UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Control Systems

Distributed Dual Subgradient Scheme for Charging of Plug-in Electric Vehicles

Professor: **Giuseppe Notarstefano**

Students:
Sarim Mehdi
Semih Cakiroglu
Erjon Ballukja
Marta Borri

Academic year 2018/2019

Abstract

We use the distributed dual-subgradient method to try and come up with an optimum overnight charging schedule for plug-in electric vehicles (PEVs). The vehicles are connected to one another to form a doubly-stochastic adjacency matrix where the edges are time-varying. Each vehicle is allotted a certain number of time slots and all vehicle are restricted by the power capacity of the charging substation. Hence, the problem is another example of applying consensus protocols where a collection of agents must agree on a suitable resource vector. However, the resource vector is partitioned such that each agent is allowed to influence only a small portion of it but all agents must do so with an agreement based on minimizing an objective function within fixed constraints. Making the problem distributed allows us to exploit parallelism and data privacy. Thus, each agent in the network has knowledge of its neighbor's dual variable vector and nothing else. We compare the results of our distributed simulation with results from a centralized version of the same problem where the optimization is done using the simplex method. Furthermore, we show convergence of our results from the distributed simulation to those obtained from a centralized version of the same problem. And we show that the consensus error weighted by step-size is still summable, meaning it is limited from above by infinity.

Contents

Introduction **7**

1 Problem Set-up **10**
1.1 General template 10
1.1.1 Arrangement of matrices for Matlab simulation 11
1.2 Fitting the problem to the template 12
1.2.1 Local matrices for a generic vehicle 14
1.2.2 Adjacency Matrix 18
1.2.3 Setting up matrices for the centralized version of the problem 19

2 Simulation results **21**
2.1 Convergence 21

Conclusions **34**

Appendix **35**

Bibliography **36**

List of Figures

1.1 Example evolution of energy for generic vehicle i 14

2.1 evolution of cost measured using estimated value of state at each iteration 22
2.2 absolute difference between centralized cost and dual cost . . 23
2.3 absolute difference between centralized cost and estimated cost 23
2.4 absolute difference between centralized cost and better estimated cost (red marks indicate iterations where convergence error became less than a user-specified threshold for some agent/s) 24
2.5 Two-norm of difference between each node vector and unique solution from centralized version against 1000 iterations $\|x_i - x^*\|$ 25
2.6 Evolution of all dual vectors against 1000 iterations(top) and evolution of the 2-norm difference of each dual vector with the average at each iteration for a total of 1000 iterations (bottom), $\left\|\lambda_i - \overline{\lambda}\right\|$ 26
2.7 Evolution of all dual vectors against 1000 iterations with some agents assigned higher power rating 27
2.8 (from top to bottom) Graph of violation using better estimate of the state. Graph of violation using better estimate of the state with some agents assigned higher power. Graph of violation using simple estimate of the state. Graph of violation using simple estimate of the state with some agents assigned higher power 28
2.9 absolute difference between centralized cost and dual cost (some agents have higher power) 29
2.10 absolute difference between centralized cost and estimated cost (some agents have higher power) 30
2.11 absolute difference between centralized cost and better estimated cost (red marks indicate iterations where convergence error became less than a user-specified threshold for some agent/s. Some agents have higher power) 30

2.12 absolute difference between centralized cost and better estimated cost (red marks indicate iterations where convergence error became less than a user-specified threshold for some agent/s. Some agents have higher power) 31
2.13 violation calculated using better estimate of the state vector (Some agents have higher power) 31
2.14 Summability of consensus error 32
2.15 duality gap between dual cost, estimated cost and better estimated cost 33
2.16 duality gap between dual cost, estimated cost and better estimated cost (some agents have higher power) 33

Introduction

Optimization problems are becoming more and more complicated as we try to solve problems from diverse domains. From financial issues like portfolio management to engineering problems related to energy generation and consumption, there are a range of problems whose answer is not so obvious and getting to the best solution requires an iterative algorithm. As such, optimization techniques have evolved to tackle complex problems. The usefulness of distributed algorithms was understood a long time ago when it became obvious that breaking down a really large problem into smaller subsets not only allows us to solve the problem with the help of a computer (memory limit stonewalls any effort you can make towards trying to solve the problem in a centralized fashion) but also allows us to see certain trends in the solution (i.e. convergence) that otherwise would not be possible had we tried to solve the problem in a centralized fashion.

Motivations

It is not a surprise that in an effort to reduce the human carbon footprint, we are moving more and more towards technology that is more green and eco-friendly. Plug-in Electric Vehicles serve this purpose by running entirely on electricity. However, with the advent of PEVs, a new set of problems has emerged. One such problem is related to efficiently charging PEVs without overloading the substation to which they are connected. Many efforts have been made to try to find an optimal solution to the problem [7]. In this report, we focus on the solution given by [1] and apply it to the original problem in [6].

It is clear that as more and more vehicles are connected to a charging substation, then the problem is more easily and sensibly tackled from a distributed viewpoint. Using a central solution may be faster (simplex method has a complexity of 2^n in worst-case and n^k in average-case) but it is infeasible as the number of cars become huge. Our main motivation is to study and understand the convergence property of the solution and also see how the evolution of the dual variables vector relates to coupling constraints being active or not.

Contributions

Falsone solved the problem of only-charging (and not with V2G, meaning vehicle to grid aka discharging) with a slightly modified version of the problem from Vujanic's paper. He was interested in showing convergence properties of the solution and consensus among agents regarding the dual variable vector. We wish to do the same thing but on Vujanic's version of the problem (with charging and V2G).Whereas Falsone was only interesting in minimizing the cost of charging a set of vehicles, our problem is to maximize profit or, to reformulate the problem as an objective function, minimize the loss from charging and V2G.

Our simulation results show that (for the data generated according to the parameters given by Vujanic) at most time slots the total power spent charging and/ or discharging stays well below the upper bound. This means, that the coupling constraints remain inactive and it is the same as ignoring the GLOBAL constraint and minimizing the objective function based on the local constraints. We were also able to show different behavior by increasing the power of charging/ discharging for some vehicles. This causes the agents to arrive at a much different consensus regarding the dual variable vector and the consensus is no longer close to 0 (as is the case when we generate data using the original parameters given in [6]). We say close to 0 because the dual vector at the next time slot is a linear combination, weighted by corresponding columns of the given row of adjacency matrix, of all the dual vectors present in the graph at that moment. So, even if the coupling constraint associated to some agent becomes 0 or less, the effect from summing all the dual vectors from the entire graph might still cause the value of the dual vector to be greater than 0 for the next iteration, albeit not by much).

Hence, we do two simulations: In one, we generate values according to the parameters given by Vujanic and then we generate a new set of values using the same parameters but with the exception that we now assign higher power to some agents so that the coupling constraint for them would become active. And this immediately becomes apparent when the evolution of the dual vectors is plotted against the total number of iterations.

Another thing worth mentioning is the necessity of having a unique solution to the problem. In order to avoid symmetry in the c matrix of the objective function, we perturb the cost at each time slot by a constant randomly generated factor (as formulated by Vujanic for the same reason). This ensures the presence of a unique solution.

Organization

The structure of the report is as follows. In the next section, we show how we set up the problem for Matlab (the construction of the matrices) followed

by a brief explanation of how the matrices were set up for solving the same problem in a centralized fashion. In the next chapter, we show results of our simulation and discuss convergence properties. Specifically, we show how the solution for our problem approaches the true solution we obtained from linear programming (i.e. treating the problem in a centralized fashion) in a logarithmic fashion. And we also show what kind of consensus is reached among agents regarding the dual variable vector. We also show how the consensus error of each agent approaches 0.

Chapter 1

Problem Set-up

Before setting up the problem statement of PEVs, it is important to describe the algorithm as a general template. Then, in the later section, the problem itself is fitted on this template.

1.1 General template

Consider an optimization problem defined as:

$$\min_{x \in X} \sum_{i=1}^{m} c_i x_i \tag{1.1}$$

$$X \triangleq \{x \in \mathbb{R}^n | Hx = h, Dx \leq d\}$$

Where m defines the total number of agents in our distributed network. Now, let the above minimization problem be subject to the following global constraint:

$$\sum_{i=1}^{m} A_i x_i \leq b \tag{1.2}$$

Each A_i is a matrix defining the GLOBAL constraint and it has a size such that the number of rows are equal to the number of global constraints applied on that specific x_i agent and the number of columns are the same as the number of subsystems for the specific x_i i.e. the number of rows of x_i

According to [1], the distributed algorithm, when applied, to the above problem statement, becomes:

1. $l_i = \sum_{j=1}^{m} \mathrm{a}_j^i \lambda_j$
2. $x_i[k+1] \in \arg\min_x (c_i x_i + \mathrm{l}_i^T A_i x_i - \mathrm{l}_i^T b)$

3. $\lambda_i[k+1] = max[0, l_i + c_k A_i x_i[k+1] - c_k \frac{b}{m}]$

4. $\hat{x_i}[k+1] = \frac{[\sum_{r=0}^{k-1} c_r x_{ir}] + c_k x_i[k+1]}{[\sum_{r=0}^{k-1} c_r] + c_k}$

Where, l_i is a weighted sum of all the dual variable vectors such that, for given agent i, each dual variable vector between the current agent and it's out-neighbors (or just neighbors in an undirected graph) is multiplied by the weight of the connection between the agent and the corresponding neighbor. Thus, l_i has the same dimensions as the dual variable column vector. a_j^i is the value in the adjacency matrix at row i (corresponding to the ith agent for which we run this algorithm locally) and column j (the other agent in the graph to which my agent i is connected. Self-edges are allowed, so we also include the weight of an agent connected to itself).

$\hat{x_i}[k+1]$ is a filtered version of the vector of subsystems for a given agent (like a running average). It is a linear combination of the previous and current value of the agent's state up till the iteration k+1 (where k is our current iteration). Notice how separating the sum of linear combination of previous states and weighting factor c_r allows us to use them as memory element. This means that, at every iteration, instead of repeating the entire summation from iteration 0 we can just use the sum we stored from the previous iteration and add to it the new value of the agent's state weighted by the factor c_k (where $c_k = \frac{\beta}{k+1}$ and β is a very small number between 0 and 1). This technique was heavily inspired from filtering methods used in computer graphics and manipulation [5]. c_k is our diminishing step-size and this is needed to ensure we converge to the exact solution and not some neighborhood around it as would be the case with a constant step-size [3]. x_{ir} is the value of the state vector of agent i at any generic iteration r before the current iteration.

1.1.1 Arrangement of matrices for Matlab simulation

To make the execution of the program more simple and concise, we store all the single dimension vectors, (row and column vectors such as c_i and λ_i) into one single matrix. So, for example, we store all the dual variable vectors in one single matrix and then we index into that matrix column-wise to extract the dual variable vector for a given agent.

On the other hand, 2D matrices, such as the global inequality matrix, are stored in a separate array of matrices (aka cell) and then indexed from there. To conclude this discussion, it is worth mentioning how Matlab's linprog is used to execute step 2 of the previously mentioned algorithm:

$$x_i[k+1] = linprog(c_i + \mathrm{l}_i^T A_i, A_{in}, b_{in}, A_e, b_e, min, max) \qquad (1.3)$$

Where, A_{in} and A_e are normal 2D matrices that represent the local inequality and equality matrices respectively. The remaining matrices, however, are fed as row vectors. b_{in} is the upper bound on the local inequality such that $A_{in}x_i \leq b_{in}$ and b_e is associated to the local equality such that $A_e x_i = b_e$. It is worth mentioning that GLPK (linear optimization library used in C++), however, requires the vectors to be fed as column vectors. The exception to this is the objective function which is still fed as a row vector as is done here when it is fed as the first argument of linprog.

1.2 Fitting the problem to the template

The problem of charging PEVs requires that we generate maximum profit while not overloading the central substation to which all the PEVs are connected. As such, in order to encourage generation of profit, the price vector related to the discharge time slots must be greater than that for the charging time slots.

According to [6], we have to minimize the following objective function:

$$\min_{u,v,e} \sum_{i \in I} P_i(\mathrm{u}_i^T C^u - \mathrm{v}_i^T C^v) \tag{1.4}$$

Since, u_i, v_i, C^u and C^v are vectors whose length is equal to the total number of time slots, we can re-write the above equation as:

$$\min_{u,v,e} \sum_{i \in I} \sum_{k=0}^{N-1} P_i(C^u[k]u_i(k) - C^v[k]v_i[k]) \tag{1.5}$$

Notice that even though e_i is included in the set of solutions, it is not present in the objective function (or the GLOBAL constraint). However, it plays a crucial role in the minimization problem as will be seen later. P_i is the power for a specific vehicle and it is constant for each time slot. Therefore, each car's energy level increases or decreases at a constant rate for all available time slots.

Since we want to maximize the profit, that means minimizing loss and so the above equation represent the money we lose when we are trying to charge our PEVs. And so it makes sense to minimize it (ideally, it must stay below 0 and that is indeed the case as will be shown in our simulation results). Finally, the GLOBAL constraints are defined as:

$$P^{min} \leq \sum_{i \in I} P_i(u_i - v_i) \leq P^{max} \tag{1.6}$$

Which can be re-written as:

$$P^{min}[k] \leq \sum_{i \in I} P_i(u_i[k] - v_i[k]) \leq P^{max}[k] \tag{1.7}$$

So, in my effort to maximize my profit, I also want to make sure that at any time slot k, I don't overload the substation to which all my PEVs are connected. This is the same as saying that the total power due to charging and/ or discharging all connected PEVs must stay within the bound. The dynamics for each vehicle introduce local constraints and, therefore, for each vehicle we have the following local constraints:

1. $e_i[0] = \mathrm{E}_i^{init}$
2. $e_i[k+1] = e_i[k] + P_i \Delta T(\zeta_i^u \mathrm{u}_i[k] - \zeta_i^v \mathrm{v}_i[k]), k \in (0, ..., N-1)$
3. $e_i[N] \geq \mathrm{E}_i^{ref}$
4. $\mathrm{E}_i^{min} \leq e_i[k] \leq \mathrm{E}_i^{max}$
5. $u_i[k] + v_i[k] \leq 1$

Suffice to say, a brief explanation of the idea behind local constraints is needed to have a general idea of how the local search space for each vehicle is defined where the linprog (using the simplex algorithm) looks for an optimal solution for a given agent. Inequality (5) states that a vehicle can only charge or discharge at any given time slot. Furthermore, when a vehicle decides to charge at arbitrary time slot k, the value of $u_i[k]$ becomes 1, otherwise it is 0. Same idea applies to $v_i[k]$. Therefore, $u_i[k], v_i[k] \in [0, 1]$

The goal of charging is to make sure that by the time we are at the last time slot (time slot N), the vehicle needs to have accumulated a certain amount of charge (indicated here as E_i^{ref}). However, at any time slot from 1 till N, the charging level needs to be bounded from above and below by E_i^{max} and E_i^{min} respectively. One way of seeing this is through the following graph:

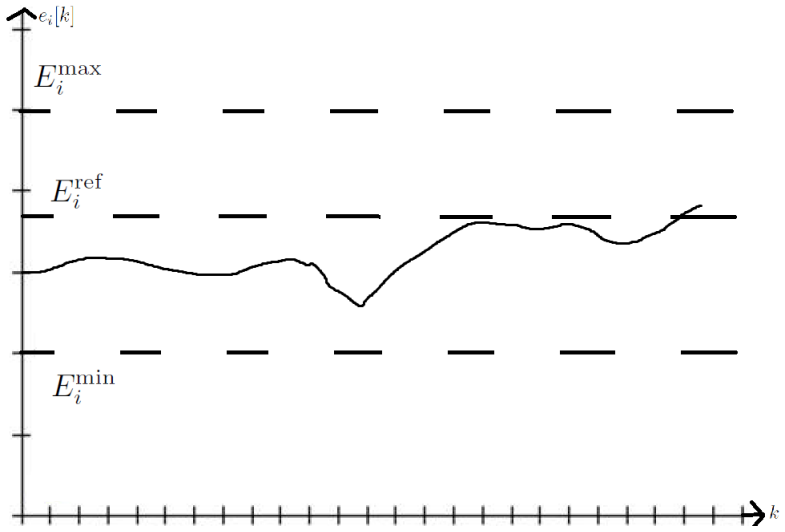

Figure 1.1: Example evolution of energy for generic vehicle i

So, E_i^{ref} can be though of as an inequality for only the last energy level of vehicle. Keeping this in mind, we can now start writing the matrices which define the local constraints for any given vehicle.

1.2.1 Local matrices for a generic vehicle

To make a comprehensive case, we have decided to present a generic case for 4 subsystems to show a pattern in the construction of the matrices. This way, it would not be hard to convince the reader of the overall structure for a matrix with some unknown N number of subsystems.

For 4 timeslots, the generic x_i vector will have 4 charging times (u[0] to u[3]), 4 discharging times (v[0] to v[3]) and 4 energy levels (e[1] to e[4]). In the interest of keeping the structure of x_i simply and easily manipulable, we have decided to not include e[0] in the state vector. This makes sense, in general, because e[0] is constant anyway for a generic vehicle i.

$$
\begin{aligned}
u[0] + v[0] &\leq 1 \\
u[1] + v[1] &\leq 1 \\
u[2] + v[2] &\leq 1 \\
u[3] + v[3] &\leq 1 \\
-e[4] &\leq -\mathrm{E}^{ref}
\end{aligned}
$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \vdots & 1 & 0 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \vdots & 0 & 1 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \vdots & 0 & 0 & 1 & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \vdots & 0 & 0 & 0 & 1 & \vdots & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 & \vdots & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \\ \cdots \\ v[0] \\ v[1] \\ v[2] \\ v[3] \\ \cdots \\ e[1] \\ e[2] \\ e[3] \\ e[4] \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \cdots \\ -\mathrm{E}^{ref} \end{bmatrix}$$

So, for N timeslots:

$$\begin{bmatrix} 1 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & -1 \end{bmatrix} \begin{bmatrix} u[0] \\ \vdots \\ u[N-1] \\ v[0] \\ \vdots \\ v[N-1] \\ e[1] \\ \vdots \\ e[N] \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ -\mathrm{E}^{ref} \end{bmatrix}$$

For local equalities, once again consider 4 timeslots and let $A = P\Delta T\zeta^u$ and let $B = P\Delta T\zeta^v$:

$$e[1] + Bv[0] - Au[0] = \mathrm{E}^{init}$$
$$e[2] + Bv[0] + Bv[1] - Au[0] - Au[1] = \mathrm{E}^{init}$$
$$e[3] + Bv[0] + Bv[1] + Bv[2] - Au[0] - Au[1] - Au[2] = \mathrm{E}^{init}$$
$$e[4] + Bv[0] + Bv[1] + Bv[2] + Bv[3] - Au[0] - Au[1] - Au[2] - Au[3] = \mathrm{E}^{init}$$

Therefore, each energy level is simply a linear combination of current and

previous energy levels.

$$\begin{bmatrix} -A & 0 & 0 & 0 & \vdots & B & 0 & 0 & 0 & \vdots & 1 & 0 & 0 & 0 \\ -A & -A & 0 & 0 & \vdots & B & B & 0 & 0 & \vdots & 0 & 1 & 0 & 0 \\ -A & -A & -A & 0 & \vdots & B & B & B & 0 & \vdots & 0 & 0 & 1 & 0 \\ -A & -A & -A & -A & \vdots & B & B & B & B & \vdots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \\ \cdots \\ v[0] \\ v[1] \\ v[2] \\ v[3] \\ \cdots \\ e[1] \\ e[2] \\ e[3] \\ e[4] \end{bmatrix} = \begin{bmatrix} \mathrm{E}^{init} \\ \mathrm{E}^{init} \\ \mathrm{E}^{init} \\ \mathrm{E}^{init} \end{bmatrix}$$

So, for N timeslots:

$$\begin{bmatrix} -A & 0 & \cdots & 0 & \vdots & B & 0 & \cdots & 0 & \vdots & 1 & 0 & \cdots & 0 \\ -A & -A & \cdots & 0 & \vdots & B & B & \cdots & 0 & \vdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -A & -A & \cdots & -A & \vdots & B & B & \cdots & B & \vdots & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} u[0] \\ \vdots \\ u[N-1] \\ v[0] \\ \vdots \\ v[N-1] \\ e[1] \\ \vdots \\ e[N] \end{bmatrix} = \begin{bmatrix} \mathrm{E}^{init} \\ \vdots \\ \mathrm{E}^{init} \end{bmatrix}$$

Hence, the local equality matrix is nothing more than a concatenation of two lower triangular matrices associated with the charging and discharging efficiencies plus an identity matrix that represents the contribution of the current charging level. After this, we now write the GLOBAL inequality matrices and cost function as a linear combination of 4 timeslots (followed by a generic representation of N timeslots):

$$P(C^u[0]+\delta^u)u[0]+P(C^u[1]+\delta^u)u[1]+P(C^u[2]+\delta^u)u[2]+P(C^u[3]+\delta^u)u[3]$$

$$-P(C^v[0]+\delta^v)v[0]-P(C^v[1]+\delta^v)v[1]-P(C^v[2]+\delta^v)v[2]-P(C^v[3]+\delta^v)v[3]$$

$$\begin{bmatrix} P(C^u[0]+\delta^u) \\ P(C^u[1]+\delta^u) \\ P(C^u[2]+\delta^u) \\ P(C^u[3]+\delta^u) \\ \cdots \\ -P(C^v[0]+\delta^v) \\ -P(C^v[1]+\delta^v) \\ -P(C^v[2]+\delta^v) \\ -P(C^v[3]+\delta^v) \\ \cdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \\ \cdots \\ v[0] \\ v[1] \\ v[2] \\ v[3] \\ \cdots \\ e[1] \\ e[2] \\ e[3] \\ e[4] \end{bmatrix}$$

So, for N timeslots:

$$\begin{bmatrix} P(C^u[0]+\delta^u) \\ \vdots \\ P(C^u[N-1]+\delta^u) \\ -P(C^v[0]+\delta^v) \\ \vdots \\ -P(C^v[N-1]+\delta^v) \\ 0 \\ \vdots \\ 0 \end{bmatrix}^T \begin{bmatrix} u[0] \\ \vdots \\ u[N-1] \\ v[0] \\ \vdots \\ v[N-1] \\ e[1] \\ \vdots \\ e[N] \end{bmatrix}$$

Where the left matrix is the c row vector related to the ith agent. In a similar fashion, we can write the GLOBAL inequality matrix:

$$\begin{aligned} Pu[0] - Pv[0] &\leq P^{max}[0] \\ Pu[1] - Pv[1] &\leq P^{max}[1] \\ Pu[2] - Pv[2] &\leq P^{max}[2] \\ Pu[3] - Pv[3] &\leq P^{max}[3] \\ Pv[0] - Pu[0] &\leq -P^{min}[0] \\ Pv[1] - Pu[1] &\leq -P^{min}[1] \\ Pv[2] - Pu[2] &\leq -P^{min}[2] \\ Pv[3] - Pu[3] &\leq -P^{min}[3] \end{aligned}$$

$$\begin{bmatrix} P & 0 & 0 & 0 & \vdots & -P & 0 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & P & 0 & 0 & \vdots & 0 & -P & 0 & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & 0 & P & 0 & \vdots & 0 & 0 & -P & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & P & \vdots & 0 & 0 & 0 & -P & \vdots & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ -P & 0 & 0 & 0 & \vdots & P & 0 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & -P & 0 & 0 & \vdots & 0 & P & 0 & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & 0 & -P & 0 & \vdots & 0 & 0 & P & 0 & \vdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -P & \vdots & 0 & 0 & 0 & P & \vdots & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \\ \cdots \\ v[0] \\ v[1] \\ v[2] \\ v[3] \\ \cdots \\ e[1] \\ e[2] \\ e[3] \\ e[4] \end{bmatrix} \leq \begin{bmatrix} P^{max}[0] \\ P^{max}[1] \\ P^{max}[2] \\ P^{max}[3] \\ \cdots \\ -P^{min}[0] \\ -P^{min}[1] \\ -P^{min}[2] \\ -P^{min}[3] \end{bmatrix}$$

So, for N timeslots:

$$\begin{bmatrix} P & \cdots & 0 & -P & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & P & 0 & \cdots & -P & 0 & \cdots & 0 \\ -P & \cdots & 0 & P & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -P & 0 & \cdots & P & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} u[0] \\ \vdots \\ u[N-1] \\ v[0] \\ \vdots \\ v[N-1] \\ e[1] \\ \vdots \\ e[N] \end{bmatrix} \leq \begin{bmatrix} P^{max}[0] \\ \vdots \\ P^{max}[N-1] \\ -P^{min}[0] \\ \vdots \\ -P^{min}[N-1] \end{bmatrix}$$

Where The 2D matrix is the GLOBAL A inequality matrix. Finally, it is important to mention the upper and lower bound on every x_i:

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ E^{min} \\ \vdots \\ E^{min} \end{bmatrix} \leq \begin{bmatrix} u[0] \\ \vdots \\ u[N-1] \\ v[0] \\ \vdots \\ v[N-1] \\ e[1] \\ \vdots \\ e[N] \end{bmatrix} \leq \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 1 \\ \vdots \\ 1 \\ E^{max} \\ \vdots \\ E^{max} \end{bmatrix}$$

1.2.2 Adjacency Matrix

Due to the time-varying nature of the problem, we formulate a randomized adjacency matrix in the beginning and then we generate a new one after

every T iterations (this period can be adjusted by the user in code). It is worth mentioning, briefly, the method used to randomly generate the adjacency matrix. We first generate a random I by I matrix (I is the number of agents in the network) by picking the number 1 from a binomial distribution with a success rate of 0.8. So, we get a matrix of 1s and 0s. Then we convert it to a lower triangular matrix and add the transpose of this new matrix to its lower triangular version to get a symmetric matrix. Next, we count the number of 1s in each column and store that value in an array of I elements (we call this array *degree*).

Then, we scan each row of our matrix. We ignore places in the row where we put 0 and only focus on where we have 1 in that row. So, the column at which we find a 1, we index into *degree* and record the value at that index. Then, we index into *degree* once again but this time according to the index of the row we were scanning in our original I by I matrix. Now we compare this value with the value we recorded previously. We take the larger of these two values and simply put the inverse of it in the original matrix at the same place where we found the 1 that prompted this whole comparison in the first place. Repeating this for each row would eventually give us a doubly stochastic matrix where the sum of all elements in each row and column is 1. This is a necessary condition for strong connectivity of our network's graph [4]

1.2.3 Setting up matrices for the centralized version of the problem

We can write the overall objective function for I agents:

$$\begin{bmatrix} c_1 \cdots c_I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix}$$

We can combine the GLOBAL inequalities of all I agents into one:

$$\begin{bmatrix} A_1 & \cdots & A_I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix} \leq \begin{bmatrix} b_1 \\ \vdots \\ b_I \end{bmatrix}$$

We can combine the local inequalities of all I agents into one:

$$\begin{bmatrix} Aineq_1 & \cdots & 0_{N+1\times 3N} \\ \vdots & \ddots & \vdots \\ 0_{N+1\times 3N} & \cdots & Aineq_I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix} \leq \begin{bmatrix} bineq_1 \\ \vdots \\ bineq_I \end{bmatrix}$$

The final inequality matrix is obtained by combining the GLOBAL and local inequalities into one:

$$\begin{bmatrix} A_1 & \cdots & A_I \\ Aineq_1 & \cdots & 0_{N+1\times 3N} \\ \vdots & \ddots & \vdots \\ 0_{N+1\times 3N} & \cdots & Aineq_I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix} \leq \begin{bmatrix} b_1 \\ \vdots \\ b_I \\ bineq_1 \\ \vdots \\ bineq_I \end{bmatrix}$$

Finally, we can combine the local equalities of all I agents into one:

$$\begin{bmatrix} Aeq_1 & \cdots & 0_{N\times 3N} \\ \vdots & \ddots & \vdots \\ 0_{N\times 3N} & \cdots & Aeq_I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix} = \begin{bmatrix} beq_1 \\ \vdots \\ beq_I \end{bmatrix}$$

We combine the upper and lower bound:

$$\begin{bmatrix} X_{low1} \\ \vdots \\ X_{lowI} \end{bmatrix} \leq \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix} \leq \begin{bmatrix} X_{high1} \\ \vdots \\ X_{highI} \end{bmatrix}$$

Chapter 2

Simulation results

We ran simulations starting with a fleet of 100 vehicles. We incremented the sample size by 100 and stopped at 400 because trying to run the centralized version of the problem for a fleet of vehicles greater than 400 gave us memory issues. The hardware we used for simulation had an Intel Core i5-8250U CPU with 8 cores (each of 1.60 GHz) and 12 GB RAM and we ran each simulation on a 64-bit Ubuntu 18.04.1 LTS. For each simulation, we also added the option to increase the power of charging/ discharging for some vehicles. Furthermore, we would periodically alter the adjacency matrix (period was 2 in both simulations). For every fleet of vehicles, we ran two simulations (each simulation was run for 1000 iterations). In the first simulation, the data was generated exactly according to the parameters given by [6] and in the second one we randomly increased the power for some vehicles. Below we discuss in detail the simulation results for a fleet of 100 vehicles.

2.1 Convergence

Due to bad initial estimate of x_i and λ_i we decided to use a threshold on the convergence error $\|l_i[k+1] - \lambda_i[k+1]\|$ as done by Falsone. However, for our charging and V2G scenario, we used a different threshold of 0.001. As expected, we had a better convergence estimate because we decided to neglect the values of the state before the iteration at which this threshold became true. This allows us to negate the influence of the bad initial estimate when calculating the estimate of each node's state by recomputing it from a new starting point. And this threshold gives us a better guarantee that at this new starting point we will have a much better estimate of the node's state as compared to the very beginning. It also comes as no surprise to see the cost function being minimized during a span of 1000 iterations (as we wish to minimize our loss during charging and V2G).

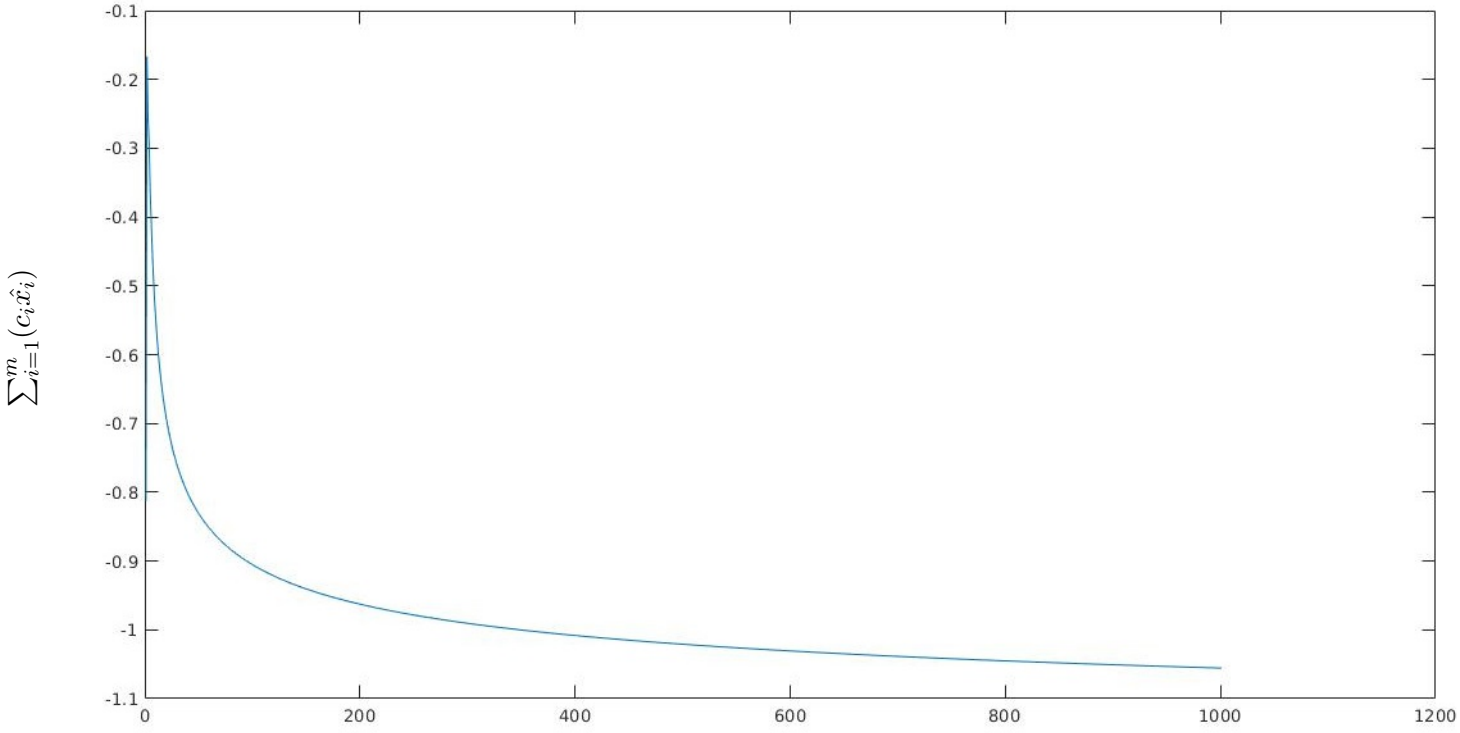

Figure 2.1: evolution of cost measured using estimated value of state at each iteration

We decided to plot the dual cost against the cost computed using the unique solution obtained by solving the problem in a centralized fashion. This allowed us to observe some convergence behavior and we saw that the dual cost and the cost calculated using the estimated and better estimated state approaches the true cost (from the centralized version) in a logarithmic fashion.

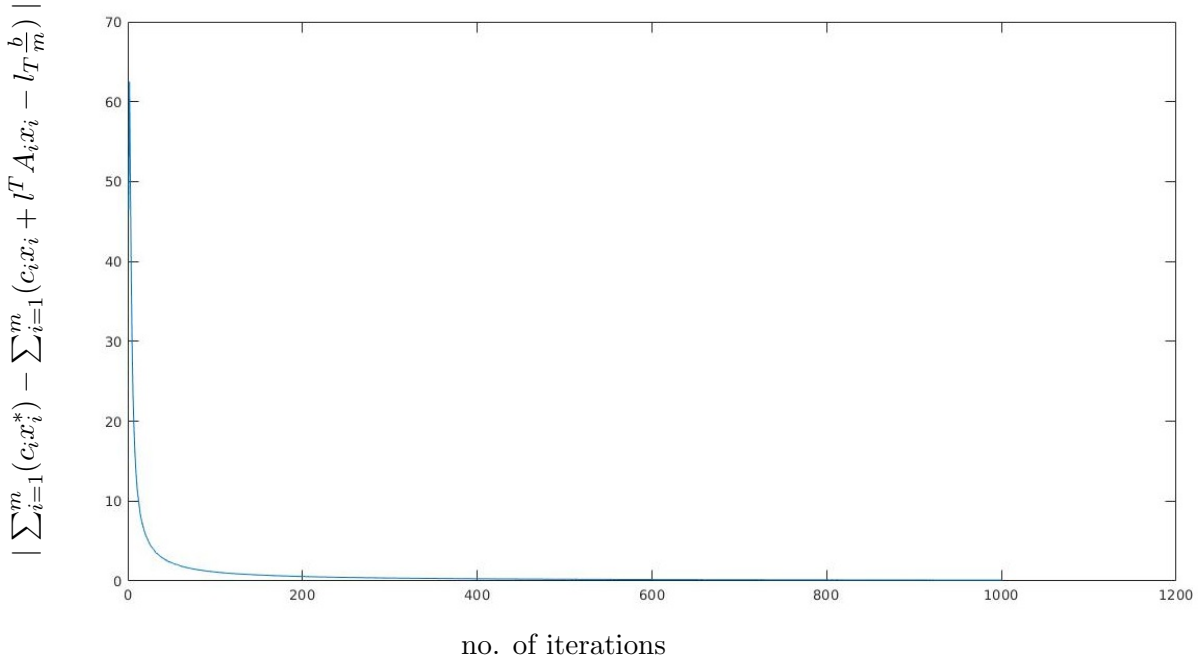

Figure 2.2: absolute difference between centralized cost and dual cost

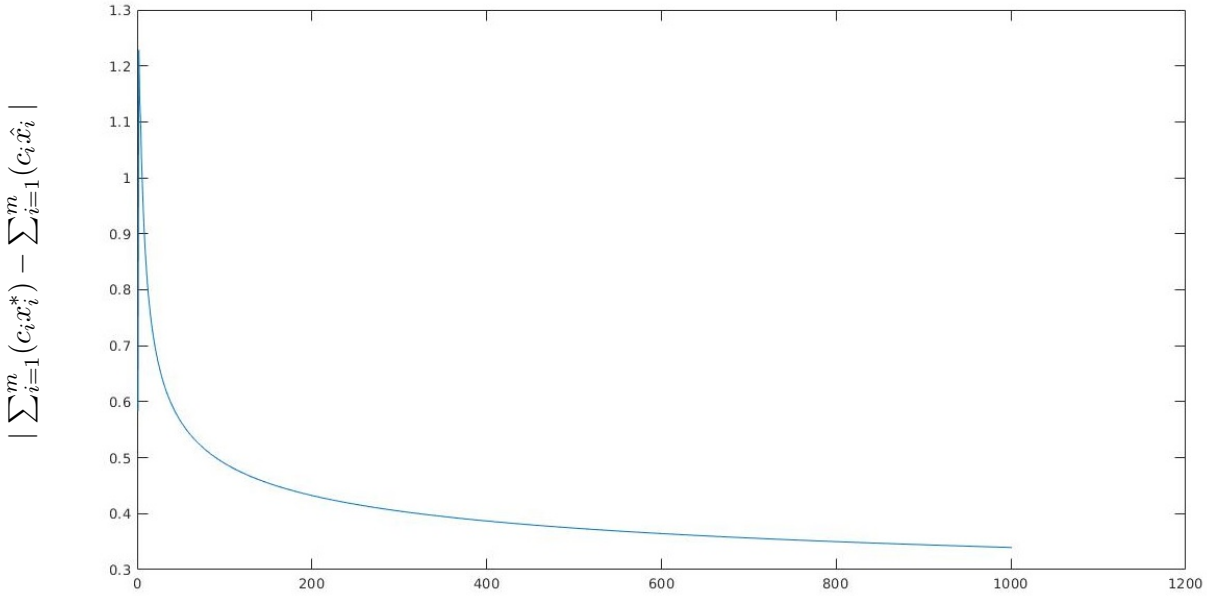

Figure 2.3: absolute difference between centralized cost and estimated cost

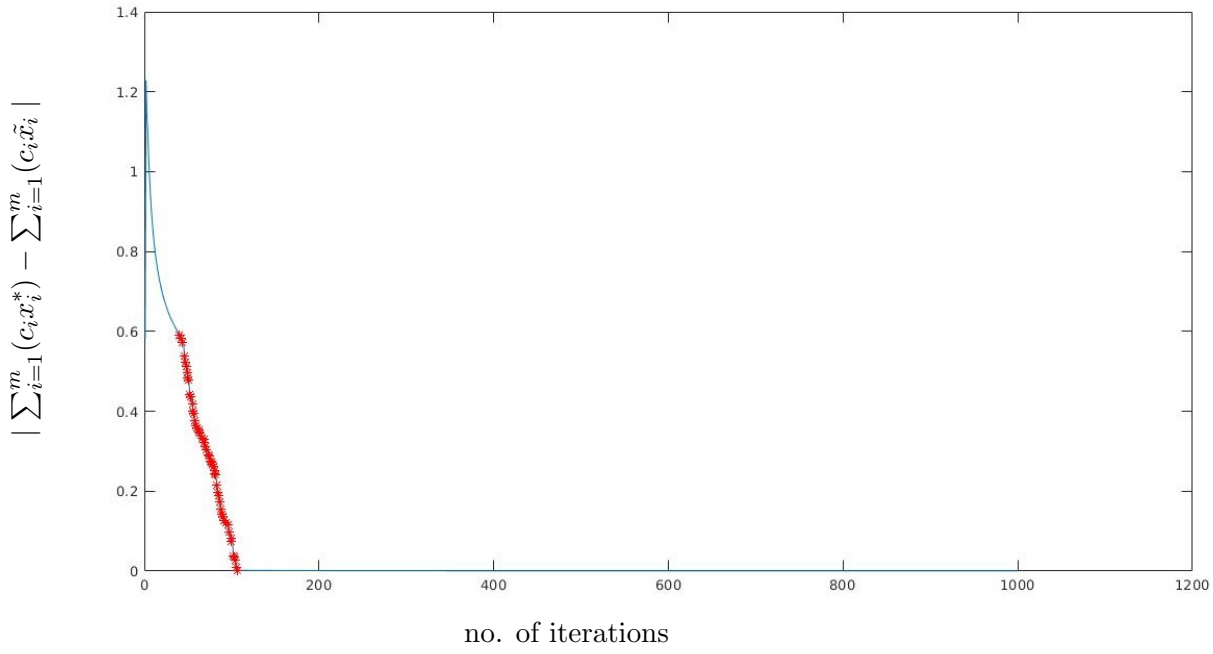

Figure 2.4: absolute difference between centralized cost and better estimated cost (red marks indicate iterations where convergence error became less than a user-specified threshold for some agent/s)

As expected, at the end of 1000 iterations, each x_i approached the unique solution from the centralized version of the program. Furthermore, the agents also arrive at a consensus regarding the dual vectors which can be seen when we plot the 2-norm between each dual vector and the average of all dual vectors at every iteration. At the end of 1000 iterations, this 2-norm difference approaches zero for each agent. This means that we can safely ignore the GLOBAL inequality and minimize the cost function purely based on the local constraints of each agent.

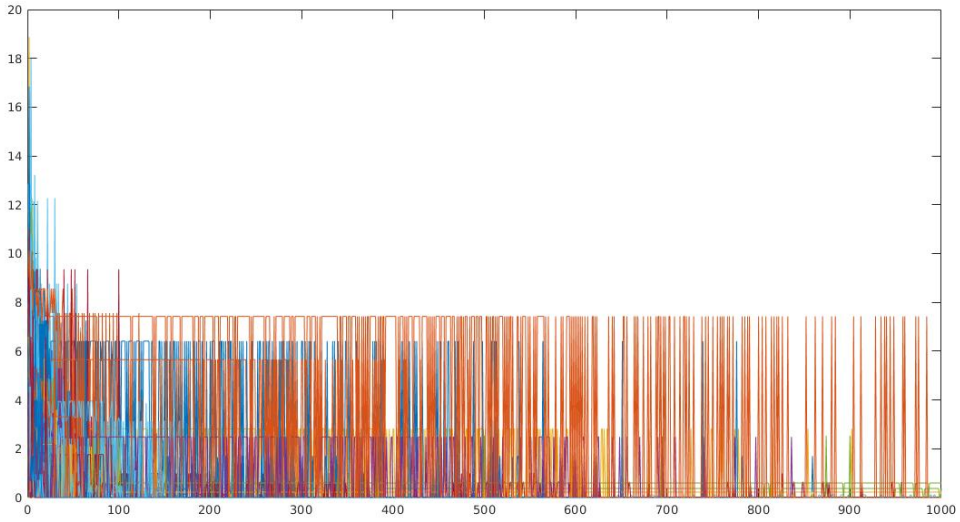

Figure 2.5: Two-norm of difference between each node vector and unique solution from centralized version against 1000 iterations $\|x_i - x^*\|$

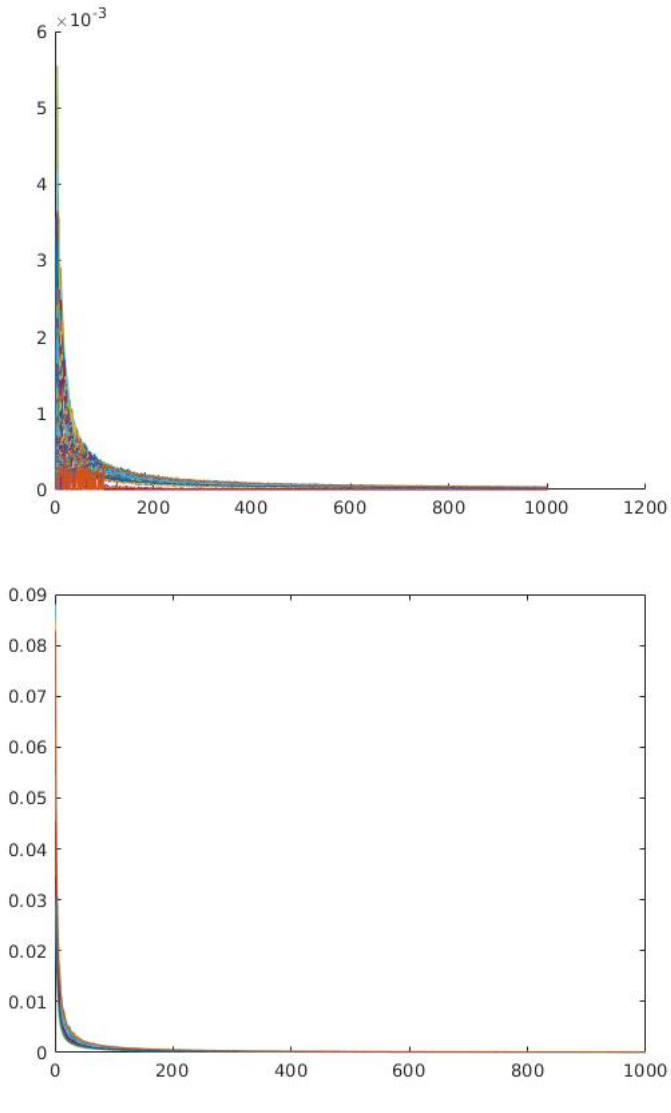

Figure 2.6: Evolution of all dual vectors against 1000 iterations(top) and evolution of the 2-norm difference of each dual vector with the average at each iteration for a total of 1000 iterations (bottom), $\left\|\lambda_i - \overline{\lambda}\right\|$

In the case of assigning significantly higher powers to some agents in the network, however, the consensus on the dual variable vector was quite different. This made sense because, in this case, $A_i x_i - \frac{b}{I}$ had a much larger value than before at each iteration for those agents that now had a power rating significantly higher as compared to those who got assigned a power rating based on the parameters from [6].

In order to accommodate for increased power, we also had to increase

E^{max} suitably. The new range necessary for randomizing the upper threshold at each energy level was carefully chosen such that it wasn't too high but not too low either. If the new threshold was too high, linprog would be unable to find a local solution since higher threshold meant higher reference energy level for the last time slot too which might not be satisfied under the current power rating for the vehicle. A significantly lower threshold (or even keeping the same threshold as before) meant that the threshold would be violated immediately within the first two or three timeslots and thus most of the u and v vector would consist of 0s and we would once again be left with inactive coupling constraints when evaluating the dual vector. The results of this alternative simulation can be seen below.

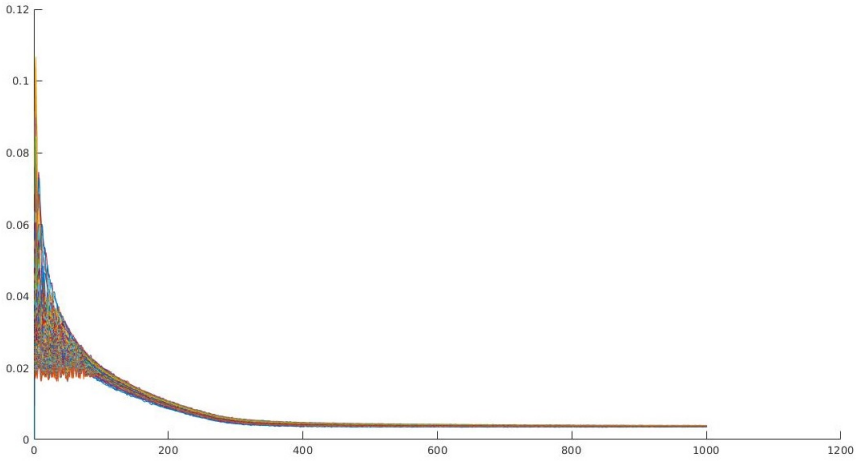

Figure 2.7: Evolution of all dual vectors against 1000 iterations with some agents assigned higher power rating

It comes as no surprise because, from the algorithm, $\lambda_i[k+1]) = max[0, l_i + c_k A_i x_i[k+1] - c_k \frac{b}{m}]$. And due to the higher power rating (this information is encoded in the A_i matrix which consists of identity matrices multiplied by the power and vertically concatenated), as explained above, $c_k A_i x_i[k+1] - c_k \frac{b}{m}$ ends up having a much larger value as compared to before for each iteration. As a result of this, all agents end up having consensus on a value of the dual that is much larger than before. We see the result of this alternate simulation when we plot the evolution of the violation using both the estimated and better estimated state vector. Violation is given by $\sum_{i=1}^{m} A_i x_i - b$

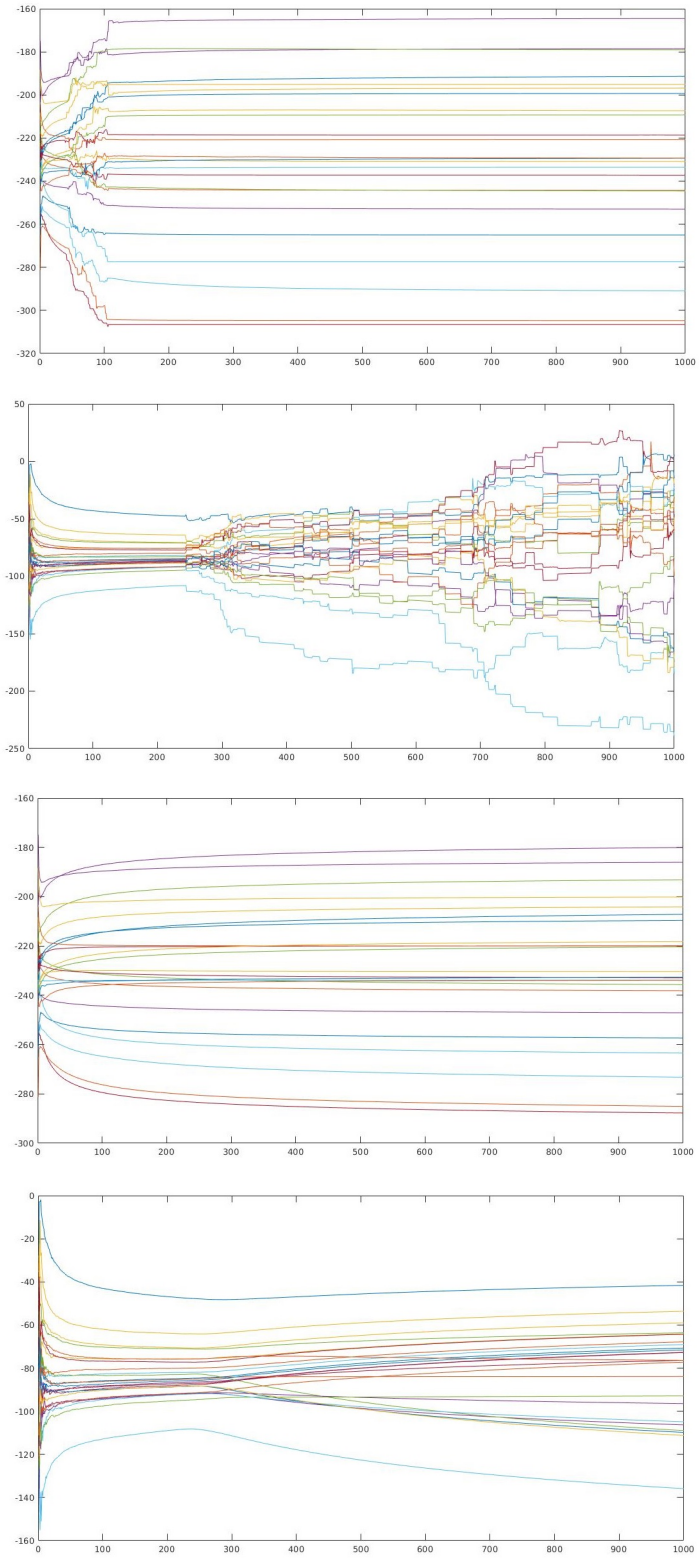

Figure 2.8: (from top to bottom) Graph of violation using better estimate of the state. Graph of violation using better estimate of the state with some agents assigned higher power. Graph of violation using simple estimate of the state. Graph of violation using simple estimate of the state with some agents assigned higher power

In the second graph from the bottom (where we use the better estimate of the state. It is worth remembering that the better estimate of the state is the same as the running average with the only difference that the running average is computed onwards from the iteration at which the convergence error dropped below a user specified threshold. The reader is directed to read [1] for more details), we notice that at two time slots the GLOBAL inequality becomes active. And indeed this is not surprising since the dual vectors have a significantly higher value as before.

In the alternative simulation, we also see a significantly different behavior for the evolution of the cost as compared to before where all the data was generated using the usual parameters. Specifically, the dual cost takes a bit longer to converge to the true cost (calculated from the centralized version of the problem) whereas the estimated and better estimated cost show poorer convergence properties as compared to before:

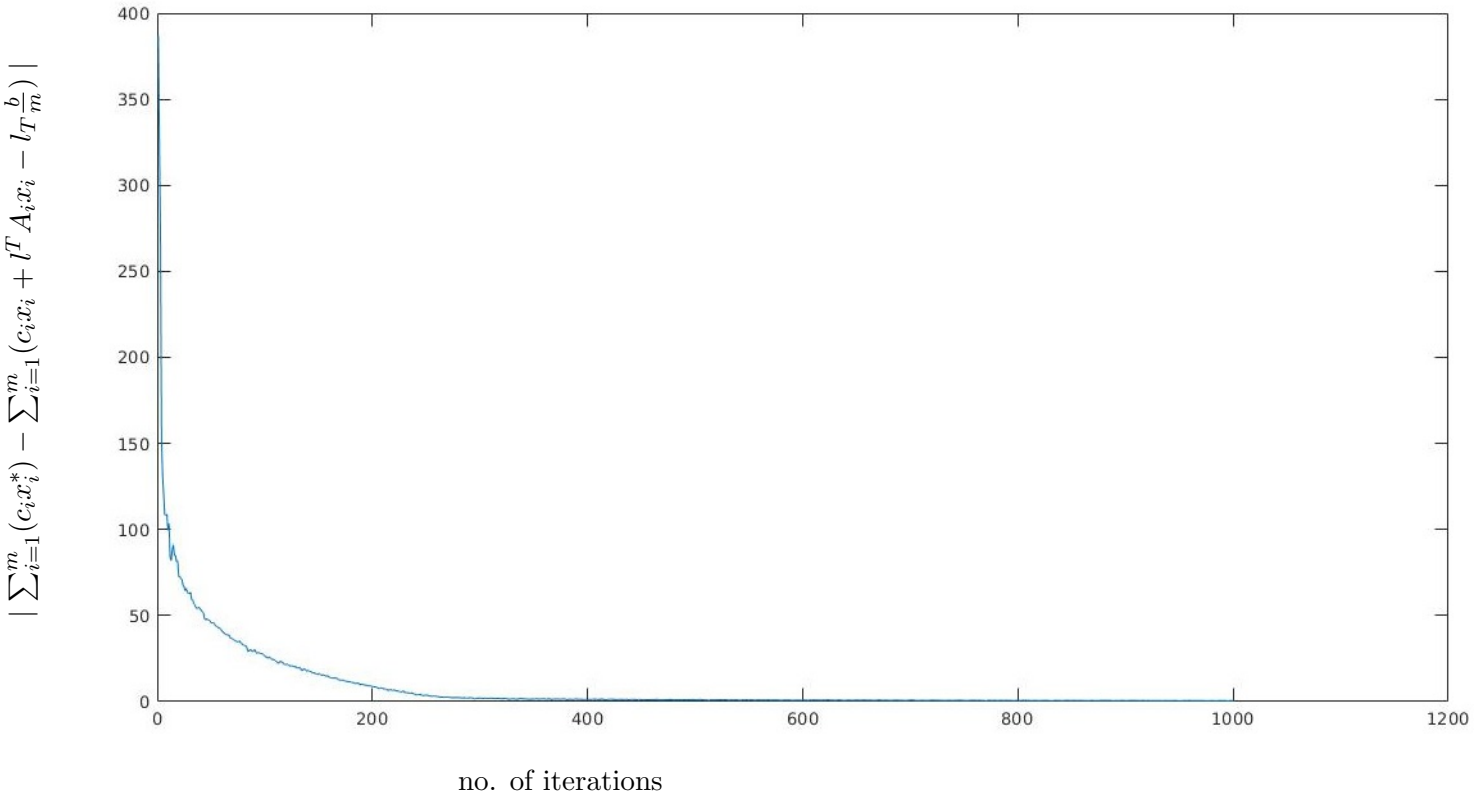

Figure 2.9: absolute difference between centralized cost and dual cost (some agents have higher power)

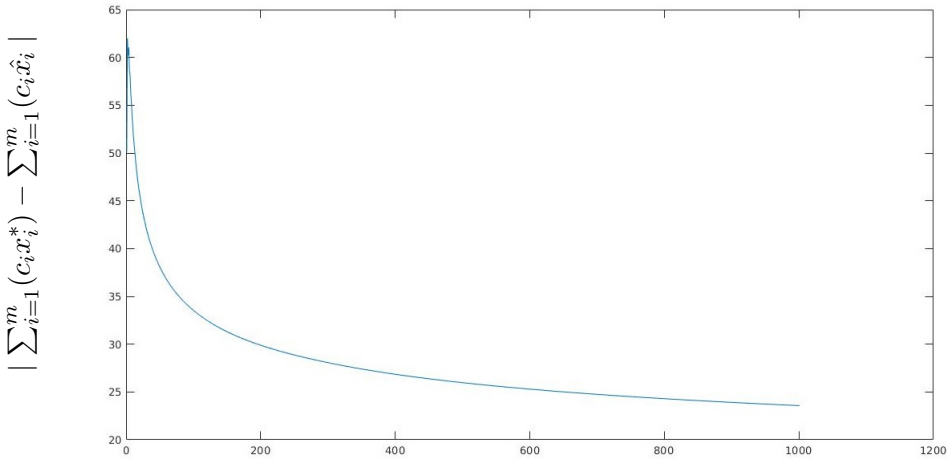

Figure 2.10: absolute difference between centralized cost and estimated cost (some agents have higher power)

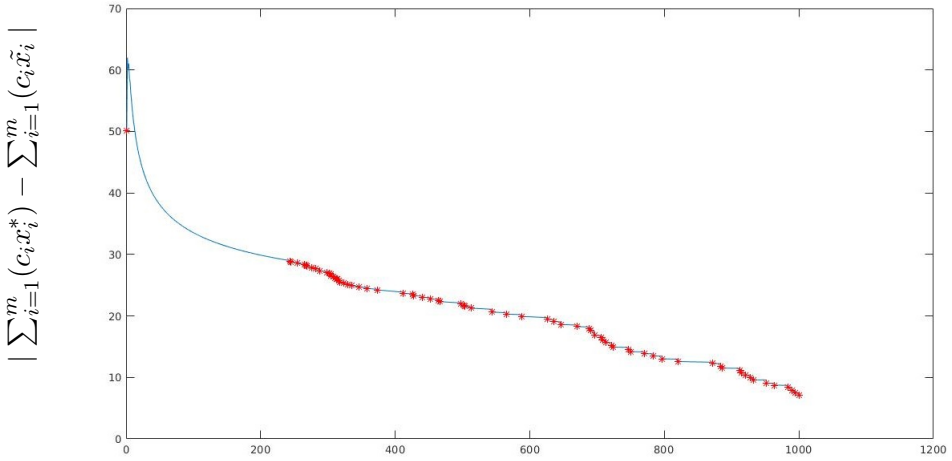

Figure 2.11: absolute difference between centralized cost and better estimated cost (red marks indicate iterations where convergence error became less than a user-specified threshold for some agent/s. Some agents have higher power)

To confirm convergence, we decided to repeat this alternative simulation for 5000 iterations. As expected, convergence happens a few iterations after 1000.

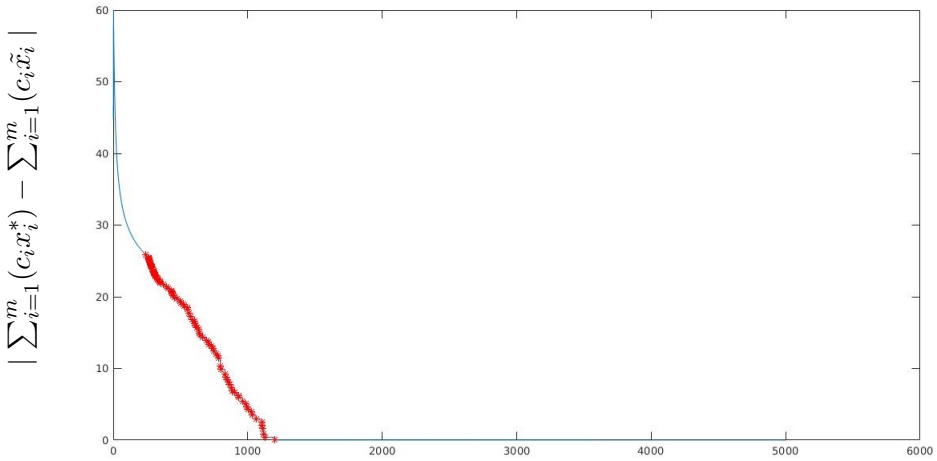

Figure 2.12: absolute difference between centralized cost and better estimated cost (red marks indicate iterations where convergence error became less than a user-specified threshold for some agent/s. Some agents have higher power)

And at some time slots we see the coupling constraint become active.

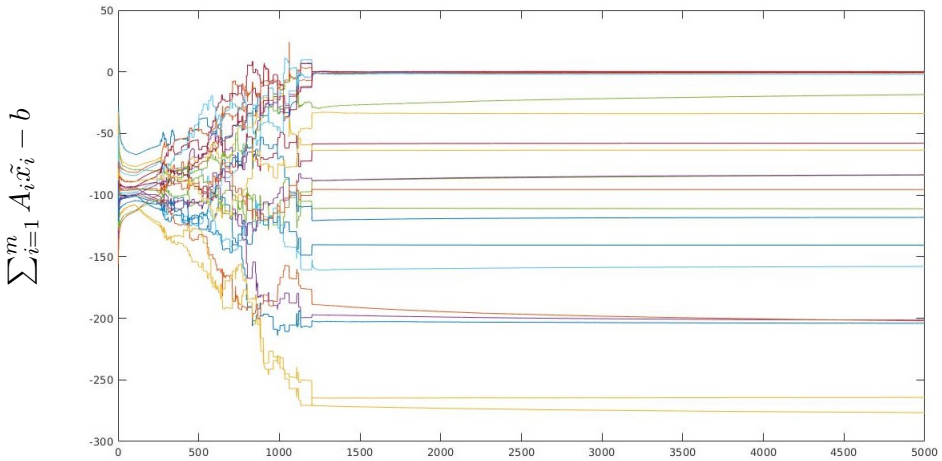

Figure 2.13: violation calculated using better estimate of the state vector (Some agents have higher power)

As a final confirmation, we also decided to check (see Appendix for proof):

$$\lim_{T\to\infty} \sum_{t=0}^{T} \gamma^t \left\| x_i^t - \overline{x^t} \right\| \leq \infty$$

As expected, this holds true and we see the sum of all 2-norms, weighted by the diminishing step-size, approach some value less than ∞.

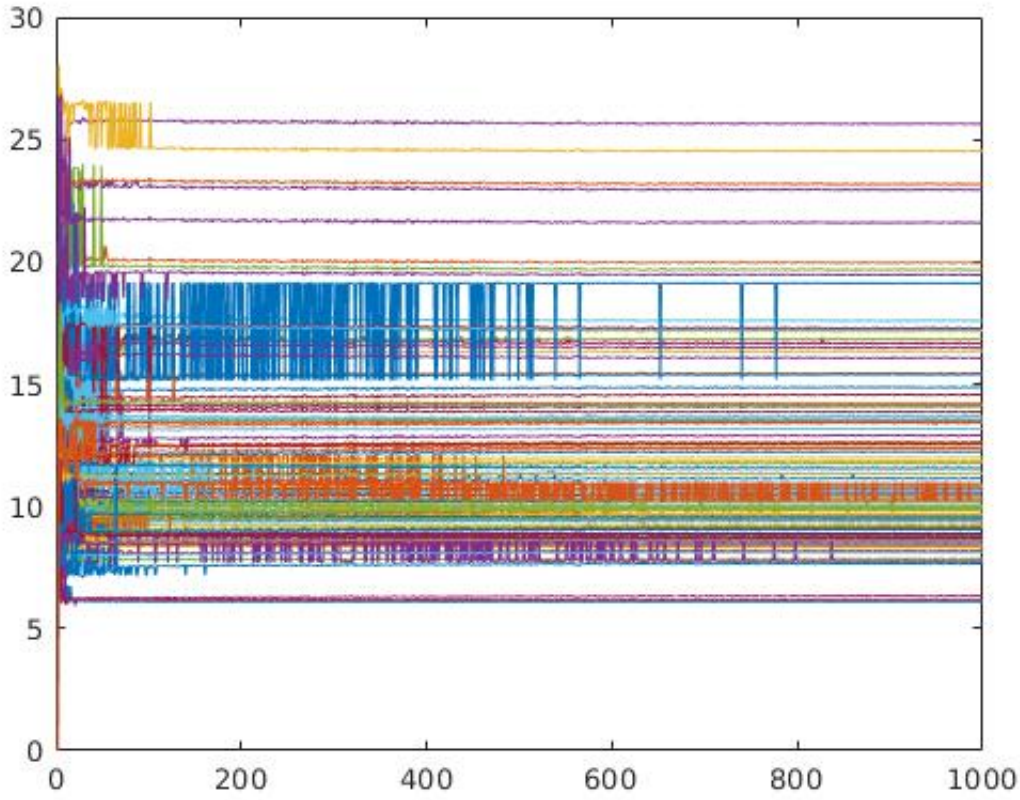

Figure 2.14: Summability of consensus error

We also decided to investigate the duality gap for both versions of the simulation. As shown below, the dual cost approaches the cost calculated using the estimated and better estimated state vector within the first 100 iterations. However, that is not the case for the alternate simulation (with some agents assigned a higher power rating) and the dual cost only approaches the cost calculated using the better estimated state vector (and that too after 1000 iterations), whereas there is a strict (and constant) duality gap with the cost calculated using the normal estimated state vector. Hence, this shows how we can use the better estimate of the state vector to recover the original solution of the problem.

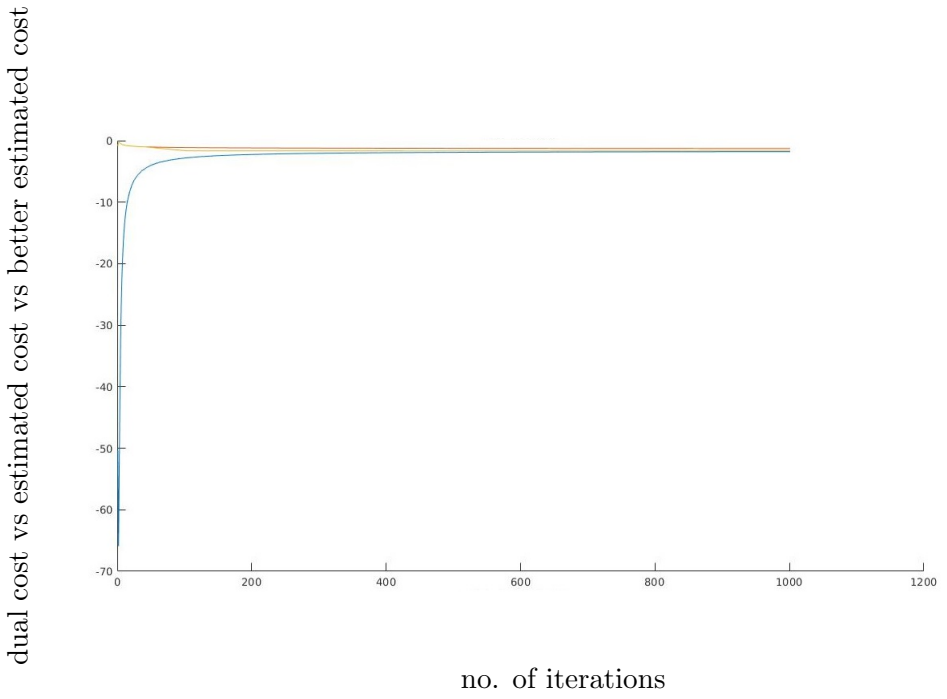

Figure 2.15: duality gap between dual cost, estimated cost and better estimated cost

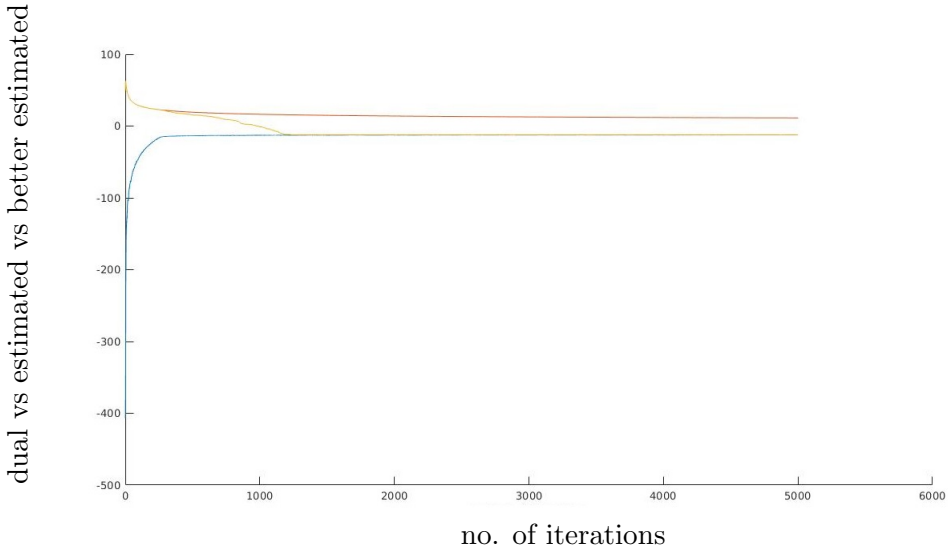

Figure 2.16: duality gap between dual cost, estimated cost and better estimated cost (some agents have higher power)

Conclusions

We applied the distributed dual subgradient on two versions of the same problem set and were able to draw interesting conclusions after looking at the consensus results. The algorithm exploits parallelism to a great degree and one cannot appreciate more the fact that being distributed allowed us to also solve the problem for a really huge set (more than 500 vehicles) of resources. This could not have been possible with the centralized version of the problem. We are also sure of the correctness of the algorithm as the solution from the algorithm approached the actual result obtained from linear programming (centralized version of the problem). Also, the result from using higher power values for some agents allowed us to see how vastly different the consensus can be. Future work will discuss more in detail the application of this algorithm to other problems that require creation of complicated schedules. We could effectively apply this method to logistics and draw up a decent time table to improve the efficiency in a warehouse. Or, something that hits close to home (no pun intended), we can use it to evaluate better travel routes for buses since on-time arrival of buses is of most importance and that doesn't seem to be the case in many cities (including Bologna).

Appendix

$$\lim_{T\to\infty}\sum_{t=0}^{T}\gamma^t\left\|x^t-\overline{x^t}\mathbf{1}\right\|\leq\lim_{T\to\infty}\sum_{t=0}^{T}\gamma^t\sigma_A^t\left\|x^0-\overline{x^0}\mathbf{1}\right\|+\lim_{T\to\infty}\sum_{t=0}^{T}\gamma^t\sum_{\tau=0}^{t-1}\sigma_A^{t-1-\tau}\left\|\epsilon^\tau\right\|$$

$$\lim_{T\to\infty}\sum_{t=0}^{T}\gamma^t\left\|x^t-\overline{x^t}\mathbf{1}\right\|\leq\lim_{T\to\infty}\sum_{t=0}^{T}\gamma^t\sigma_A^t\left\|x^0-\overline{x^0}\mathbf{1}\right\|+\lim_{T\to\infty}\sum_{t=0}^{T}\gamma^t\sum_{\tau=0}^{t-1}\sigma_A^{t-1-\tau}\gamma^\tau\left\|\tilde{\nabla}f_i\right\|$$

The first term is bounded due to geometric series property and the second term is bounded due to subgradients being bounded and because the we have a diminishing step-size [2].

Bibliography

[1] S. Garatti A. Falsone, K. Margellos and M. Prandini. Dual decomposition for multi-agent distributed optimization with coupling constraints. *Automatica*, 2017.

[2] A. Ozdaglar A. NediÂ´c. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. on Autom. Control*, 2009.

[3] Hadi Jamali-Rad Andrea Simonetto. Primal recovery from consensus-based dual decomposition for distributed convex optimization. *Journal of Optimization Theory and Applications*, 2016.

[4] J. W. Spellman D. J. Hartfiel. A role for doubly stochastic matrices in graph theory. *Proceedings of the American Mathematical Society*, 1972.

[5] M.J. McDonnell. Box-filtering techniques. *Computer Graphics and Image Processing*, 1980.

[6] P.J. Goulart S. MariÃ¨thoz R. Vujanic, P. M. Esfahani and M. Morari. A decomposition method for large scale milps with performance guarantees and a power system application. *Automatica*, 2016.

[7] Yong Hyun Song Seung Wan Kim, Young Gyu Jin and Yong Tae Yoon. A priority index method for efficient charging of pevs in a charging station with constrained power consumption. *JEET*, 2016.