Bus Charging Schedule Simulated Annealing with MILP Constraints

Alexander Brown

November 19, 2022

Contents

1 Introduction

This document outlines the simulated annealing (SA) approach to the bus charging scheduling problem utilizing Mixed Integer Linear Programming (MILP) constraints as the method of determining feasible charging schedules. The problem statement is as follows: given a set of routes for a fleet of Battery Electric Buses (BEB) and a set of fast and slow chargers, generate an optimal charging schedule to minimize the consumption cost (amount of electricity used over a certain time) and the demand cost (rate at which electricity is being used) within the constraints that the buses must maintain sufficient charge to complete the working day and do not have any delays in their respective routes.

Simulated Annealing (SA) shall be introduced and utilized as a means of finding optimizing. The SA algorithm shall be constrained by a set of Mixed Integer Linear Program (MILP) constraints derived from the Position Allocation Problem (PAP). These constraints are set in place to ensure validity of the proposed charging schedules. A set of objective functions describing consumption cost and demand cost, as stated above, shall be minimized to reduce power consumption and total cost of using the BEBs.

2 Problem Description

It is assumed that there are a total of I visits to the station by B buses. There are a total of Q queues for the bus at the station. Given a set of bus arrivals to a charging station $i \in \{1, ..., I\} = \mathcal{I} \subset \mathcal{Z}$ with a set of chargers to be queued $q \in \{1, ..., Q\} = \mathcal{Q} \subset \mathcal{Z}$ where the bus is indicated by an identification number $b \in \{1, ..., B\} = \mathcal{B} \subset \mathcal{Z}$. Each bus arrival, i, can be represented by the tuple: $(b_i, a_i, e_i, u_i, d_i, v_i, \eta_i, \xi_i)$, in which the ordered elements denote the bus identification number, b_i, arrival time to the station, a_i, departure time from the station, e_i, time to start charging, u_i, time to stop charging, d_i, the charger queue for the bus to be placed into, v_i, and the initial State Of Charge (SOC), η_i.

It is assumed that each visit occurs over the time horizon $T = \{t : t_0 \leq t \leq t_f\}$. The set of all arrivals is represented by the set $\mathbb{I} = \{\forall i \in \mathcal{I}\ (b_i, a_i, e_i, u_i, d_i, v_i, \eta_i, \xi_i) : b_i, \xi_i \in \mathcal{B}, a_i, e_i, u_i, d_i \in T, v_i \in \mathcal{Q}\}$. The concept of "arrivals" is derived from the PAP [7]. This idea of arrivals is useful in the sense that it is easy to describe the state of any arbitrary arrival; however, although a bus may revisit the station multiple times, the model assumes that each arrival is unique (i.e. no two bus arrives twice) therefore a system must be put in place to track each bus over each arrival. That is why a bus identifier is placed in the tuple, and in that way each bus can be tracked over each arrival.

For each arrival a bus must be placed in a singular queue, $v_i \in \mathcal{Q}$. The charger q is assumed to be either a fast or slow charger or no charger at all. A bus is only allowed to visit one queue per visit; however, there may be multiple buses charging simultaneously. The amount of time the bus is allowed to charge is dictated by the scheduled arrival time and required departure time, $[a_i, e_i]$. Although a bus must be placed in a queue, if a bus does not require much charge, or none at all, partial charges, or no charging, is allowed. It is not allowed for the bus to charge over its battery capacity limit. The battery charging rate is modeled as linear, which remains accurate up to about an SOC of 80% charge [5].

Each bus arrival, with the exception of the last arrival for each bus, has a paired "route" that the bus must perform. This route, as one would expect, causes the bus to discharge by some certain amount. This paper assumes an average discharge over a period of time where an estimated discharge is calculated for each route, Δ_i. The charge supplied while at the station is required to supply enough charge for each route (battery charge does not deplete to zero) with an additional battery capacity percentage, m, acting as a safety factor.

The scheduler's task shall be to schedule the set of arrivals $\mathbb{I}$ to fulfill the minimum charge requirements over the time horizon T as well as minimize the demand cost as well as minimize over the consumption cost. The objective function and constraints are discussed in further detail in section Subsection 3.2.

3 Optimization Problem

This sections introduces the problem in the form of the objective function as well MILP constraints. The objective function is required to allow comparisons between candidate solutions. In the context of this formulation, the objective function is broken down into two major components as alluded in the introduction: consumption cost, demand cost, assignment cost, as well as a penalty for undercharging a bus. The constraints ensure that candidate solutions are in the feasible region. They are composed of a series of equations defined by decision variables which are unknown variables that are manipulated in the attempt to optimize the objective functions and input variables predefined input variables that are assumed to be known. Furthermore, the decision variables have components that are directly and indirectly manipulated. This will be further discussed in Subsubsection 3.1.2.

Table 1: Table of variables used in the paper.

Variable	**Description**
Input constants	
C	Penalty method gain factor
B	Number of buses in use
H	Numer of discrete steps, h, in time horizon
I	Number of total visits
$J(u, d, v, \eta)$	Objective function
K	Local search iteration amount
M	Total number of steps created by initial temperature, T_0, and cooling schedule
Q	Number of chargers
$\mathcal{T}$	Time horizon
T	Temperature
Input variables	
Δ_i	Discharge of visit over after visit i
α_b	Initial charge percentage time for bus b
δ_i	Discharge rate for vehicle i per mile
ϵ_q	Cost of using charger q
κ_b	Battery capacity for bus b
ρ_i	Route distance after visit i
ξ_i	The next index bus b will arrive
a_i	Arrival time of visit i
b_i	ID for bus visit i
dt_h	Discrete time step $dt_h = t_h - t_{h-1}$
e_i	Time bus visit i must exit the station
k	Local search iteration k
m	Minimum charge percentage allowed for each visit
r_q	Charge rate of charger q
Decision Variables	
η_i	Charge for the bus at the beginning of visit i
ι_h	Binary variable that is 1 if $u_i \leq dt_h \leq d_i$, 0 othewise
μ_h	Binary variable that is 1 if $u_i \leq dt_h$, 0 otherwise
ϕ_i	Binary term to enable/disable charge penalty for visit i
ψ_{ij}	Tracks spatial overlap for visit pair (i, j)
σ_{ij}	Tracks temporal overlap for visit pair (i, j)
θ	Binary variable that is 1 if $d_i \geq dt_h$, 0 otherwise
d_i	Detach time from charger for visit i
$p_{dem}(t)$	Demand cost
s_i	Amount of time spent on charger for visit i (service time)
u_i	Time to start charging for visit i
v_i	Assigned queue for visit i

3.1 Parameter Definitions

This section defines the input variables and decision variables used by the system. The input variables are the parameters that are assumed to be known prior to optimizing the system. The decision variables are the values that the SA algorithm has the freedom to manipulate. The values produced by the SA algorithm will be interpreted as a candidate charging solution. This is further described in Section 4.

3.1.1 Input Variables

The input values of any MILP system are defined prior to the solving of the system. They specify initial conditions, known state properties, etc. Roughly following the order in Table 1, each variable will be introduced.

Δ_i is the amount power required to complete the bus route after visit i. Because there is no route after the last visit, the power consumed after the final visit is zero. Formally, let $\mathcal{J}_b \subset \mathcal{I}$ denote the set of visit indices for bus b, and let $\mathcal{J}_b^f$ denote the final visit index for bus b. Furthermore, let the set of final visits for each bus, b, such that $\mathcal{I}_f = \{x \in \mathcal{I}_f \subset \mathcal{I} : \forall b \in \mathcal{B}, x \in \mathcal{J}_b^f\}$. As mentioned before, the last visit for each bus has no discharge and can be written as $\Delta_{i \in \mathcal{I}_f} = 0$. The discharge for visit all visits $i \in \{\mathcal{I} \setminus \mathcal{I}_f\}$, is defined by $\Delta_i = \delta_i \cdot \rho_i$ where δ_i is the amount of energy consumed by the bus per mile and ρ_i is the route mileage after visit i. ρ_i is calculated using $\rho_i = \text{average mph} \cdot (a_{\xi_i} - d_i)$. For the final visits, $\rho_{i \in \mathcal{I}_f} = 0$. The average MPH is assumed to be 20 miles per hour.

α_b is the initial SOC percentage of bus b at the beginning of the working day. The initial SOC for bus b can be represented as shown in Equation 1 where κ_b is the battery capacity for bus b, $\eta_{i \in \mathcal{I}_0}$ indicates the initial charge for bus b where $\mathcal{I}_0 = \{x \in \mathcal{I}_0 \subset \mathcal{I} : \forall b \in \mathcal{B}, x \in \mathcal{J}_b^0\}$. The rest of the η_i terms are considered decision variables and will be further discussed in Subsubsection 3.1.2. dt_h is the discrete time step used for calculating the demand cost. ϵ_q is the cost for assigning a charger to queue q. This parameter is utilized by the objective function and is further discussed in Subsection 3.2. ξ_i represents the next arrival index for bus b_i. In other words, given a set of bus visit IDs $\{1, 2, 3, 1\}$, using a starting index of 1, $\xi_1 = 4$. This is found by first identifying tho ID of the first element in $\mathcal{I}_b$, which is 1. The next step is to find the next occurrenc of bus 1 in the sequence. In this exapmple, the next index for bus 1 is $i = 4$. a_i and e_i are the arrival and departure times of bus visit i to the station, respectively. k represents the local iteration search for the SA algorithm. This is further discussed in Section 4. Lastly, r_q represents the rate of charge for the charger in queue q.

$$\eta_{i \in \mathcal{I}_0} = \alpha_b \cdot \kappa_b, \tag{1}$$

3.1.2 Decision Variables

Decision variables are the defined by the optimizer and are therefore unknown prior to running the optimization algorithm. In this case the optimizer is SA. Once SA has been run, each of the decision variables will be specified and the fitness of the solution will be determined by the objection functions The variables will be broken into two sections: direct and indirect decision variables. Decision variables that are direct are values that the system has direct control over and indirect variables are those that are influenced by the direct.

Direct Decision Variables Decision variables that are direct are variables that can be immediately chosen by SA. The first two variables are u_i and d_i $\forall i \in \mathcal{I}$. They represent the initial and final charging times. These values must remain within range of the arrival time and departure time for visit i, $[a_i, e_i]$. μ_h and $theta_h$ are binary decision variables, $\mu_h, \theta_h \in \{0, 1\}$. $\mu_h = 1$ when $u_i \leq dt_h$, and is 0 otherwise. $\theta_h = 1$

when $d_i \geq dt_h$ and is 0 otherwise. The last direct decision variable is the queue that bus visit i can be placed in to charge, $v_i \in \mathcal{Q}$.

Indirect Decision Variables Indirect decision variables are variables that are dependent on direct decision variables. For example η_i is the initial charge for visit $i \in \mathcal{I} \setminus \mathcal{I}_f$. These variables are chained together per bus by using the bus identifier, b, and next arrival index, ξ_i. The initial charges must be chained so that the battery charge can be calculated per bus as it is charged and discharged over each visit, $[u_i, d_i]$. The concept of chaining the battery charges together is shown in Equation 2. The equation states that the charge for bus i's next visit is equal to the initial charge for visit i plus the charge added to it by charger v_i over duration $d_i - u_i$ minus the discharge accumulated over route i.

$$\eta_{\xi_i} = \eta_i + r_{v_i}(d_i - u_i) - \Delta_i \tag{2}$$

$\iota_h = 1$ when μ and θ are both active. ϕ_i is a boolean decision variable, $\phi_i \in \{0, 1\}$, that either enables or disables the charge penalty defined in Subsection 3.2. σ_{ij} and ψ_{ij} are used to indicate whether a visit pair (i, j) overlap the same space as show in Figure 1. A little more formally, Equation 3 describes the relationship that σ_{ij} and ϕ_{ij} uphold. That is, for every visit, if the start charge time of either visit i or j is greater than end charge time of the other, then σ_{ij} is active. Similarly, if the queue for visit i and j are different, then ψ_{ij} is active. These variables will be further elaborated on in Subsection 3.3.

$$\sigma_{ij} = \begin{cases} 1 & \text{if } u_i \geq d_j \\ 0 & \text{otherwise} \end{cases} \tag{3a}$$

$$\psi_{ij} = \begin{cases} 1 & \text{if } v_i \geq v_j \\ 0 & \text{otherwise} \end{cases} \tag{3b}$$

p_d is the demand cost of the overall charging schedule. It is calculated after all the decision variables have been assigned. This is further described in Subsection 3.2.

3.2 Objective Function

The objective function is used to compare the fitness of different candidate solutions against one another. This objective function takes in input and decision variables to calculate some value of measure. The calculated objective function value can either be maximized or minimized. The desired option is dependent on the problem to be solved as well as the formulation of said objective function. Let J represent the objective function. The objective function for this problem has four main considerations: charger assignment, consumption cost, demand cost, and sufficient charge.

Suppose the objective function is of the form min $J = AC(u, d, v, \eta) + PC(u, d, v)$. $AC(u, d, v, \eta)$ is the assignment cost, and $PC(u, d, v)$ is the power usage cost. The assignment cost represents the costs of assigning a bus to a particular queue as well as the chosen charging period, $[u_i, d_i]$, as shown in Equation 4. $v_i \in \mathcal{Q}$ is the charger index, u_i is the initial charge time, d_i is the detach time for visit i, ϕ_i is a binary decision variable, m is the minimum charge percentage allowed, κ_i is the battery capacity for visit i, and η_i is the charge for b_i when it arrives for visit i.

$$AC(u, d, v, \eta) = \sum_{i=1}^{I} \left(\epsilon_{v_i}(d_i - u_i) + \frac{1}{2} C \phi_i (\eta_i - m\kappa_i)^2 \right) \tag{4}$$

The first term in the summation represents the calculation of the cost for assigning a bus to queue q (i.e. cost of using the charger multiplied by the usage time). The second term is the penalty function that is either enabled or disabled by ϕ_i [6]. ϕ_i is enabled when the initial charge, η_i, is less than the allowed minimum charge, $m\kappa_{b_i}$. This is further discussed in Subsection 3.3. Note that the variables ϕ_i and η_i are both decision variables that are being multiplied together. This is called a bilinear term. Using a traditional MILP solver, this would require linearization [9]; however, because SA handles nonlinearities easily these bilinear terms will be ignored [8].

The demand cost quantifies the amount of power being used over a given period and adjusts the cost accordingly. The consumption cost calculates the total amount of power being consumed by the chargers. The consumption cost is merely the summation of all the energy being used over all the active periods for each charger in the time horizon as written in Equation 5. r_{v_i} is the charge rate for the active charger v_i and is multiplied by the time that the charger will be utilized, $d_i - u_i$.

$$\sum_{i=1}^{I} \Big(r_{v_i}(d_i - u_i) \Big) \tag{5}$$

The demand cost is calculated based on 15 minute increments (900 s). This cost is also referred to as the peak 15. The average power used over an arbitrary 15 minute interval is represented by Equation 6.

$$p_{15}(t) = 1/900 \int_{t-900}^{t} p(\tau) d\tau \tag{6}$$

Worst case must be assumed to always ensure enough power is supplied; therefore, the maximum value found is retained as represented in Equation 7.

$$p_{max}(t) = \max_{\tau \in [0,t]} p_{15}(\tau) \tag{7}$$

A fixed minimum threshold, p_{fix}, is introduced as a base power rate to be charged at. Let this fixed threshold be defined as p_{fix}. In a similar manner as p_{max}, the maximum value is retained. Furthermore, let s_r define the demand rate which has the units of $\frac{\$}{kW}$.

$$p_d(t) = \max(p_{fix}, p_{max}(t)) s_r \tag{8}$$

Equation 8, again, retains the largest p_{dem} value with a starting fixed value of p_{fix}. To write the total power demand at any discrete time, consider Equation 9. Let $\omega_h \in \omega$ be the discrete power demand at time step h where $h \in \{1, 2, ..., H\} \subset \mathcal{Z}$ and $H = \frac{T}{900}$. For conciseness of notation we will abuse t_h to denote the time in discrete form (as opposed to t being continuous), let $dt_h = t_h - t_{h-1}$, and $\mathcal{H} = \{1, 2, ..., H\}$. Let ι_h be a binary decision variable that is enabled if charger $v_i \in \mathcal{Q}$ is enabled in the time frame dt_h. Let the power usage of charger v_i be denoted as r_{v_i}.

$$\omega_h = \sum_{i}^{I} \iota_h \cdot r_{v_i} \tag{9}$$

The average power can be rewritten as $p_{15}^h = \omega_h$, p_d can be rewritten as $p_d^h = \max_{h \in \mathcal{H}}(p_{15}^h)$, there the h repesents the h^{th} step. Finally p_d^f can be written as Equation 10.

$$p_d^h = \max_{h \in \mathcal{H}}(p_{fix}, p_{max}^h) s_d \tag{10}$$

To write the power cost, Equation 5 and Equation 10 are superimposed to create Equation 11.

$$PC(u,d,v) = p_d + \sum_{i=1}^{I} \Big(r_{v_i}(d_i - u_i) \Big) \tag{11}$$

3.3 Constraints

Now that a method of calculating the fitness of a schedule has been established, a method for determining the feasibility of a schedule must be defined. Feasible schedules require that the schedule maintain a certain list of properties. These properties are enforced by a set of constraints derived from the MILP PAP. The constraints must ensure no overlap temporally or spatially, receives enough charge to complete route after each visit i, bus visit i cannot be over-charged, and departs on time. The aforementioned constraints are shown in Equation 12.

$$u_i - d_j - (\sigma_{ij} - 1)T \geq 0 \tag{12a}$$
$$v_i - v_j - (\psi_{ij} - 1)Q \geq 0 \tag{12b}$$
$$\sigma_{ij} + \sigma_{ji} \leq 1 \tag{12c}$$
$$\psi_{ij} + \psi_{ji} \leq 1 \tag{12d}$$
$$\sigma_{ij} + \sigma_{ji} + \psi_{ij} + \psi_{ji} \geq 1 \tag{12e}$$
$$\Delta_i = \delta_i \rho_i \tag{12f}$$
$$\eta_{\xi_i} = \eta_i + r_{v_i}(d_i - u_i) - \Delta_i \tag{12g}$$
$$\kappa_i \geq \eta_i + r_{v_i}(d_i - u_i) \tag{12h}$$
$$\eta_i - m\kappa_{b_i} \leq T(1 - \phi_i) \tag{12i}$$
$$m_{k_i} - m\kappa_{b_i} < T\phi_i \tag{12j}$$
$$a_i \leq u_i \leq d_i \leq e_i \leq T \tag{12k}$$
$$u_i - dt_h \leq T\theta_h \tag{12l}$$
$$dt_h - u_i \leq T(1 - \theta_h) \tag{12m}$$
$$dt_h - d_i \leq T\mu_h \tag{12n}$$
$$d_i - dt_h \leq T(1 - \mu_h) \tag{12o}$$
$$\iota_h = \mu_h \theta_h \tag{12p}$$

Constraints Equation 12a-Equation 12e are the "queuing constraints". They are used to prevent overlapping in both spatially and temporally as shown in Figure 1. The y-axis represents the possible queues for a bus visit to be placed into, and the x-axis represents the time that can be reserved for each visit. The shaded rectangles represent time that has been scheduled and the queue allocated for each bus visit. The set of constraints Equation 12a - Equation 12e aim to ensure that these shaded rectangles never overlap.

Constraint Equation 12e states that the starting service time for bus j, u_j, must be greater than the starting time of bus i, u_i, plus its service time, s_i. The last term utilizes big-M notation to activate or deactivate the constraint. A value of $\sigma_{ij} = 1$ will activate the constraint to ensure that i is complete before j is allowed to begin being serviced. If $\sigma_{ij} = 0$, then the constraint is of the form $T + u_j + s_j > u_i$ rendering the constraint "inactive" because u_i cannot be larger than d_i. This effectively allows the charging windows of the vehicle to overlap.

Similarly, ϕ_{ij} determines whether the vehicles will be charging in the same queue. If $\phi_{ij} = 1$ then (Equation 12b) is rendered active and vehicle i and j must be charging in different queues. If $\phi_{ij} = 0$ then the constraint is deactivated and the vehicle queue assignments may be the same.

Equation 12f calculates the discharge for the route after visit i. Equation 12g calculates the initial charge for the next visit for bus b_i. Equation 12h ensures that the bus is not being over-charged. Equation 12i and Equation 12j are used to enable and disable the penalty method in Equation 4. This is done by checking if the initial charge for visit i is greater than or equal to the minimum allowed charge. Equation 12k ensures the continuity of the times (i.e. the arrival time is less than the initial charge which is less than the detach time which is less than the time the bus exits the station and all must be less than the time horizon).

The last set of constraints Equation 12l - Equation 12p set the rules for the decision variable ι_h. Equation 12l and Equation 12m ensure that the initial charge time for visit i, u_i, is greater than the discrete time, $900 \cdot h$,

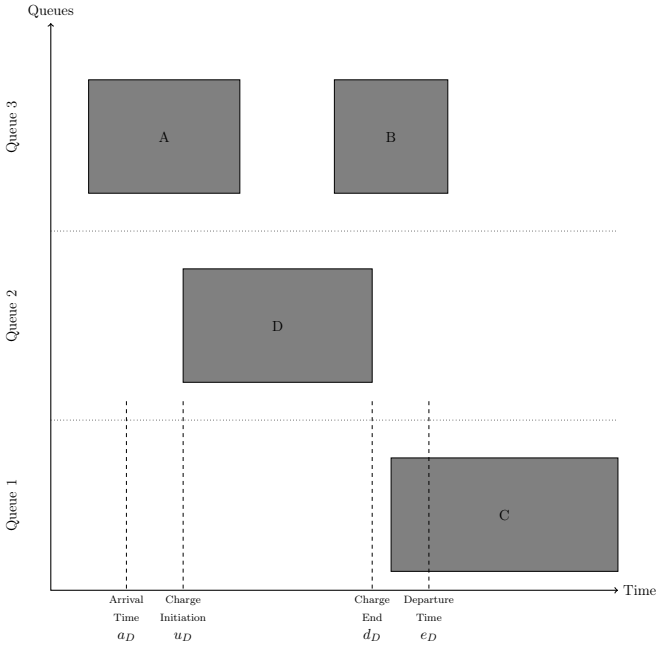

Figure 1: Visualization of the temporal and spatial aspect of the aspect of the schedule.

being checked. Similarly, Equation 12n and Equation 12o ensure that the final charge time, d_i, is less than the next discrete time step, dt_h.

4 Simulated Annealing

SA is an exploitation oriented, single-solution based (as compared to population based) metaheuristic approach in which its main advantage is its simplicity both theoretically and in its implementation as well its inherit ability to overcome nonlinearities [1, 8]. This model is named after its analogized process where a crystalline solid is heated then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuration [3]. There are five key components to SA: initial temperature, cooling schedule (temperature function), generation mechanism, acceptance criteria, local search iteration count (temperature change counter) [4].

4.1 Cooling Equation (Experimental)

The initial temperature and cooling schedule are used to regulate the speed at which the solution attempts to converge to the best known solution. When the temperature is high, SA encourages exploration. As it cools down (in accordance to the cooling schedule), it begins to encourage local exploitation of the solution [10, 3]. There are three basic types of cooling equations as shown in Figure 2 [4]. The different types merely dictate the rate at which the algorithm progressively disallows exploration. A linear cooling schedule is defined by Equation 13.

$$T[n] = T[n-1] - \Delta_0 \tag{13}$$

with $T[0] = T_0$ and $\Delta_0 = 1/2\ C^\circ$ in Figure 2. A geometric cooling schedule is mostly used in practice [4]. It is defined by Equation 14.

$$T[n] = \alpha T[n-1] \tag{14}$$

where $\alpha = 0.995$ in Figure 2. An Exponential cooling schedule is defined by the difference equation is defined as Equation 15.

$$T[n] = e^{\beta} T[n-1] \tag{15}$$

where $\beta = 0.01$ in Figure 2. The initial temperature, T_0, in the case of Figure 2, is set to $500^\circ\ C$ and each schedule's final temperature is $1\ C^\circ$.

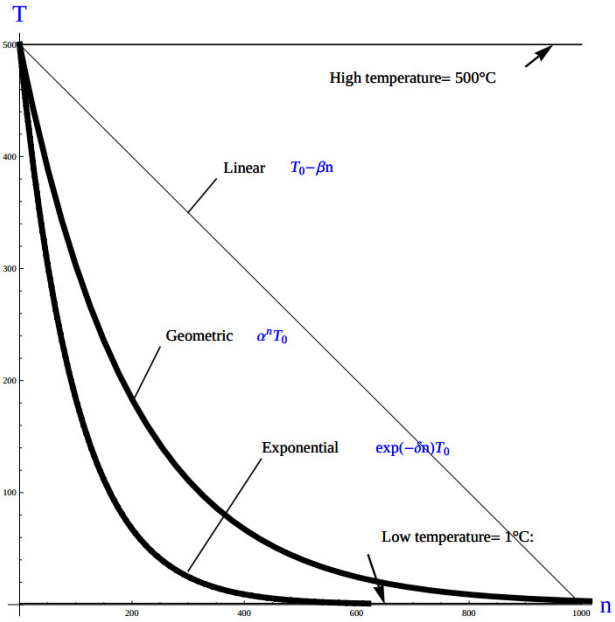

Figure 2: Cooling equations

4.2 Acceptance Criteria

Acceptance criteria describes the method to accept or reject a given candidate solution. In SA, if a new candidate solution is more fit than the currently stored solution it is always accepted as the new solution. However, within SA, worse candidate solutions may be accepted as the new solution. The probability of accepting the candidate solution is described by the function $\exp(-\frac{J(x)-J(x')}{T})$ where $J(\cdot)$ is the objective functions described in Subsection 3.2. The probability of acceptance is a function of the cooling equation just described and difference of the current solution and a new candidate solution. Let $\Delta E \equiv J(x) - J(x')$ where x is the current solution and x' is the new candidate solution. The probability of acceptance of x' is defined by Equation 16 [4].

$$f(x, x', T) = \begin{cases} 1 & \Delta E > 0 \\ e^{-\frac{\Delta E}{T}} & \text{otherwise} \end{cases} \tag{16}$$

4.3 Generation Mechanisms

Generation mechanisms in SA are used to generate random solutions to propose to the optimizer, these are known as candidate solutions. In the case of the problem statement made in Section 2, five generation mechanism shall be used: new visit, slide visit, new charger, remove, new window. The purpose of each of these generators is to assign new visits to a charger, adjust a bus visits initial and final charge time within

the same time frame/queue, remove a bus from a charger, and place a bus visit into a new time slot/queue. Each generator will be discussed in more detail in Subsubsection 4.3.2.

These generator mechanisms will in turn be utilized by three wrapper functions. The purpose of the route generation to create a set of bus route data to feed to the SA algorithm. Although, strictly speaking, is not a part of the SA algorithm. It is vital in specifying the initial conditions and "setting the stage" for the SA algorithm to solve. The schedule generation is to used create candidate solutions for SA to compare with other solutions, and the perturb schedule generator is used to take a candidate solution and alter it slightly in an attempt to fall into a global/local minimum.

4.3.1 Generator Input/Output

This section discusses in detail the expected inputs and output of each generator. It is important to discuss these parameters in order to have an understanding of the generating algorithms derived. The input consists of the bus visit index of interest, information about the current state of arrivals, $\mathbb{I}$, and the current state of the chargers' availability, $\mathbb{C}$. The output of each generator affects the tuple of decision variables $(v_i, u_i, d_i) \subset \mathbb{I}_i$.

Generator Input Each generator has the tuple input of $(i, \mathbb{I}, \mathbb{C})$ where i is the visit index, $\mathbb{I}_i$ is the tuple $(b_i, a_i, e_i, u_i, d_i, v_i, \eta_i, \xi_i)$ (Section 2), that describes the set of visits generated by the route generation algorithm (Section 4.3.3), and $\mathbb{C}$ is the set that describes the availability for all chargers $q \in \mathcal{Q}$. In other words, $\mathbb{C}$ defines the set of times when the chargers are not being utilized or are "inactive".

To derive $\mathbb{C}$, consider its inverse, $\mathbb{C}'$, which is the set of "active" time periods for each charger, $\mathbb{C}' = \bigcup\{\mathbb{C}'_q : q \in \mathcal{Q}\}$ where $\mathbb{C}'_q \subset \mathbb{C}'$ describes the active times for chagrer q. Focusing on an individual charger, consider $\mathbb{C}'_q$ before a schedule has been imposed upon it, $\mathbb{C}'_q \in \varnothing$. In other words, no buses have been assigned to be charged over the time period of $[u_i, d_i]$. After the scheduling process is complete, $\mathbb{C}'_q$ will have a set of active periods of the form $\mathbb{C}'_q \in \{[u_j, d_j] : j \in \mathbb{J}\}$ where $\mathbb{J} \subset \mathcal{I}$. For $\mathbb{C}'_q$ to be of value, its compliment is to be found, $\mathbb{C}_q$.

To determine the inverse of $\mathbb{C}'_q$, begin by noting $\mathbb{C}'_q \bigcap \{[u_j, d_j] : j \in \mathbb{J}\} = \varnothing$, is said to be disjoint [2]. The inverse of a disjoint set can be found by the De Morgan Law as shown in Equation 17. Using Equation 17, the set of inactive periods can be written as $\mathbb{C}_q \equiv \bigcup\{[u_j, d_j]' : j \in \mathbb{J}\}$. Let $\mathbb{S}$ denote the tuple $\mathbb{S} \equiv (i, \mathbb{I}, \mathbb{C})$.

$$(A \cap B)' = A' \cup B' \tag{17}$$

Generator Output The output is a modified visit, noted as $\mathbb{I}'_i$. In actuality only the decision variables are being altered which is a subset of $\mathbb{I}_i$. Let this subset be defined by $x'_i \equiv (v_i, u_i, d_i) \subset \mathbb{I}_i$ which is the chosen queue, initial charge time, and detach time from the generator, (v_i, u_i, d_i). The nature of SA implies that the generators have a sense of randomness. Because of that, some of the generators may have multiple choices for what x'_i may be. Let the set of candidates for the output be defined as $x'_i \in X'_i$. Furthermore, set the set of candidate visits be denoted as $\mathbb{I}'_i \in \mathbb{I}'$.

4.3.2 Generators

This section describes and outlines the algorithm pool for the different generator types that are utilized in the wrapper functions. Note that to satisfy constraints, B extra idle queues that provide no power to the bus. Because of this, the set of queues is fully defined as $q \in \{1, ..., Q, Q+1, ..., Q+B\}$ where Q is the total amount of chargers and b is the bus ID. The use case for this is for when a bus is not to be placed on

a charger, it will be placed in the queue, $v_i \in \{Q+1, ..., Q+B\}$, which will satisfy the constraints above while allowing the bus to be "set aside" while others charge.

New visit The new visit generator describes the process of moving bus b from the idle queue, $v_i \in \{Q+1, ..., Q+B\}$ to a valid charging queue, $v_i \in \{1, ..., Q\}$. Lines 2-4 extracts the visit, i, arrival time, a_i, and departure time, e_i. Note that in subsequent algorithms these lines will be omitted for conciseness. Line 5 loops through the inactive time periods that contain the time the bus is at the station, $[a_i, e_i]$. Line 6 verifies that the inactive period selected is valid and returns a random charging time, $[u_i, d_i]$, if it is. Line 7 returns the new visit.

Algorithm: New Visit
Input: $(\mathbb{S})$
Output: $\mathbb{I}'_i$

```
1  begin
2      i ← {i : i ∈ 𝕊} ;                                   /* The index of the visit i */
3      a_i ← {a_i ∈ 𝕀_i : 𝕀_i ∈ 𝕀 ⊂ 𝕊} ;                   /* Get the arrival time for visit i */
4      e_i ← {e_i ∈ 𝕀_i : 𝕀_i ∈ 𝕀 ⊂ 𝕊} ;                   /* Get the departure time for visit i */
       /* Randomly select a free time, j, from any charger q ∈ that is within the time frame [a_i e_i]  */
5      while C_q^j ∈ {[a_i, e_i]} ⊂ 𝒰_ℂ do
6          if x'_i ← findFreeTime(C_q^j, (a_i, e_i)) ∉ ∅ then   /* If there is time available in C_q^j */
7              return 𝕀'_i ← x'_i                           /* Return visit */
8          end
9      end
10 end
```

Algorithm 1: New visit algorithm

The algorithm to find free time is defined in Algorithm 2. L and U are the lower and upper bound of the time between scheduled times. The possible use cases are depicted in Figure 3. In each case depicted by Figure 3, the red line shows the arrival and departure time for an arbitrary bus visit, i. The blue lines indicate reigons in which charger q is active. $C \in \mathbb{C}_q \subset \mathbb{C}$ represents one of the regions between the blue lines, $[L, U]$ which stand for the lower and upper portions of the regions, respectively. The output of Algorithm 2 is a range for which the bus may be charged and the empty set if it cannot. As an example, consider a bus that is in the process of being scheduled and it encounters a situation similar to Figure 3c. That is, the only scheduling constraint is that the arrival time is before charger q is available to charge the bus. Therefore, the bus must wait intil L before changer q may charge it. Furthermore, the range that u_i must be selected from is $[L, e]$.

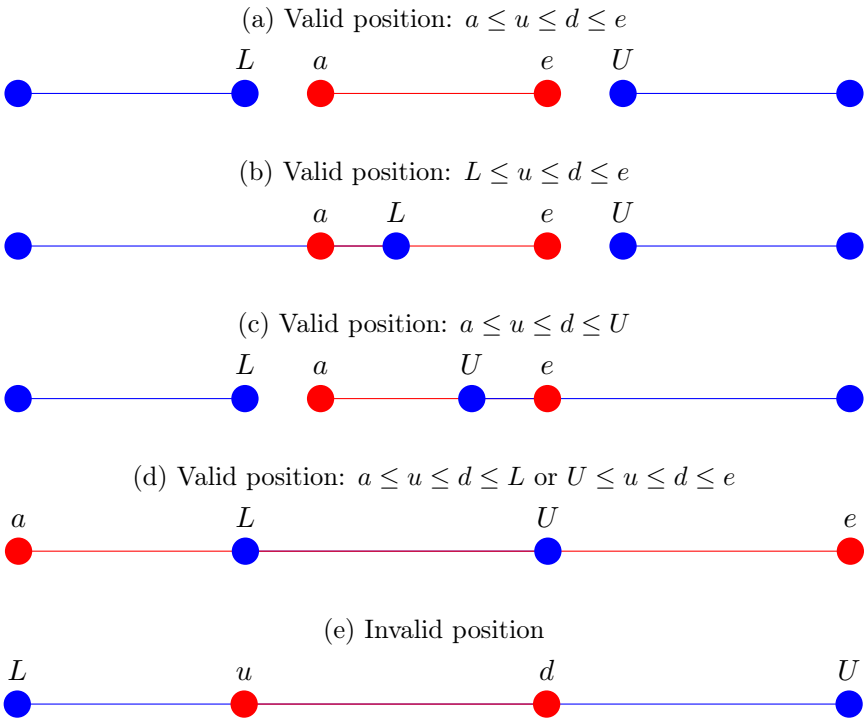

Figure 3: Outlines the different cases that requested time and charger allocated time can overlap

Algorithm: Find Free Time
Input: (L, U, a, e)
Output: (u, d)

```
1  begin
2      if L ≤ a and U ≥ e then                          /* If L < a < e < U] (Figure 3a) */
3          u ← U_[a,e];
4          d ← U_[u,e];
5      end
6      else if L > a and U ≥ e then                     /* Else if a < L < e < U (Figure 3b) */
7          u ← U_[L,e];
8          d ← U_[u,e];
9      end
10     else if L ≤ a and U < e then                     /* Else if L < a < U < e (Figure 3c) */
11         u ← U_[a,U];
12         d ← U_[u,U];
13     end
14     else if L > a and U < e then                     /* Else if a ≤ u ≤ d ≤ L or U ≤ a ≤ d ≤ e (Figure 3d) */
15         u ← U_[a,L],[U,e];
16         d ← U_[u,L],[u,e];
17     end
18     else                                             /* Otherwise the bus cannot be scheduled in this time frame (Figure 3e) */
19         u ← ∅;
20         d ← ∅;
21     end
22     return (u,d)
23 end
```

Algorithm 2: Find free time algorithm searches and returns the available time frames

Slide visit Slide visit is used for buses that have already been scheduled. Because of the constraint Equation 12k there may be some room to move u_i and d_i within the window $[a_i, e_i]$. Two new values, u_i and d_i are selected with a uniform distribution to satisfy the constraint $a_i \leq u_i \leq d_i \leq e_i$. Line 2 randomly loops through the other inactive time frames from charger v_i that is within the time that the bus is at the station, $[a_i, e_i]$. Line 3 verifies the time frame previously selected and returns a new charging time. Line 4 returns the new visit.

Algorithm: Slide Visit

Input: $\mathbb{S}$

Output: $\mathbb{I}'_i$

```
1 begin
    /* Randomly select an inactive time frame, j, from charger v_i that is within the time frame
       [a_i, e_i]                                                                              */
2   while C_q^j ∈ {[a_i, e_i]} ⊂ U_{C_{v_i}} do
3     if x'_i → findFreeTime(C_q^j, (a_i, e_i)) ∉ ∅ then      /* If there is time available in C_q^j */
4       return I_i ← x'_i                                                  /* Return visit */
5     end
6   end
7 end
```

Algorithm 3: Slide Visit Algorithm

New charger The new charger generator takes a visit $\mathbb{I}_i$ and changes the charger it is on while maintianing the same charge time, $[u_i, d_i]$. Similarly to 1, the new candidate, x'_i, must be checked before being added to the set X'_i. Line 2 randomly loops through any time frame from any charger, C_q^j, that is within the time frame that the bus is at the station, $[a_i, e_i]$. Line 3 verifies the selection and line 4 returns the new visit.

Algorithm: New Charger

Input: $\mathbb{S}$

Output: $\mathbb{I}'_i$

```
1 begin
    /* Randomly select an inactive time frame, j, from any charger q ∈ Q that is within the time
       frame [a_i, e_i]                                                                        */
2   while C_q^j ∈ {[a_i, e_i]} ⊂ U_C do
3     if L ≤ u  and U ≥ e then                  /* If the charge time is within the region [L,U] */
4       return I_i ← x'_i                                                  /* Return visit */
5     end
6   end
7 end
```

Algorithm 4: New Charger Algorithm

Remove The remove generator simply removes a bus from a charger queue and places it in its idle queue, $v_i \in \{Q, ..., Q + B\}$.

Algorithm: Remove

Input: $\mathbb{S}$

Output: $\mathbb{I}'_i$

```
1 begin
2   return (Q + b, a_i, e_i)
3 end
```

Algorithm 5: Remove algorithm

New Window New window is a combination of the remove and then new visit generators (Section 7 and Section 3). By this it is meant that current scheduled tuple (v_i, u_i, d_i) is removed and added back in as if it were a new visit.

Algorithm: New Window
Input: $\mathbb{S}$
Output: $\mathbb{I}$

```
1 begin
2 |  x'_i ← Remove(v, u, d) ;        /* Remove visit i from its charger */
3 |  x'_i ← NewVisit(x'_i) ;          /* Add visit i back in randomly */
4 |  return (v, u, d)
5 end
```

Algorithm 6: New window algorithm

4.3.3 Generator Wrappers

This section covers the algorithms utilized to select and execute different generation processes for the SA process. The generator wrappers are the method immediately called by SA. Each wrapper utilizes the generators previously described and returns either metadata about the bus routes or a new valid charger schedule.

Route Generation The objective of route generation is to create a set of metadata about bus routes given the information in Figure 4. Specifically, the objective is to generate the input variables in $\mathbb{I}$ for I visits with B buses. Each visit will have an initial charge (specified for first visit only), arrival time, departure and time. In other words, $y_i \equiv (a_i, e_i) \subset \mathbb{I} \forall i \in \mathcal{I}$ and each bus will be initialed with the SOC defined by Equation 1.

In essence the logic is as follows: Generate B random numbers that add up to I visits (with a minimum amount of visits set for each bus). For each bus and for each visit, set a departure time that is between the range $[\min_{\text{rest}}, \text{nmax}_{\text{rest}}]$ (Figure 4), set the next arrival time to be $j \cdot \frac{T}{\text{J}}$ where j is the j^{th} visit for bus b and J is the total number of visits for bus b. Finally, calculate the amount of discharge from the previous arrival to the next departure time as defined by Equation 1.

The metadata in Figure 4 will be denoted by $\mathbb{M}$ visually represents the YAML file used. This data contains all the parameters required to create a set of bus routes. Each of the cells will now be described from left to right, top to bottom. `time_horizon` represents the amount of time that the routes will be running for in hours. `schedule` contains the parameters that directly affect the generated set of routes. Some of the parameters in the YAML have already been defined in this paper, but go under a different name in the file. These parameters are: `num_bus` $\equiv B$, `num_visit` $\equiv I$, and `bat_capacity` $\equiv \kappa$. `max_charge` and `min_charge` represent the maximum and minimum percentages that the buses may be charged to. `max_rest` and `min_rest` represent the maximum and minimum times that buses may remain at the station, and `max_route` and `min_route` represent the maximum and minimum lengths that bus routes may be.

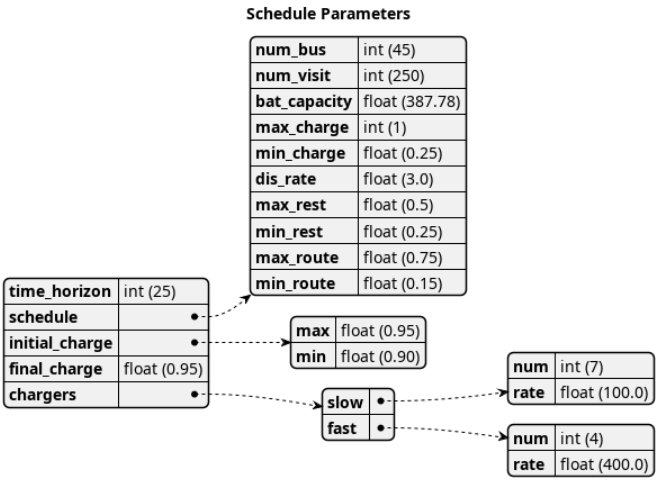

Figure 4: Route YAML file with example data

Algorithm: RouteGeneration
Input: $\mathbb{M}$
Output: $\mathbb{I}$

begin
  while ***not*** *schedule-created* **do**
    arrival-new $\leftarrow$ 0.0;
    arrival-old $\leftarrow$ 0.0;
    departure-time $\leftarrow$ 0.0;
    schedule-created $\leftarrow$ false;
    foreach $b \in B$ **do**
      foreach $n \in J_b$ **do**
        arrival-old $\leftarrow$ arrival-new;
        if $j = J_b$ **then**
          final-visit = true;
        end
        else
          final-visit = false;
        end
        departure-time $\leftarrow$ `DepartureTime(`*arrival-old, final-visit*`)`;
        arrival-new $\leftarrow$ current-visit*$\frac{T}{J_b}$;
        discharge $\leftarrow$ discharge-rate*(next-arrival, depart-time) ;
        `Union(`$\mathbb{I}$*, (arrival-old, departure-time, discharge)*`)`;
      end
    end
    schedule-created $\leftarrow$ `Feasible(`$\mathbb{I}$`)`;
    `SortByArrival(`$\mathbb{I}$`)`;
  end
  return $\mathbb{I}$
end

Algorithm 7: Route generation algorithm

The `Departure` algorithm is shown in Algorithm 8, and the `Feasible` method is used to determine if the generated schedule is valid (conditions covered in Subsection 3.3). This is done to generate a schedule that is in the solution space.

Algorithm: DepartureTime

Input: (arrival-old, final-visit)

Output: depart

```
1 begin
2 |  if final-visit then
3 |  |  depart ← T;
4 |  end
5 |  else
6 |  |  depart ← arrival-old + U_[min-rest,max-rest];
7 |  end
8 |  return depart
9 end
```

Algorithm 8: Departure time algorithm

Schedule Generation The objective of this generator is to generate a candidate solution to the given schedule. To generate a candidate solution, the generator is given $\mathbb{I}$, a bus is picked at random, $b \in \mathcal{B}$, then a random visit is picked. The new visit generator (1) is then utilized. This process is repeated for each visit. This algorithm is summarized in 9.

Algorithm: ScheduleGeneration

Input: $\mathbb{I}$, $\mathbb{C}$

Output: $\mathbb{I}'_i$

```
1 begin
2 |  I ← ∅;
3 |  for i in I do
4 |  |  b ← U_B;
5 |  |  i ← U_I;
6 |  |  I_i ← NewVisit((visit.a, visit.e));
7 |  |  Update C_q with information in I'_i;
8 |  end
9 |  return I
10 end
```

Algorithm 9: Schedule generation algorithm

Perturb Schedule As described in SA, local searches are also employed to try and exploit a given solution [8]. The method that will be employed to exploit the given solution is as follows: pick a bus, pick a visit, pick a generator. The algorithm is outlined in Algorithm 10.

Algorithm: PerturbSchedule

Input: $\mathbb{I}$, $\mathbb{C}$

Output: $\mathbb{I}'_i$

```
1 begin
2 |  Select the bus b ← U_B;
3 |  Select the visit j ← U_J_b;
4 |  generator ← U_[1,generator−count];
5 |  I_i ← GeneratorCallback [generator](j, I, C);
6 |  Update C_q with information in I'_i;
7 |  return I_i
8 end
```

Algorithm 10: Perturb schedule algorithm

5 Optimization Algorithm

This final section combines the generation algorithms and the optimization problem into a single algorithm. It begins with an introduction to a general SA algorithm which will be used to springboard into the construction of the SA PAP algorithm. For the case of the pseudo SA algorithm to be presented, the notation pesented will be self-contained and not related to any of the variables presented for SA PAP thus far. Consider Algorithm algorithm **??** [3]. ω and ω' are the current solution and the candidate solution, respectively. t_k is the temperature cooling schedule, T is the initial temperature that will iterate until $k = K$. M_k is the repetition counter, it defines the number of iterations to execute for each temperture t_k.

The algorithm behaves as follows: initialize the SA algorithm with an initial solution, temperature schedule, and repetition schedule. The first loops until $T = t_K$, the second loop finished whin $m = M_k$. For each loop, create a new solution, calculate the difference in the fitness of ω and ω'. Update ω with ω' if the candidate solution is better. Update ω with ω' with probability $e^{\frac{-\Delta_{\omega,\omega'}}{t_k}}$ if the candidate solution is worse than the current solution. This is repeated until the stopping criteria is met.

begin
- Select an initial solution $\omega \in W$;
- Select the temperature change counter $k = 0$;
- Select a temperature cooling schedule t_k;
- Select an initial temperature $T = t_0 \geq 0$;
- Set the initial inactive time for each charger to the time horizon $\forall q \in \mathcal{Q} : \mathbb{C}_q \in \{[0, T]\}$;
- Select a repetition schedule M_k, that defines the number of iterations executed at each temerature t_k;
- **while** *Stopping criterion not met* **do**
  - Set repetition counter $m \rightarrow 0$;
  - **while** $m = M_k$ **do**
    - Generate a new solution $\omega' \in N(\omega)$;
    - Calculate $\Delta_{\omega,\omega'} \rightarrow$ `f(`

The objective now is to outline SA-PAP in Algorithm 12. Lines 2-4 initialize the SA algorithm by defining the initial temperature, selecting the cooling schedule, and setting the repetition schedule. Line 5 loops through each of the step in the temperature schedule $T \in \{T_0, T_1, ..., T_m\}$. Lines 6 and 7 generate a new solution and calculates its fitness. ν in this context is defined as $\nu = (u, d, v,)$ Lines 8 through 13 updates the solution depending on if the new solution is better or worse than the previous solution. Line 14 iterates through the repetition schedule, $k \in \{1, 2, ..., K\}$. Lines 15-23 perturbs the previously generated solution, calculates its fitness, and updates the current solution with the candidate solution depending on the fitness.

Algorithm: SA PAP

Input: $\mathbb{I}$

Output: $\mathbb{I}'$

begin
  Initialize temperature T_0;
  Select cooling equation $T_M \leftarrow$ `CoolingEquation(`T_0`)`;
  Set a repetition schedule K;
  for $T_m \in \{T_0, T_1, ..., T_M\}$ **do**
    Generate a new solution $\upsilon' \in Y \leftarrow$ `ScheduleGeneration(`$\mathbb{I}$`)`;
    Calculate $\mathcal{V}_{\upsilon,\upsilon'} =$ `J(`υ'`) - J(`υ`)`;
    if $\mathcal{V}_{\upsilon,\upsilon'} \leq 0$ **then**
      $\upsilon \leftarrow \upsilon'$
    end
    if $\mathcal{V}_{\upsilon,\upsilon'} \leq 0$ **then**
      $\upsilon \leftarrow \upsilon'$ with probability $e^{\frac{\mathcal{V}_{\upsilon,\upsilon'}}{T_m}}$
    end
    for $k \in \{1, 2, ..., K\}$ **do**
      Perturb the solution and reassess $\upsilon' \in Y \leftarrow$ `PerturbSchedule(`$\mathbb{I}$`)`;
      Calculate $\mathcal{V}_{\upsilon,\upsilon'} =$ `J(`υ'`) - J(`υ`)`;
      if $\mathcal{V}_{\upsilon,\upsilon'} \leq 0$ **then**
        $\upsilon \leftarrow \upsilon'$
      end
      if $\mathcal{V}_{\upsilon,\upsilon'} \leq 0$ **then**
        $\upsilon \leftarrow \upsilon'$ with probability $e^{\frac{\mathcal{V}_{\upsilon,\upsilon'}}{T_m}}$
      end
    end
  end
end

Algorithm 11: Simulated annealing approach to the position allocation problem

References

[1] Michel Gendreau and Jean-Yves Potvin, editors. *Handbook of Metaheuristics*. Internationalseries in operation research & management science. Springer International Publishing, 3 edition, oct.

[2] Paul R. Halmos. Naive set theory. *Undergraduate Texts in Mathematics*, 1974.

[3] Darrall Henderson, Sheldon H. Jacobson, and Alan W. Johnson. The theory and practice of simulated annealing. In *International Series in Operations Research &amp: Management Science*, pages 287–319. Kluwer Academic Publishers.

[4] Andre A. Keller. *Multi-Objective Optimization In Theory and Practice II: Metaheuristic Algorithms*. BENTHAM SCIENCE PUBLISHERS, mar 2019.

[5] Jing-Quan Li. Battery-electric transit bus developments and operations: A review. *International Journal of Sustainable Transportation*, 10(3):157–169, 2016.

[6] David G. Luenberger and Yinyu Ye. Penalty and barrier methods. *Linear and Nonlinear Programming*, page 401–433, 2008.

[7] Ahad Javandoust Qarebagh, Farnaz Sabahi, and Dariush Nazarpour. Optimized scheduling for solving position allocation problem in electric vehicle charging stations. In *2019 27th Iranian Conference on Electrical Engineering (ICEE)*, pages 593–597, 2019.

[8] Jordan Radosavljevic. *Metaheuristic Optimization in Power Engineering*. Energy Engineering. Institution of Engineering and Technology, Stevenage, England, June 2018.

[9] Maria Analia Rodriguez and Aldo Vecchietti. A comparative assessment of linearization methods for bilinear models. *Computers and Chemical Engineering*, 48:218–233, 2013.

[10] R.A. Rutenbar. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26, jan 1989.