total cost of given schedule of starting times, and amount of
% energy that is associated with failing to meet power promised to eirgrid.
[totalCost, ...             Total cost of energy (in euros)
penalisedEnergy, ...        Energy associated with power below promised value to eirgrid
     (in MWmin)
eventTimes, ...             Times at which changes in power gradient occur (useful for
    plotting)
powerOutputs] ...           Power outputs at given event times
= costOfSchedule(...
tStartRamps, ...            Times for starting ramp for each site (in minutes)
rampRates, ...              Ramp rate for each site (in MW/min)
finalPowers, ...            Final power outputs for each site (in MW)
pricePerMwmin, ...          Price of energy from sites (in euros per MWmin)
penaltyPerMwmin, ...        Penalty for failing to deliver power to eirgrid (in euros
    per MWmin)
totalRampTime); ...         Total time for ramping system (in minutes)

% Calculate minimum required notification time to eirgrid.
%notificationToEirgrid = -min(tStartRamps - noticePeriods);
notificationToEirgrid = totalNoticePeriod;

% Calculate times to notify each site to begin [measured from beginning of
% notification to eirgrid]
timesToNotifySites = notificationToEirgrid + tStartRamps - noticePeriods;

%% Create figure for overall power output

% Create figure to show power output
figure
hold on
plot(eventTimes+notificationToEirgrid,powerOutputs,'b');
plot([0 notificationToEirgrid totalRampTime+notificationToEirgrid],[0 0 sum(finalPowers)
    ], 'r');
title('Power␣output␣in␣blue␣and␣goal␣power␣curve␣in␣red')
xlabel('Time␣since␣start␣of␣overall␣notification␣period')
ylabel('Power␣(MW)')
hold off

%% Create results and figure for cumulative power outputs
```

```
% Useful numbers and vectors
tFinishRamps = tStartRamps + rampTimes;
numEventTimes = numel(eventTimes);
numSites = numel(rampRates);

% Create numEventTimes-by-numSites arrays which indicate whether ramping is
% finshed (finishedRampMatrix) or the time since ramping began in the case
% where ramping is currently occurring (rampingMatrix).
repeatedStartTimesCol = (tStartRamps*ones(1,numEventTimes))';
repeatedFinishTimesCol = (tFinishRamps*ones(1,numEventTimes))';
repeatedCurrTimesRow = eventTimes*ones(1,numSites);
finishedRampMatrix = (repeatedCurrTimesRow>=repeatedFinishTimesCol);
rampingMatrix = repeatedCurrTimesRow-repeatedStartTimesCol;
rampingMatrix(rampingMatrix<0) = 0;
rampingMatrix(finishedRampMatrix) = 0;

% Matrix giving power output from each site and cumulative power from sites
powerPerSite = rampingMatrix*diag(rampRates) + finishedRampMatrix*diag(finalPowers);
cumulativePowerFromSites = cumsum(powerPerSite,2);

% Figure for cumulative power
figure
hold on
plot(eventTimes+notificationToEirgrid,cumulativePowerFromSites);
plot([0 notificationToEirgrid totalRampTime+notificationToEirgrid],[0 0 sum(finalPowers)
    ], 'k--');
title('Cumulative power outputs from different sites, target shown as dashed black line'
    )
xlabel('Time since start of overall notification period')
ylabel('Power (MW)')
```