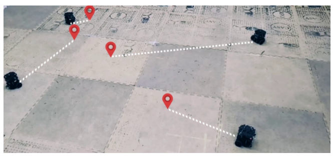
UNIVERSITÀ DI BOLOGNA



School of Engineering

Master Degree in Automation Engineering

Distributed Control Systems

Distributed Task Assignment for Robotic Networks

Student:
Stefano Mulargia

Academic year 2020/2021

Abstract

In this report we will deal with modeling and control of a network of robots that need to self-assign a set of target locations. The objective is to choose a complete assignment that optimizes the total path length. First, we will show how it is possible to solve a constraint-coupled linear program in a distributed way, then we will consider a team of robots that want to self-assign a set of tasks scattered in the environment. We developed a MATLAB implementation of the dual subgradient both with a centralized version and a distributed setup, showing the comparison between the two formulations.

Contents

List of Figures

List of Tables

Introduction

Assignment problems mainly consist in optimizing the total cost of matching tasks and agents represented by a weighted bipartite graph.

In our scenario, the task assignment considers an environment populated with tasks and a group of robots, also called agents, that are not necessarily equally capable of performing each of the tasks. We will consider the tasks such as locations to reach, but in general they can be considered also as objects or other agents to interact with and an agent may only be assigned to one task at any given time.

The goal of this project, just like in a usual distributed multi-agent task assignment, is to optimizing a cost associated with the total distance traveled by the agents.

In the next section we will recall some theoretical concepts, in order to deal with the problems easily. In the first chapter we address a general situation with a constraint-coupled linear program in which we will perform an optimization, both in a centralized way and in a distributed way. In the second chapter we will deal with the task assignment of a swarm of robots. The solutions we will present go from the most particular one, in which we have all drones that can perform every task, to a more general one in which we separated agents that can do only tasks on the ground, called mobile robots, and agents that can do both tasks on the ground and tasks in air, called drones.

Motivations

Distributed task assignment with multiple agents can be very useful in swarm robotics. In particular, distributed control of multi-agent systems compared to centralized classical control provide an increase of efficiency, performance, scalability and robustness. On the other hand, distributed algorithms are way slower than the centralized ones and they require a lot of additional computations.

General constraints-coupling

Suppose to have a system composed of a set of agents in which the optimization problem can be solved with a distributed computation minimizing the sum of local cost functions. Each agent depends on a local variable, it is subjected to a local constraint for each variable and to a coupling constraint involving all the decision variables. Each agent will solve its local optimal solution and then they will be stacked to obtain a global optimal solution. [3].

The following is the general optimization problem formulation (see Primal problem in Appendix C):

$$\begin{aligned} \min_{x_1,\dots,x_N} \quad & \sum_{i=1}^{N} f_i(x_i) \\ \text{subj. to} \quad & x_i \in X_i \qquad i \in \{1,\dots,N\} \\ & \sum_{i=1}^{N} g_i(x_i) \leq 0 \end{aligned} \tag{1}$$

where $f_i : \mathbb{R}^{n_i} \to \mathbb{R}, \quad g_i : \mathbb{R}^{n_i} \to \mathbb{R}^S, \quad X_i \subseteq \mathbb{R}^{n_i} \quad$ with $n_i, S \in \mathbb{N}$

Assumptions valid for each agent:

- f_i is a convex function and it is the *cost function* that we want to minimize;
- g_i is component-wise convex function. Namely each local constraint function is convex.
- X_i is a non-empty compact convex set.

A particular case is the so called *resource allocation*, where the coupling constraint is linear, e.g., $\sum_{i=1}^{N} x_i = b$ and there are no local constraints.

Chapter 1

Constraint-Coupled Linear Program Optimization

In this chapter, we will introduce the generic optimization problem and we will show how to implement it into a constraint-coupled linear problem. First we will write about the problem formulation and then we will illustrate both the centralized and the distributed solutions.

1.1 Task 1 formulation

Suppose we have N agents that want to cooperatively solve the constraint-coupled linear program:

$$\min_{z_1,\dots,z_N} \quad \sum_{i=1}^{N} c_i^T z_i \tag{1.1}$$

$$\text{subj.to} \quad \sum_{i=1}^{N} H_i z_i = b \tag{1.2}$$

$$z_i \in P_i, \quad \forall i \in \{1,\dots,N\}$$

where $c_i \in \mathbb{R}^{n_i}$, $H_i \in \mathbb{R}^{S\times n_i}$, $b \in \mathbb{R}^S$ and P_i is a compact polyhedron described by linear equality and inequality constraints, i.e.,

$$P_i \triangleq \{z_i \in \mathbb{R}^{n_i} \quad | \quad D_i z_i \leq d_i\}, \quad \forall i \in \{1,\dots,N\} \tag{1.3}$$

where $D_i \in \mathbb{R}^{n_i\times n_i}$ and $d_i \in \mathbb{R}^{n_i}$.

We can think of c_i as the cost that an agent i must pay when assigned to j.

The goal of this task is to minimize the cost function (1.1), taking into account the equality constraint (1.2). The polyhedron definition adds an inequality constraint. The presence of coupling constraints does not allow to solve N optimization problems separately.
The duality theory (Appendix C) helps us in cast the problem in an convex problem in the form of cost-coupled. Since the function can be not differentiable, we can apply the subgradient method [4] to obtain the optimal solution.

The generation of the data is done with the code given by the tutor according to [5].

1.2 Centralized Dual Subgradient

We can write the projected subgradient on the dual problem (see Appendix C.8).

$$\lambda_j^{t+1} = \lambda_j^t + \alpha^t \tilde{\nabla} q(\lambda^t) \tag{1.4}$$

The update rule can be written as follow:

$$x_i^{t+1} = \arg\min_{x_i \in X_i} f_i(x_i) + (\lambda^t)^T h_i \tag{1.5}$$

The computation of x_i^{t+1} is now like a minimization of N independent problem which can be solved separately. We can appreciate the fact that through duality we have parallelized the computation and we can think of solve large scale constraint coupled in a master-client architecture.

1.2.1 Matlab linprog

We solved the centralize computation with the *linprog* MATLAB function [2], which finds the minimum of a problem specified by:

$$\min_x f^T x \quad \text{such that} \begin{cases} Ax \leq b \\ A_{eq}x = b_{eq} \\ lb \leq x \leq ub \end{cases}$$

where lb and ub represent the lower and the upper bounds.

We will use the centralized solution as a comparison with the distributed one to appreciate the result.

1.3 Distributed setup

In order to apply a distributed algorithm we need to determine the topology of the communication graph. In particular, in our example we used a customized function *binomialGraph* to simulate the connections as in a binomial graph (see definition in Appendix A.4). Moreover, in order to guarantee the convergence, we need to have a "doubly stochastic" graph (see Consensus Theorem A.3)

1.4 Dual formulation

In order to solve (1.1) we should use dual formulation. We can consider the Lagrangian function associated to 1.1 defined as

$$\mathcal{L}(z, \lambda) = f_i(z) + \sum_{l=1}^{m} \lambda_l h_l \qquad i \in \{1, \dots, N\} \quad l \in \{1, \dots, m\}$$

where $f_i(z) = c_i^T z_i$ and $h_l = H_i z_i - b$

We can define the **dual function** as

$$q(\lambda) = \inf_{z \in P_i} \mathcal{L}(z, \lambda)$$

which domain is

$$D = \{\lambda \in \mathbb{R}^m \quad | \quad q(\lambda) > -\infty\}$$

The Primal update

$$z^{t+1} = \arg\min_{z \in P_i} \mathcal{L}(z, \lambda^t)$$

knowing that the subgradient of $q(\lambda)$ is given by

$$\tilde{\nabla} q(\lambda^t) = \tilde{\nabla}_\lambda q(\lambda^t) = \begin{bmatrix} h_l(z^{t+1}) \\ \vdots \\ h_m(z^{t+1}) \end{bmatrix}$$

Basically, in our example, in order to compute the gradient you have to minimize the Lagrangian function with respect to λ and then evaluate it at $\lambda = \lambda^t$.

The proof of the Theorem can be found in [4].

1.4.1 Distributed Dual Subgradient

Once we have the adjacency matrix (definition in Appendix A.1), we can develop the distributed version of the dual subgradient algorithm. In general we must be able to work also with non-smooth convex function f and x constrained to a close, convex set X. So we have to use a subgradient method [4] .

The following formula are referred to a single agent i:

$$v_i^{t+1} = \sum_{k \in \mathcal{N}^i} a_{ik} \lambda_k^t = \sum_{k=1}^{N} a_{ik} \lambda_k^t$$

Note: a_{ij} are defined only for $(j, i) \in E$. Namely $a_{ij} = 0$ is $(j, i) \notin E$.

the update is

$$z_i^{t+1} = \arg\min_{z_i \in Z_i} f_i(z_i) + (v_i^{t+1})^T h_i(x_i) \qquad i \in \{1, \dots, N\}$$

where

$$f_i(z_i) = c_i z^t \qquad \text{and} \qquad h_i(z_i) = H_i z_i - b$$

then the final update is

$$\lambda_i^{t+1} = v_i^{t+1} + \alpha^t h_i(x_i^{t+1})$$

The stepsize at iteration t, α^t is computed as

$$\alpha^t = k \left(\frac{1}{t} \right)^q$$

where k is a scalar term and $q \in (0.5, 1]$ (see Appendix B)

1.5 Code Implementation

At the beginning of the code, the dimensions of the system are defined: number of agents N, dimensions of the agents n_i, number of constraints. System matrices are written below. Note that we considered $S = n_i$.

$$c = \begin{bmatrix} c_1 & \dots & c_N \end{bmatrix}_{n_i \times N} \qquad \text{where} \quad c_i = \begin{bmatrix} c_{i1} \\ \vdots \\ c_{in_i} \end{bmatrix}$$

$$z = \begin{bmatrix} z_1 & \dots & z_N \end{bmatrix}_{n_i \times N} \qquad \text{where} \quad z_i = \begin{bmatrix} z_{i1} \\ \vdots \\ z_{in_i} \end{bmatrix}$$

$$H = \begin{bmatrix} H_1 & \dots & H_N \end{bmatrix}_{S\times(n_i\cdot N)} \qquad H_i = I_{S\times n_i} \qquad b = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix}_{S\times 1}$$

Definition of the polyhedron matrix:

$$D = \begin{bmatrix} D_1 & \dots & D_N \end{bmatrix}_{N\times(n_i\cdot N)} \qquad d = \begin{bmatrix} d_1 \\ \vdots \\ d_N \end{bmatrix}_{N\times 1}$$

where d acts as the term of the inequality constraint.

After defining the matrices, we simulated the behavior of the agents in the MATLAB environment. In order to obtain statistically relevant results we performed a Monte Carlo simulation with different cases. The results are shown in the following plots.

Monte Carlo Simulations

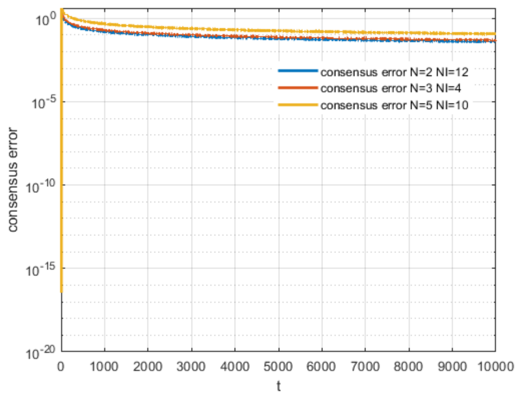


Figure 1.1: Task 1 - Consensus error

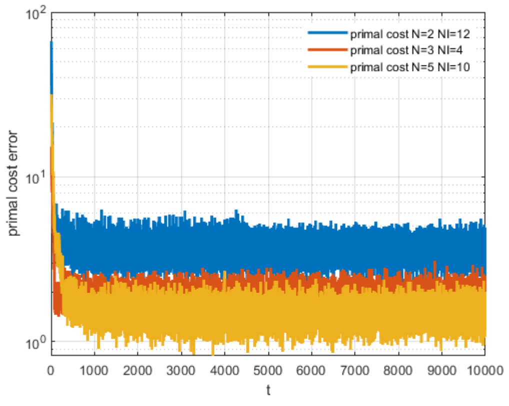


Figure 1.2: Task 1 - Primal cost average

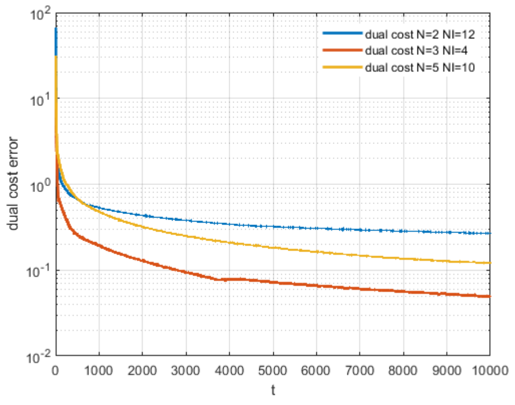


Figure 1.3: Task 1 - Dual cost and Dual cost running average

In order to perform a Montecarlo simulation we perform the same experiment twenty times and recorded the results. Each time the only things that changes is the position of the task and the initial pose of the agents.

The graph associated to the previous results was a *binomial graph* with $p = 0.5$. Once we have collected all the data we merged them into a structure and we compute the mean of the data at each time instant and then we put them in a graph. As we can see from the plot 1.1 we notice that the consensus error gets very small meaning that our algorithm reach a consensus. Additionally, the dual cost error 1.3 is lower than the primal one in figure 1.2. In figure 1.3, we can observe that the smaller is the dimension of the agents, the smaller will be the dual cost error. If there are more agents, the dual cost error goes to zero slowly and the difference between the dual and primal cost errors become smaller.

Chapter 2

Task Assignment

In this chapter, we will see how a constraint coupled formulation, with the tool of the distributed dual subgradient, can solve a Task assignment problem.

2.1 Task 2 formulation

Consider a team of N robots that want to self-assign a set of N task scattered in the environment. Each robot has to serve a task and each task must be served by at most one robot. Robots want to minimize the total travelled distance. The assignment problem can be illustrated with a bipartite assignment graph $G_A = \{V_A, U_A, E_A\}$, where the nodes sets $V_A = \{1, \dots, N\}$ and $U_A = \{1, \dots, N\}$ represent the set of agents and tasks respectively. An edge $(i, k) \in E_a$ exists if and only if agent i can be assigned to the task k. The cost c_{ik} for agent i to perform the task k is a weight on the edge. An illustrative example is depicted in figure 2.1.

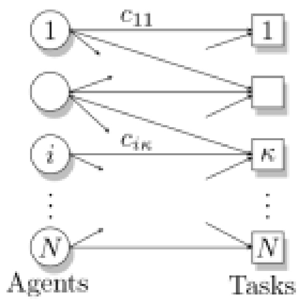


Figure 2.1: Task 2 - Bipartite graph for the task assignment problem

This problem can be cast as the following linear optimization program:

$$\min_x \sum_{(i,k)\in E_A} c_{ik} x_{ik} \tag{2.1}$$

$$\text{subj.to} \begin{cases} 0 \le x \le 1 \\ \displaystyle\sum_{k|(i,k)\in E_A} x_{ik} = 1 \qquad \forall i \in \{1, \dots, N\} & (2.2) \\ \displaystyle\sum_{i|(i,k)\in E_A} x_{ik} = 1 \qquad \forall k \in \{1, \dots, N\} & (2.3) \end{cases}$$

where x is the variable stacking all x_{ik} for all i, k.

The equation (2.2) represents the local constraint. This indicates that at most one task can be assigned to an agent.

The equation (2.3) represent the coupling constraint. This means that at most one agent can reach a task.

Problem (2.1) can be cast in the form (1.1) by defining z_i and c_i as the vector stacking all x_{ik} and c_{ik} (respectively) for all k such that $(i,k) \in E_A$, and by defining the local polyhedra P_i as

$$P_i = \left\{ x_i \mid 0 \leq x_i \leq 1 \quad \text{and} \sum_{\{k|(i,k)\in E_A\}} x_{ik} = 1 \right\}, \quad i \in \{1,\dots,N\}$$

and, finally, by defining suitably the H_i matrices and the b vector.

Note: task positions can be generated randomly, but it is necessary to make sure the optimal solution is unique. To this end, add a random, small perturbation to the c_i vectors.

2.2 Problem approach

In order to cast the problem (2.1) into the one in task 1, as suggested, we defined new matrices:

$$z_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ik} \\ \vdots \\ x_{iN} \end{bmatrix} \qquad c_i = \begin{bmatrix} c_{i1} \\ \vdots \\ c_{ik} \\ \vdots \\ c_{iN} \end{bmatrix}$$

The matrix C can be seen as the matrix collecting all the distance from each agent to each task. We computed the Euclidean distance:

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \qquad \text{for mobile robots that work in the plane}$$

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \qquad \text{for robots that work in the 3D space}$$

where (x_i, y_i, z_i) are the agents coordinates and (x_j, y_j, z_j) are the tasks coordinates.

The matrix z can be seen as the selection matrix, so it tells us which tasks have been assigned to the agents. In order to explicit the capability of each agent to perform or not a single task we define a matrix whose dimension depends on the number of agents. This matrix S is generated randomly with the function *Binord* from *Statistics and Machine Learning Toolbox.*

Now in order to manage element with different size we cannot use anymore a matrix data structure. So from now on we will use a *cell array* data structure for most of our elements.

The following is an example of the selection matrix S:

$$S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Each column indicates one agent and each row indicates a task, so if the element S_{ij} is 1 it means that agent i can do task j. For example, if we look at the first column we see that agent 1 can do tasks 1 and 3.

So we get

$$z_1 = \begin{bmatrix} x_{11} \\ x_{13} \end{bmatrix} \qquad z_2 = \begin{bmatrix} x_{22} \\ x_{23} \end{bmatrix} \qquad z_3 = \begin{bmatrix} x_{31} \\ x_{32} \\ x_{33} \end{bmatrix}$$

2.3 Solution 1: Equally capable agents

In this first solution we considered the most general configuration. All agents can solve all tasks since we assume that they are drones, namely they can move in all 3D direction and can reach both ground tasks and air tasks, without any limitation. We have a complete bipartite graph. After we have fixed the number of agents N, we randomly generate initial pose of the agents and the tasks in the space:

$$C = \begin{bmatrix} 0.4701 & 1.0318 & 0.6226 \\ 0.4423 & 1.1595 & 0.3425 \\ 0.8368 & 0.6746 & 0.7033 \end{bmatrix} \qquad ZZ\_RA = \begin{bmatrix} 0.9993 & 0.0002 & 0.0005 \\ 0.0006 & 0 & 0.9994 \\ 0.0001 & 0.9998 & 0.0001 \end{bmatrix}$$

The tasks are chosen if the value of the running average is greater than the threshold, that is an arbitrary value. We made a trade-off between the number of iterations and the threshold. After some tests, we end up with 10k iterations and a threshold of 0.9.

Then we perform the distributed dual subgradient that will give us the task assignment in order to minimize the total travelled cost which can be seen as the distance.

Plots - Solution 1

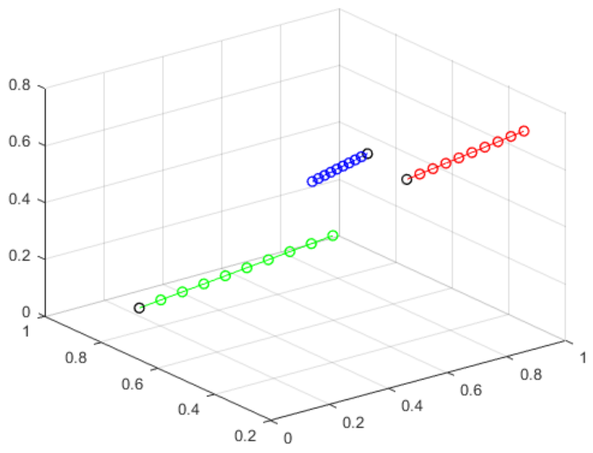


Figure 2.2: Task 2, Solution 1 - Animation

Note that with black dots we denote the task position and we plot with colored dots the evolution of the trajectories of each agent.

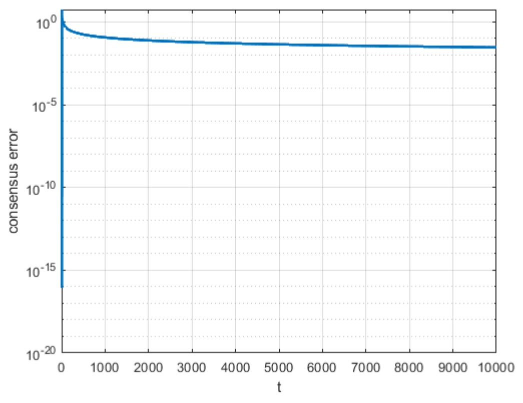


Figure 2.3: Task 2, Solution 1 - Consensus error

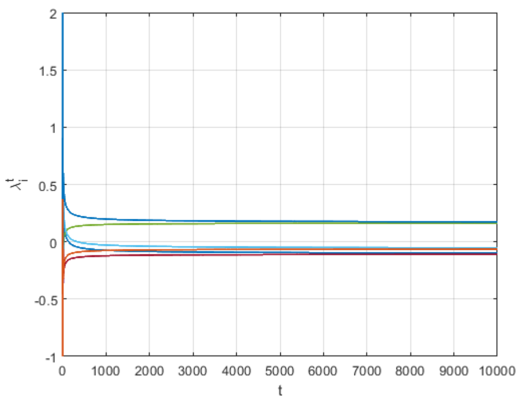


Figure 2.4: Task 2, Solution 1 - Lambda

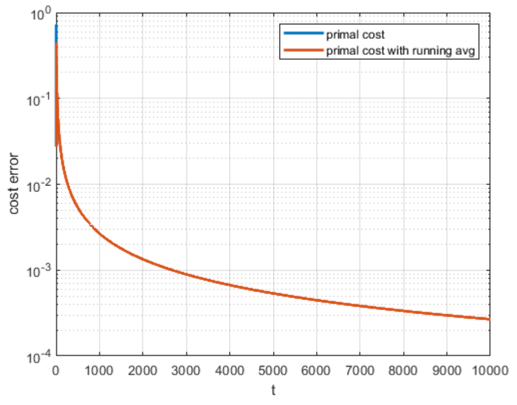


Figure 2.5: Task 2, Solution 1 - Primal cost and Primal cost with Running Average

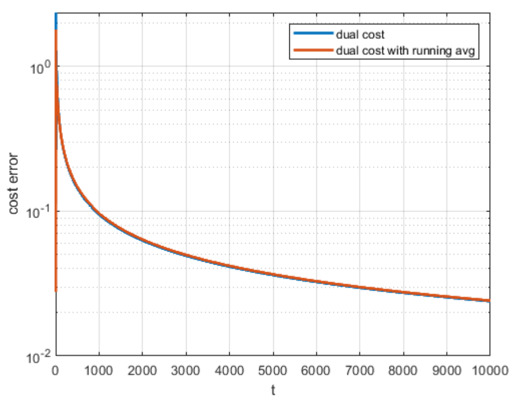


Figure 2.6: Task 2, Solution 1 - Dual cost and Dual cost with Running Average

All the previous plot are done with *binomial graph* with $p = 0.5$ and with step size $\alpha = 3$.

As we can see from all the plot, as expected from asymptotic algorithm, we go to zero with all the error's cost. In this particular case, running only once the algorithm, we have noticed that the primal cost goes to zero faster then the dual one.

Agents	A1	A2	A3
Tasks	1	3	2

Table 2.1: Task Assignment equally capable agent

The table 2.1 show us the agent-task assignments. As we can see each task has been assigned to a different agent.

2.4 Solution 2: Decoupled problem

Assumption
The number of agent of each type is exactly equal to the number of tasks. In particular we considered the case in which we have five mobile robots that perform five tasks on the ground and five drones that perform 5 tasks in the 3-D Space.

2.4.1 Problem formulation

We report the matrices dimensions for clearer explanation. The agent can perform the following tasks.

$$z_1 = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix} \quad z_2 = \begin{bmatrix} x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{25} \end{bmatrix} \quad z_3 = \begin{bmatrix} x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{35} \end{bmatrix} \quad z_4 = \begin{bmatrix} x_{41} \\ x_{42} \\ x_{43} \\ x_{45} \end{bmatrix} \quad z_5 = \begin{bmatrix} x_{52} \\ x_{53} \\ x_{55} \end{bmatrix}$$

$$z_6 = \begin{bmatrix} x_{66} \\ x_{68} \end{bmatrix} \quad z_7 = \begin{bmatrix} x_{77} \\ x_{79} \\ x_{710} \end{bmatrix} \quad z_8 = \begin{bmatrix} x_{87} \\ x_{89} \\ x_{810} \end{bmatrix} \quad z_9 = \begin{bmatrix} x_{96} \\ x_{98} \\ x_{99} \\ x_{910} \end{bmatrix} \quad z_{10} = \begin{bmatrix} x_{106} \\ x_{108} \\ x_{109} \\ x_{1010} \end{bmatrix}$$

The local constraint can be written in a matricial form, where the tasks are in the rows and the agents in the columns. If an agent cannot do a certain task we skip a column, so we end up with matrices in which are only present the tasks that can be reached by its agent. Since we satisfied the problem constraint (2.2), we obtain the following matrices H_i:

$$H_1 = H_2 = H_3 = \begin{bmatrix} 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{bmatrix} \quad H_4 = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&0 \\ 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} \quad H_5 = \begin{bmatrix} 0&0&0 \\ 1&0&0 \\ 0&1&0 \\ 0&0&0 \\ 0&0&1 \\ 0&0&0 \\ 0&0&0 \\ 0&0&0 \\ 0&0&0 \\ 0&0&0 \end{bmatrix}$$

$$H_6 = \begin{bmatrix} 0&0 \\ 0&0 \\ 0&0 \\ 0&0 \\ 0&0 \\ 1&0 \\ 0&0 \\ 0&1 \\ 0&0 \\ 0&0 \end{bmatrix} \quad H_7 = H_8 = \begin{bmatrix} 0&0&0 \\ 0&0&0 \\ 0&0&0 \\ 0&0&0 \\ 0&0&0 \\ 0&0&0 \\ 1&0&0 \\ 0&0&0 \\ 0&1&0 \\ 0&0&1 \end{bmatrix} \quad H_9 = H_{10} = \begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 1&0&0&0 \\ 0&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}$$

The coupling constraint (2.3) can also be written in a matricial form. The matrices can be read in this way: the rows represent the agents and the columns represent the tasks. As in the case of H_i matrix, if an agent cannot reach a certain task, we skip the column associated with that task. The matrices G_i are defined in a similar way.

$$G_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad G_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$G_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad G_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad G_6 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$G_7 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad G_8 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad G_9 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$G_{10} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Plot Binomial p=0.2

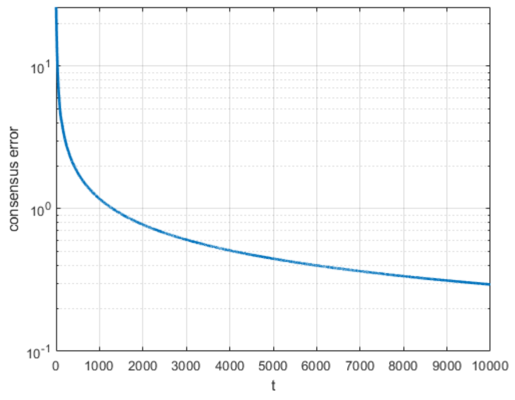


Figure 2.7: Task 2, Solution 2 - Consensus error

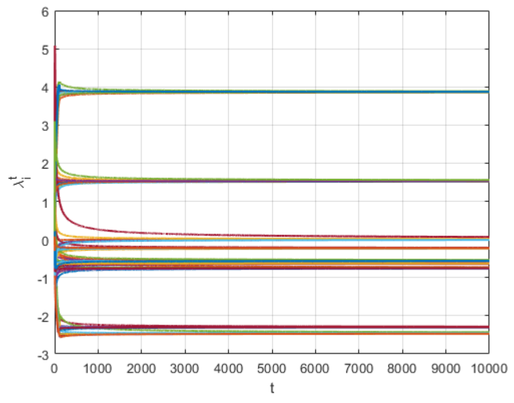


Figure 2.8: Task 2, Solution 2 - Lambda

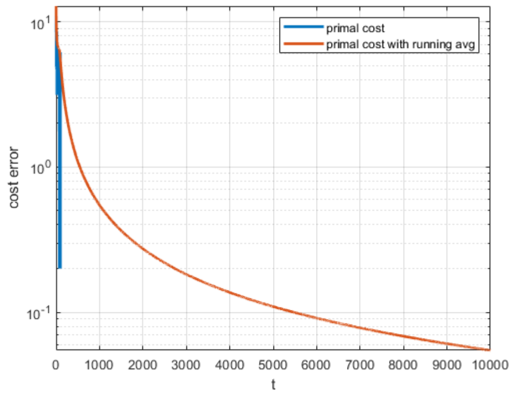


Figure 2.9: Task 2, Solution 2 - Primal cost and Primal cost with Running Average

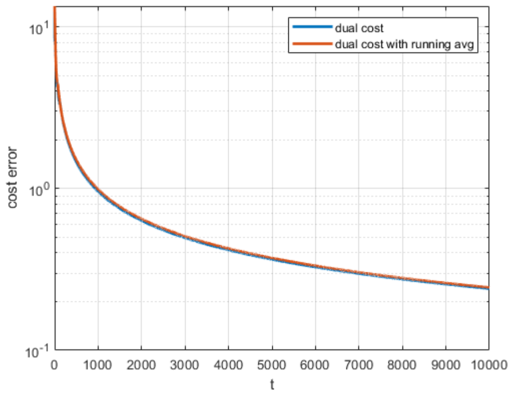


Figure 2.10: Task 2, Solution 2 - Dual cost and Dual cost with Running Average

As we can see all the error's plot are going to zero. Lambda's plot 2.5 tell us that the multipliers goes to their optimal value. This is due because Theorem (Distributed Dual Subgradient) C.9 holds.

All the previous plot are done with *binomial graph* with $p = 0.2$ and with step size $\alpha = 3$.

Assignment table Binomial

At the end of the optimization algorithm we end up in the following assignment set

Agent	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
Task	4	2	1	3	5	8	7	10	9	6

Table 2.2: Assignment Binomial Graph table

Plot animation Decoupled solution

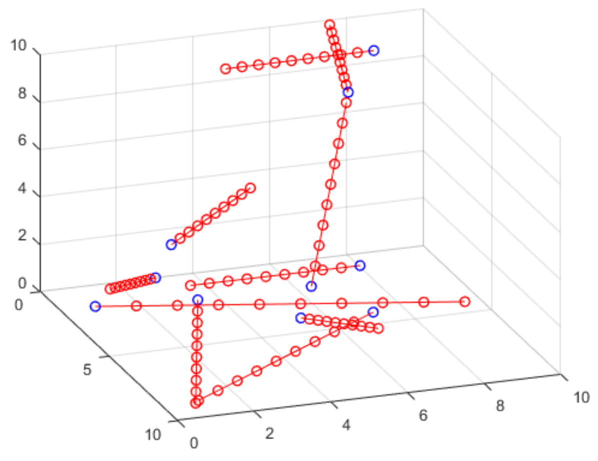


Figure 2.11: Task 2, Solution 2 - Animation

As we can see from the table and from the plot, the first five tasks are done by the first five agents, that are all mobile robots. The remaining agents are drones and they did the aerial tasks.

2.5 Solution 3: Coupled problem

In this last part of our project we tried to relax the assumptions of the decoupled problem. In particular drones are allowed to perform also tasks on the ground. We will see some comparison between the binomial graph and the cyclic graph and we will test different scenarios in which the number of tasks and the number of agents are different.

Binomial Graph

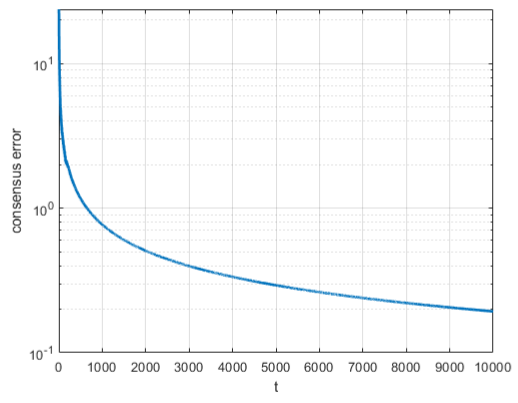


Figure 2.12: Task 2, Solution 3 - Consensus error

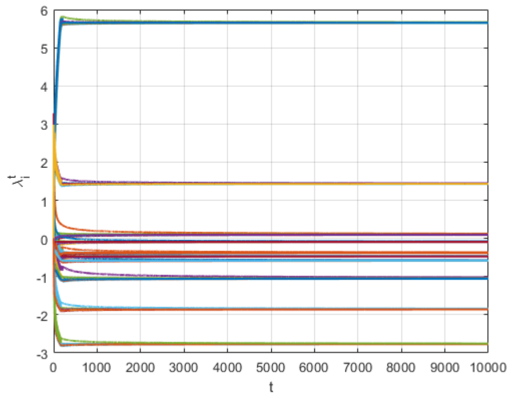


Figure 2.13: Task 2, Solution 3 - Lambda

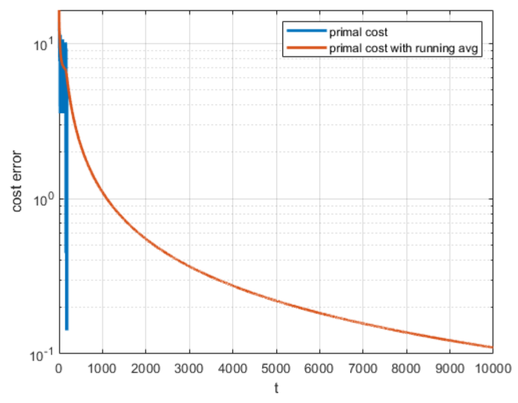


Figure 2.14: Task 2, Solution 3 - Primal cost and Primal cost with Running Average

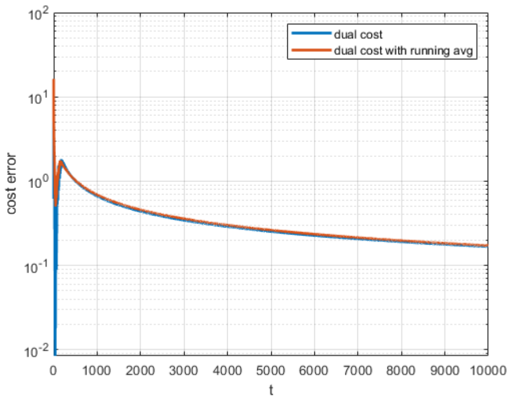


Figure 2.15: Task 2, Solution 3 - Dual cost and Dual cost with Running Average

This simulation is done with 3 mobile robots, with their 5 ground tasks, and with 7 drones with their 5 air tasks. The *binomial graph* has $p = 0.2$ and the step size is $\alpha = 3$.
We tried a configuration that forces some robot to solve a ground task and as we can see the algorithm find an optimal solution, all the errors go to zero.

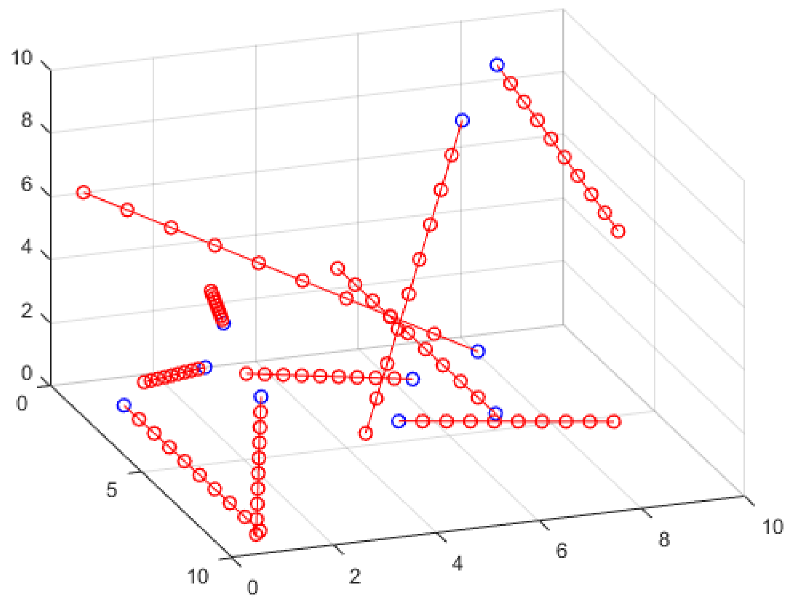


Figure 2.16: Task 2, Solution 3 - Animation

Agent	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
Task	4	2	3	9	1	7	5	8	10	6

Table 2.3: Assignment Binomial Graph table

CyclicGraph
We tested also the distributed algorithm with the cyclic graph and we get good result.

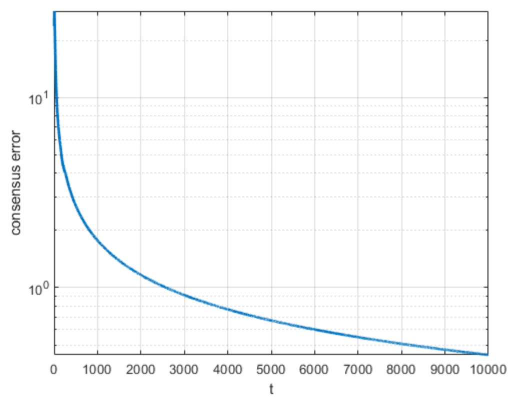


Figure 2.17: Task 2, Solution 3 cyclic - Consensus error

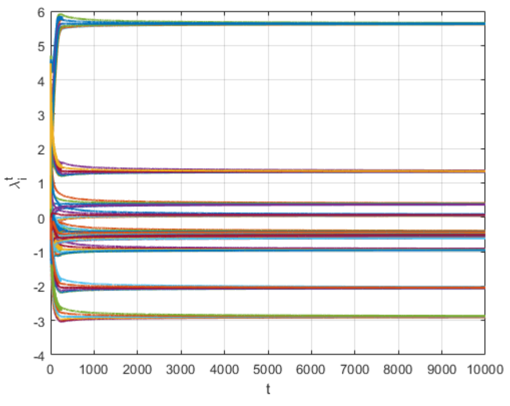


Figure 2.18: Task 2, Solution 3 cyclic - Lambda

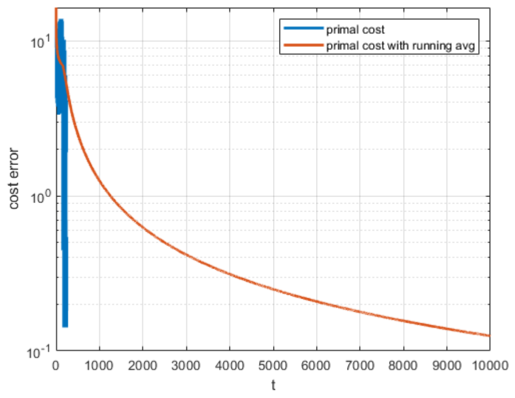


Figure 2.19: Task 2, Solution 3 cyclic - Primal cost and Primal cost with Running Average

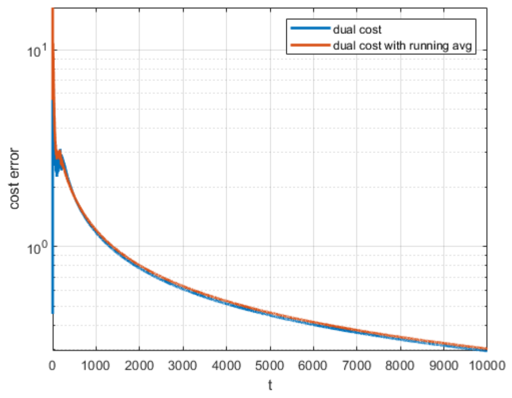


Figure 2.20: Task 2, Solution 3 cyclic - Dual cost and Dual cost with Running Average

We show that even if consider a less connected digraph such as the cyclic graph the algorithm will converge to the optimal solution.

2.5.1 Critical Case

Let's assume that at some point in out factory we have less ground tasks than the number of mobile robots. We have the problem that two mobile robot will be assigned to the same task, violating one of the two constraints. Further improvement of the project can be done in handle this type of problem.

Agents	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
Tasks	3	1	1	3	6	10	5	8	7	6

As we were expecting there is one task done by two robots and one task that it is not done by anyone else. This is an extreme case but we think that it is important to point it out. A trivial way to avoid this problem is to not allow more mobile robots than ground tasks.

p	# mobile robots	# drones	# ground tasks	# air tasks
1	4	6	3	7

Moreover, the algorithm returns the following error:

```
No feasible solution found.
Linprog stopped because no point satisfies the constraint.
```

This tells us that the algorithm is aware of the unusual configuration and cannot compute a complete task assignment.

2.5.2 Solution 3b: Coupled problem with perturbations

In this section we will extend the coupled task assignment problem to the case in which we have some perturbations in the cost function. These perturbations can be due to interferences, measurement errors.

We introduced the *perturbation vector*:

$$\Delta^T = \left[\Delta, \Delta^1, \ldots, \Delta^N\right]^T$$

We added the perturbation vector to the distance vector c_i in order to execute the optimization on the whole set of data.

$$\begin{aligned} \min_{x} \quad & (c_i + \Delta)^T x_i \\ \text{subj.to} \quad & a_k^T x \leq b \qquad k \in \{1, \ldots, N\} \end{aligned} \tag{2.4}$$

The results are quite similar to the previous case, as we expected. The only noticeable difference is the overall cost of the optimization.

Agents	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
Tasks	4	2	1	3	5	7	6	10	9	8

Adding the perturbation is useful when there are more optimal solutions. For instance, if the algorithm assigned two different tasks to the same agent, the assignment value will be 0.5 for each task. The sum will be always 1 but these values are meaningless. If the problem can have more optimal solutions, it could be a good idea to check the robustness of the algorithm by adding a perturbation to the cost.

Animation

In order to implement the animation we have chosen to use a simple linear trajectory. Basically, once we ran the algorithm and it did the the assignment of each agent, through the function *linspace* we interpolated a simple linear trajectory from the starting position of the agent to the task position. We assumed that there are no obstacles in the environment and we can move along a straight line. Since navigation was not the goal of this project we preferred to simplify the motion animation.

Conclusions

In this project we have shown the distributed dual subgradient method applied to linear programs. We started from the general setup explained in Chapter 1 and, after a brief introduction, we implement a solution to the problem. Then, through plot in section 1.5, we saw a decreasing behaviour in the cost function, primal and in particular the dual one. This means that the algorithm updates, at each iteration, goes toward the optimal solution obtained in the centralized algorithm.
In chapter 2 we analyzed a common scenario in distributed cooperative robotics in which the agents have to self-assign the tasks. In section 2.3 we defined the simplest case where every agent can be assigned to every task in the system, showing that they are able to reach the optimal solution. In section 2.4 (Decoupled problem) we introduced two main differences, first we had a separation in the typology of task that an agent can solve, namely we have drones and mobile robots, and we assumed that drones cannot work on the floor and mobile robots cannot fly. The second difference is that among the agents that could ideally reach the task, we introduce a degree of freedom that allow you to not assign that specific task to an agent. We use this trick to model the fact that an agent cannot be assigned to a specific task. In section 2.5 (Coupled problem) we tried to relax the assumptions of separability of the assignment, namely we wanted to have that in some cases drones can solve tasks on the floor.
In conclusion we showed how can be solved a task-assignment problem in a distributed way.

Appendix A

Preliminaries in Graph Theory

A.1 Matrices associated to graphs

Weighted adjacency matrix $N \times N$ non-negative matrix A s.t. the entry (i,j) of A, denoted a_{ij},satisfies: $a_{ij} > 0$ if $(i,j) \in E$, and $a_{ij} = 0$ otherwise.

Adjacency matrix: matrix A with entries $a_{ij} = 1$ if $(i,j) \in E$, and $a_{ij} = 0$ otherwise.

A.2 Stochastic matrices

The non-negative (≥ 0) square matrix $A \in \mathcal{R}^{NxN}$ is

- **row stochastic** if $A1 = 1$. Meaning that 1 is an eigenvector with eigenvalue 1. (each rows sums to 1)
- **column stochastic** if $1^T A = A^T 1 = 1$. (each column sums to 1)
- **doubly stochastic** if both row and column stochastic

A.3 Consensus result (statement)

Theorem (Consensus): Consider a discrete-time averaging system with associated digraph G and weighted adjacency matrix A. Assume G is strongly connected and aperiodic, and assume A is row-stochastic. Then

1. there exists a left eigenvector $w \in \mathcal{R}^N, w > 0$ (i.e., with positive components $w_i > 0$, $\forall i \in \{\dots, N\}$) such that

$$\lim_{t\to\infty} x(t) = \frac{w^T x(0)}{w^T 1} 1 = \frac{\sum_{i=1}^{N} w_i x_i(0)}{\sum_{i=1}^{N} w_i} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

   the meaning of that $x_i(t)$ are converging to a vector of 1 is that

$$\lim_{t\to\infty} \begin{bmatrix} x_1(t) \\ \vdots \\ x_N(t) \end{bmatrix} = \begin{bmatrix} \alpha \\ \vdots \\ \alpha \end{bmatrix} \quad \text{asymptotically consensus}$$

i.e., consensus (value) is reached to $\alpha = \underbrace{\frac{\sum_{i=1}^{N} w_i}{\sum_{i=1}^{N} w_i}}_{\gamma_i} x_i(0)$ where $\sum_{i=1}^{N} \gamma_i = 1$

In other word the consensus value is a so called **convex combination** of the initial states.

2. if additionally A us doubly stochastic, then

$$\lim_{t\to\infty} x(t) = \frac{\sum_{i=1}^{N} x_i(0)}{N} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

i.e., average consensus is reached.

Namely adding this second assumption the convex combination becomes the average.If the matrix is doubly stochastic not only we agree to a common value, moreover this value is the average of the initial states.

A.4 Binomial graph

In graph theory, we can generate a *Random binomial graph, G(n,p)*. This model has two parameters, the number of vertices n and a probability parameter $0 \leq p \leq 1$

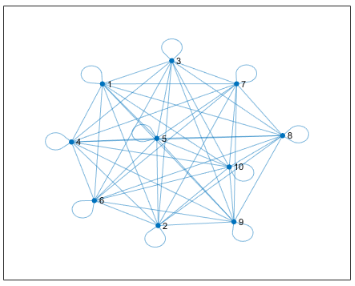


Figure A.1: Generic binomial graph

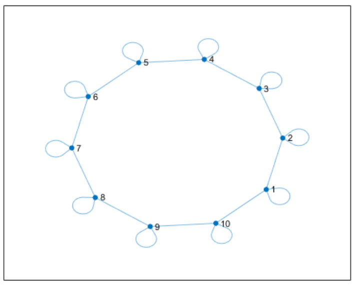


Figure A.2: Generic cyclic graph

A.5 Cyclic graph

In graph theory, a cycle graph is a directed graph that contains a path from at least one node back to itself. In simple terms, cyclic graphs contain a cycle. The weighted adjacency matrix with $N = 5$ is:

$$AA_{cyclic} = \begin{bmatrix} 1/3 & 1/3 & 0 & 0 & 1/3 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 1/3 & 0 & 0 & 1/3 & 1/3 \end{bmatrix}$$

Appendix B

Gradient Methods: step-size selection rules

A brief recap of the possible choice of the step-size. For further reading see [1]

- Constant step-size $\alpha^t = \alpha > 0 \quad \forall t = 0, 1, 2, \dots$
- Diminishing step-size
  basically we chose a sequence of step-size $\alpha^1, \alpha^2, \dots$ such that $\alpha^t \to 0$ as $t \to \infty$

  Intuition: since I'm updating with the step-size the direction, if I chose a step-size that goes to zero too fast then at some point I'm not moving enough. For example $\alpha^1 = 0.5, \alpha^2 = 0.5, \alpha^3 = \alpha^n = 0$ then this is a sequence that goes to zero in a finite time, it does not work because at some point I'm not moving anymore.

  It must be avoided that becomes too small to guarantee substantial progress. For this reason we require

  $$\sum_{t=0}^{\infty} \alpha^t = \infty$$

  Typical choice

  $$\sum_{t=0}^{\infty} \alpha^t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} (\alpha^t)^2 < \infty$$

  which implies $\alpha^t \to 0$

  Important: it does not guarantee descent at each iteration
- Minimization rule, fixing x^t, d^t then $f(x^t + \alpha d^t)$ becomes a one dimensional function

  If I'm able to evaluate the function at different point and chose the α^t such that I'm getting down as much as possible, then I will pick that α^t. The problem is that you have to compute the function in many points.
- Armijo rule (not treated in this course). It is an approximation of minimum rule.

Appendix C

Duality theory

For this part we referred to [1] and notes of the Distributed Control Systems M course.

Notation:

- (P) means Primal problem;
- (D) means Dual problem

C.1 Constrained optimization problem set-up

$$\min_{x \in X} f(x)$$

$$\text{subj.to} \quad \begin{array}{ll} g_i(x) \leq 0 & j \in \{1, \dots, r\} \\ h_l(x) = 0 & l \in \{1, \dots, m\} \end{array}$$

Assumption (feasibility and boundedness) There exists at least one feasible solution for the optimization problem and the cost is bounded from below, i.e.,

$$-\infty < f^* = \inf_{\substack{x \in X, \\ g(x) \leq 0, \\ h(x) = 0}} f(x) < \infty$$

Note: If $X = \mathcal{R}^d$, f and g_j convex (and differentiable), and h_l affine (linear equality constraints), then convex problem.

C.2 Some definitions

C.2.1 Set of active inequality constraints

For a point $x \in \mathcal{X}$, the set of active inequality constraints at x is $A(x) = \{j \in \{1, \dots, r\} | g_j(x) = 0\}$

C.2.2 Regular point

A point x is regular if the vectors $\nabla h_1(x), \dots \nabla h_m(x)$, and $\nabla g_l(x)$, $j \in A(x)$ are linearly independent

C.2.3 Lagrangian function

$$L(x, \mu, \lambda) = f(x) + \sum_{j=1}^{r} \mu_j g_j(x) + \sum_{l=1}^{m} \lambda_l h_l$$

Note $\mu_j \in \mathcal{R}^r$ (μ will be always used for inequality constraints)and $\lambda \mathcal{R}^m$ (λ will be always used for equality constraints) seen as prices for violating the associated constraint.

Note: We are relaxing the constrained problem, but we cannot minimize the Lagrangian.

C.3 Karush-Kuhn-Tucker necessary conditions

Let x^* be a regular local minimum of (P)

$$\min_{x \in \mathcal{R}^d} f(x)$$

$$subj.to \begin{array}{ll} g_j(x) \leq 0 & j \in \{1, \dots, r\} \\ h_l(x) = 0 & l \in \{1, \dots, r\} \end{array}$$

where f, g_j and h_l are C^1.

Then $\exists$ unique vector $\mu_j^* = (\mu_1^*, \dots, \mu_r^*)$ and $\lambda_l^* = (\lambda_1^*, \dots, \lambda_m^*$, called **Lagrangian multipliers**, s.t

1. $$\nabla_x L(x^*, \mu^*, \lambda^*) = \begin{bmatrix} \frac{\partial L(x,\mu,\lambda)}{\partial x_1} \\ \vdots \\ \frac{\partial L(x,\mu,\lambda)}{\partial x_d} \end{bmatrix}_{\substack{x = x^* \\ \mu = \mu^* \\ \lambda = \lambda^*}} = 0$$

   Note: $\nabla f(x^*) + \sum_{j=1}^r \mu_j^* \nabla g_j(x^*) + \sum_{l=1}^m \lambda_l^* \nabla h_l(x^*) = 0$

2. $$\mu_j^* \geq 0$$
   $$\mu_j^* g_j(x^*) = 0 \quad j \in \{1, \dots, r\}$$

   Note: $\mu_j^* g_j^*(x^*) = 0, \quad j \in \{1, \dots, r\}$, complementary slackness.

The complementary slackness condition is saying, either $g_j(x^*) = 0$ is satisfied with an equality, in that case μ can be positive or equal to zero, or if $g_j(x^*) < 0$ then the associated Lagrangian's multiplier must be equal to 0. Looking the Task assignment problem, it means that at an optimum each agent is assigned to a "best" task.

$$\nabla_x L(x, \mu, \lambda) = \nabla f(x) + \sum_{j=1}^r \mu_j \nabla g_j(x) + \sum_{l=1}^m \lambda_l \nabla h_l(x)$$

$$L(x, \mu, \lambda) = f(x) + \sum_{j=1}^r \mu_j g_i(x) + \sum_{l=1}^m \lambda_l \nabla h_l(x)$$

$$\nabla L(x^*, \mu^*, \lambda^* = 0) = \begin{cases} \nabla_x L = 0 \rightarrow \nabla f(x^*) + \sum_{j=1}^r \mu_j \nabla g_j(x^*) + \sum_{l=1}^m \lambda_l \nabla h_l(x^*) = 0 \\ \nabla_\mu L(x^*, \mu^* \lambda^*) = 0 \rightarrow g_j(x^*) \leq 0 \quad \forall j \in \{1, \dots, r\} \\ \nabla_\lambda L(x^*, \mu^* \lambda^*) = 0 \rightarrow h_l(x^*) = 0 \quad \forall l \in \{1, \dots, m\} \end{cases}$$

but also we have to remember $\mu_j^* \geq 0, \quad \mu_j^* g_j(x^*) = 0 \quad \forall j$

Note: If f, h_l and g_j twice differentiable, second order conditions.

C.4 Duality theory and decomposition methods

C.4.1 Duality: dual function definition

Lagrangian function

$$L(x,\mu,\lambda) = f(x) + \sum_{j=1}^{r} \mu_j g_j(x) + \sum_{l=1}^{m} \lambda_l h_l(x) \qquad \mu \in \mathbb{R}^r, \lambda \in \mathbb{R}^m$$

Dual function

$$q(\mu,\lambda) = \inf_{x\in X} L(x,\mu,\lambda)$$

Remember that our original (P) problem was

$$\min_{x\in X} f(x)$$

$$\begin{aligned} &\textit{subj.to} \quad g_j(x) \le 0 \qquad j \in \{1,\dots,r\} \\ &\textit{subj.to} \quad h_l(x) = 0 \qquad l \in \{1,\dots,m\} \end{aligned}$$

Consider $\bar{x}$ feasible ($g_j(\bar{x}) \le 0 \quad \forall j \; and \quad h_l(\bar{x}) = 0 \quad \forall l$) and take $\bar{\mu}_j \ge 0 \quad \forall j$ then

$$L(\bar{x},\bar{\mu},\lambda) = f(\bar{x}) + \sum_{j=1}^{r} \underbrace{\bar{\mu}_j}_{\ge 0} \underbrace{g_j(\bar{x})}_{\le 0} + \underbrace{\sum_{l=1}^{m} \lambda_l \underbrace{h_l(\bar{x})}_{=0}}_{=0}$$

So when we evaluate the Lagrangian at $\bar{x}$ feasible, $\bar{\mu} > 0$, any λ then what we get is the cost function at $\bar{x}$ + something less then or equal to zero. In other words

$$L(\bar{x},\bar{\mu},\lambda) \le f(\bar{x})$$

$$q(\mu,\lambda) = \inf_{x\in X} L(x,\mu,\lambda)$$

restricting this infimum to $\mu \ge 0$ Take the infimum of this point x rather than evaluate it at $\bar{x}$ restricting the minimization on the point that have $\mu \ge 0$ we have:
Note: $q(\mu,\lambda)$ lower bound to optimal value of f^* of original problem P (primal).

Note: $\inf_{x\in X} L(x,\mu,\lambda)$ may be $-\infty$ for some (μ,λ).

So if I'm able to satisfies this inequality for feasible point then if i minimize q, the minimum point will be a feasible one so there will be some x^* such that $L(\bar{x},\bar{\mu},\lambda) \le f(x^*)$. But with the infimum I'm taking also lower value so it is a bound. So I'm finding value that stay below the optimal cost. So now I have to increase it hopefully touching the optimal cost.

C.5 Dual problem

In some sense I'm finding value that stay below the optimal cost, now I would like to increase this value hopefully reach the optimal cost.

$$\max_{(\mu,\lambda)\in D} q(\mu,\lambda)$$

$$\textit{subj.to} \quad \mu_j \ge 0 \quad \forall j$$

Remark: if we rewrite the problem as a minimization problem

$$\min_{(\mu,\lambda)\in D} -q(\mu,\lambda)$$

$$subj.to \quad \mu_i \geq 0 \quad \forall i$$

this equivalent problem is convex, thus every local minimum is global.

Equivalently, for the dual problem **every local maximum is global.**

C.5.1 Domain and properties of dual function

$$q(\mu, \lambda) = \inf_{x \in X} L(x, \mu, \lambda)$$

Domain D of $q(\mu, \lambda)$

$$D = \{(\mu, \lambda) \in \mathbb{R}^r \times \mathbb{R}^m | q(\mu, \lambda) > -\infty\}$$

Proposition D is convex and and $q : D \to \mathcal{R}$ dual function q is concave over D

Since q is concave $\to -q$ is convex

$$\max_{(\mu,\lambda) \in \mathcal{D}} q(\mu, \lambda) \qquad\qquad \min_{(\mu,\lambda) \in \mathcal{D}} -q(\mu, \lambda)$$

$$subj.to \quad \mu \geq 0 \qquad\qquad subj.to \quad \mu \geq 0$$

And under the assumption made before the dual problem min $-q(\mu, \lambda)$ is always convex

C.6 Subgradient algorithm on the dual problem

Let us apply a projected (sub)gradient algorithm to the dual problem

$$\max_{\mu,\lambda) \in \mathbb{D}} q(\mu, \lambda) \qquad \min_{\mu,\lambda) \in \mathbb{D}} -q(\mu, \lambda)$$

$$subj.to \quad \mu \geq 0 \qquad subj.to \quad \mu \geq 0$$

Projected subgradient algorithm on the dual problem

$$\mu^{t+1} = P_{\mu \geq 0}(\mu^t + \alpha^t \tilde{\nabla}_\mu q(\mu^t, \lambda^t))$$

$$\lambda^{t+1} = \lambda^t + \alpha^t \tilde{\nabla}_\lambda q(\mu^t, \lambda^t)$$

Remark: $P_{\mu \geq 0}(\bar{\mu})$ projection of $\bar{\mu}$ on $\{\mu \in \mathbb{R}^r | \mu_j \geq 0 \forall j \in \{1, \ldots, r\}\}$

$$P_{\mu \geq 0}(\bar{\mu}) \quad max\{0, \bar{\mu}\}$$

How do we compute (sub)gradients of $q(\mu, \lambda)$ at μ^t, λ^t?

C.7 Centralized dual subgradient

We can write the projected subgradient on the dual problem

$$\begin{aligned} \mu_j^{t+1} &= \mathcal{P}_{\{\mu \in \mathbb{R}^d \ | \ \mu \geq 0\}} \big(\mu^t + \alpha^t \sum_{i=0}^{N} \tilde{\nabla} q_i(\mu^t)\big) \\ &= max\{0 \ , \ \mu_j^t + \alpha^t \tilde{\nabla} q(\mu^t)\} \\ &= max\{0 \ , \ \mu_j^t + \alpha^t \sum_{i=1}^{N} \tilde{\nabla} q_i(\mu^t)\} \end{aligned} \tag{C.1}$$

The subgradient of q_i at μ^t can be computed as:

$$x_i^{t+1} = \arg\min_{x_i \in X_i} f_i(x_i) + (\mu^t)^T g_i(x_i) \tag{C.2}$$

C.8 Dual subgradient algorithm

$$
\begin{aligned}
x^{t+1} &= \arg\min_{x\in X} L(x,\mu^t,\lambda^t)\\
\mu^{t+1} &= P_{\mu\geq 0}(\mu^t + \alpha^t g(x^{t+1})\\
\lambda^{t+1} &= \lambda^t + \alpha^t h(x^{t+1})
\end{aligned}
\tag{C.3}
$$

where $g(x^{t+1}) = \begin{bmatrix} g_1(x^{t+1}) \\ \vdots \\ g_r(x^{t+1}) \end{bmatrix}$ and $h(x^{t+1}) = \begin{bmatrix} h_1(x^{t+1}) \\ \vdots \\ h_m(x^{t+1}) \end{bmatrix}$
with $\{\alpha^t\}_{t\geq 0}$ a sequence of positive step-sizes (e.g., diminishing).

Remark: $P_{\mu\geq 0}(\mu^t + \alpha^t g(x^{t+1}) = max\{0, \mu^t + \alpha^t g(x^{t+1}\}$

Note $P_{\mu\in\mathbb{R}^r|\mu\geq 0}\begin{bmatrix}\bar{\mu}_1 \\ \vdots \\ \bar{\mu}_2\end{bmatrix} = \begin{bmatrix} max(0,\bar{\mu}_1) \\ \vdots \\ max(0,\bar{\mu}_r)\end{bmatrix}$

Dual subgradient algorithm

$$x^{t+1} = \arg\min_{x\in x}\left[f(x) + \sum_{j=1}^{r}\mu_j^t g_j + \sum_{l=1}^{m}\lambda_l^t h_l(x)\right]$$

$$
\begin{aligned}
\mu_1^{t+1} &= max\{0, \mu_1^t + \alpha^t g_1(x^{t+1})\}\\
&\vdots\\
\mu_r^{t+1} &= max\{0, \mu_r^t + \alpha^t g_r(x^{t+1})\}\\
\lambda_1^{t+1} &= \lambda_1^t + \alpha^t h_1(x^{t+1})\\
&\vdots\\
\lambda_m^{t+1} &= \lambda_m^t + \alpha^t h_m(x^{t+1})
\end{aligned}
$$

Recap, my goal was to minimize the primal problem. I write the Lagrangian, I evaluated it at some point, I do the minimization and I find x^{t+1}. Now I evaluate each constraints at x^{t+1}, and I update μ_j^{t+1} and λ_i.

Applying this algorithm is like to apply a projected sub algorithm on the dual problem.

With argmin L, x^{t+1} can return more than one minima point.

C.9 Distributed dual subgradient for constraint coupled optimization

Given a primal problem (P)

$$\min_{x_1,\ldots,x_N}\sum_{1=1}^{N} f_i(x_i)$$

$$sub.to \quad \sum_{i=1}^{N} g_{ij}(x_i) \leq 0 \quad j\in\{1,\ldots,r\}$$

$$x_i \in X_i \subset \mathbb{R}^{ni}$$

the dual one (D)

$$\max_{\mu \in \mathbb{R}^r} \sum_{1=1}^{N} q_i(\mu)$$

$$sub.to \quad \mu \geq 0$$

$$q_i(\mu) = \inf_{x_i \in X_i} \left[f_i(x_i) + \sum_{j=1}^{N} \mu_j^T g_{ij}(x_i) \right]$$

Apply a "distributed subgradient" algorithm to the dual problem.

As the first step we aggregate the local variable x_j of the neighbors, remember that a_{ij} are the coefficient of the weighted adjacency matrix A of our (communication) graph G.

$$v_i^{t+1} = \sum_{j=1}^{N} a_{ij} \mu_j^t$$

The second step

$$\begin{aligned} \mu_i^{t+1} &= P_{\mu \in \mathbb{R}^r | \mu \geq 0} \left[v_i^{t+1} + \alpha^t \tilde{\nabla} q_i(v_i^{t+1}) \right] \\ &= max\{0, v_i^{t+1} + \alpha^t \tilde{\nabla} q_i(v_i^{t+1})\} \end{aligned}$$

C.9.1 Distributed dual subgradient algorithm

From neighbors I get

$$v_i^{t+1} = \sum_{k=1}^{N} a_{ik} \mu_k^t$$

$$x_i^{t+1} = \arg\min_{x_i \in X_i} \left[f_i(x_i) + (v_i^{t+1})^T g_i(x_i) \right]$$

μ_k^t is the "solution estimate" of agent k. $\mu_k^t \in \mathbb{R}^r$ $\quad \mu_k^t \begin{bmatrix} \mu_{k1}^t \\ \vdots \\ \mu_{kr}^t \end{bmatrix}$

$$\mu_i^{t+1} = max\{0, v_i^{t+1} + \alpha^t \qquad \underbrace{g_i(x_i^{t+1}\}}_{\tilde{\nabla} q_i(v_i^{t+1}) = \tilde{\nabla} q_i(\mu) \Big|_{\mu = v_i^{t+1}}} \qquad \forall i$$

μ_i^t is the "solution estimate" of agent i. $\mu_i^t \begin{bmatrix} \mu_{i1}^t \\ \vdots \\ \mu_{ir}^t \end{bmatrix}$

I should code that whenever I have μ I should put v_i^{t+1}

We do not know explicitly $\tilde{\nabla}$ but we can recover it from the primal problem

C.9.2 Distributed dual subgradient (Agent i update)

Initialize μ_i^0

Repeat for each t

$$v_i^{t+1} = \sum_{k=1}^{N} a_{ik}\mu_k^t \quad (v_i^{t+1} = \sum_{k\in\mathbb{N}_i} a_{ik}\mu_k^t)$$

the update, completely local minimization. Primal sequences

$$x_i^{t+1} = \arg\min_{x_i\in X_i} f_i(x_i) + (v_i^{t+1})^T g_i(x_i)$$

the final update is

$$\mu_i^{t+1} = max\{0, v_i^{t+1} + \alpha^t g_i(x_i^{t+1})\}$$

I need consensus of μ_i^{t+1} at μ^*

Does x_i^t converges as $t \to \infty$ to x_i^*?(x_i^* a solution component of P)
$\begin{bmatrix} x_1^* \\ \vdots \\ x_N^* \end{bmatrix}$ solution of (P).

Remark recall that A solution of (P) is $\begin{bmatrix} x_1^* \\ \vdots \\ x_N^* \end{bmatrix}$ ans we would like that

$$\begin{bmatrix} x_1^t \\ \vdots \\ x_N^t \end{bmatrix} \to \begin{bmatrix} x_1^* \\ \vdots \\ x_N^* \end{bmatrix}$$

In general we cannot guarantee that

$$\begin{bmatrix} x_1^t \\ \vdots \\ x_N^t \end{bmatrix} \to \begin{bmatrix} x_1^* \\ \vdots \\ x_N^* \end{bmatrix}$$

with x_i^t as in the dual subgradient. Unless we can guarantee that at each t x_i^{t+1} is unique + (technical assumption).

Running average averaging over time

$$\hat{x}_i^{t+1} = \frac{1}{t}\sum_{\tau=0}^{t} x_i^\tau$$

then $\hat{x}_i^t \to x_i^*$

C.10 Distributed dual subgradient convergence

Assumption 1 Let $a_{ij}, i, j \in 1, \dots, N$ be non-negative entries of a weighted adjacency matrix A associated to the undirected graph G, with $a_{ii} > 0$ and A **doubly stochastic**.

Assumption 2 The step-size sequence $\{\alpha^t\}_{t\geq 0}$, satisfies the conditions

$$\alpha^t \geq 0, \sum_{t=0}^{\infty} \alpha^t = \infty, \qquad \sum_{t=0}^{\infty} (\alpha^t)^2 < \infty$$

Assumption 3 For all $i \in \{1, \dots, N\}$: each function f_i is convex, each constraint $\mathbb{X}_i$ is a non empty, compact and convex set; each function g_i is a component-wise convex function. Moreover, there exist $\bar{x}_1 \in \mathbb{X}_1, \dots \bar{x}_N \in \mathbb{X}_N$ such that $\sum_{i=1}^{N} g_i(\bar{x}_i) < 0$

Theorem (Distributed Dual Subgradient) Let Assumption 1,2 and 3d hold. Then, the sequence of dual variables $\{\mu_1^t, \dots, \mu_N^t\}_{t \geq 0}$ generated by the Distributed Dual Subgradient satisfies

$$\lim_{t \to \infty} ||\mu_i^t - \mu^*|| = 0 \qquad i \in \{1, \dots, N\}$$

where μ^* is an optimal solution of the dual problem. Moreover, let for each t, $\hat{x}_i^t = \frac{1}{t} \sum_{\tau=0}^{t} x_i^\tau$, Then, it holds

$$\lim_{t \to \infty} \sum_{i=1}^{N} f_i(\hat{x}_i^t) = f^*$$

$$\lim_{t \to \infty} ||\hat{x}_i^t - x^*|| = 0 \qquad i \in \{1, \dots, N\}$$

where x^* and f^* denote an optimal solution and the optimal cost of the primal problem.

Bibliography

[1] D.P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 1999.

[2] MATLAB. https://it.mathworks.com/help/optim/ug/linprog.html.

[3] I. Notarnicola and G. Notarstefano. Constraint-coupled distributed optimization: A relaxation and duality approach. *IEEE Transactions on Control of Network Systems*, 7(1):483–492, 2020.

[4] Giuseppe Notarstefano, Ivano Notarnicola, and Andrea Camisa. Distributed optimization for smart cyber-physical networks. *Foundations and Trends in Systems and Control*, 7(3):253–383, 2019.

[5] Andrea Testa and Giuseppe Notarstefano. Generalized assignment for multi-robot systems via distributed branch-and-price, 2020.