論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Parallel Distributed Optimization with Diffusion Based Stopping Criterion
著者(和文)	AYKENTAYLAN
Author(English)	Taylan Ayken
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第10012号, 授与年月日:2015年9月25日, 学位の種別:課程博士, 審査員:井村　順一,天谷　賢治,中島　求,早川　朋久,畑中　健志
Citation(English)	Degree:, Conferring organization: Tokyo Institute of Technology, Report number:甲第10012号, Conferred date:2015/9/25, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Tokyo Institute of Technology

Doctoral Thesis

Parallel Distributed Optimization with Diffusion Based Stopping Criterion

Author:
Taylan Ayken

Supervisor:
Dr. Jun-ichi Imura

August 2015

To my wife and parents

Abstract

As the size of systems to be controlled gets larger, distributed optimization is becoming one of the significant topics. This is because large systems take longer to solve in a centralized manner. With distributed optimization, the large system is divided into smaller subsystems and each local optimization problem is solved by an individual computer. This solution is then used by a lover level controller as reference. However distribution, by itself, is not enough as the subsystems have to share calculation results at each iteration and work in a synchronized manner. Former increases the communication costs which can affect systems where communication is handled via radio frequency communication protocols. Latter results in wasted calculation capability as simpler subsystems can solve their local optimization problem faster and have to wait for slower subsystems. Also clock synchronization in networked systems is hard and is another research topic by itself. Finally these systems require a supervisor which collects the convergence status of individual algorithms and signals the whole system to stop optimization if a certain criteria is met.

We can solve the first problem by using event-triggered communication where subsystems only communicate if a certain communication criteria is met. The second problem can be solved by extending the event-triggered algorithm to run asynchronously. Last problem can be solved by using a new stopping criterion called diffusion based stopping criterion.

In this thesis, we propose three parallel distributed optimization algorithms: dual decomposition based distributed optimization, the same algorithm with event-triggered communication and an asynchronous optimization algorithm based on event-triggered communication. We also propose a novel stopping criterion called diffusion based stopping criterion. We then compare all three parallel distributed optimization methods and two stopping criteria by numerical simulations.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Prof. Junichi Imura, for his guidance, advice and making time in his busy schedule to discuss my research. Without his advice and comments this thesis would not have been possible.

I would like to thank Prof. Tomohisa Hayakawa for his valuable criticisms, Ahmet Cetinkaya for our discussion sessions and all other members of Imura, Hayakawa and Nakadai Laboratories for their questions and comments during research presentations. Their feedback helped me to clarify my expressions.

I would like to acknowledge Prof. Koji Tsumura of University of Tokyo and Prof. Takeshi Hatanaka of Tokyo Institute of Technology for their valuable advices during discussion sessions. Their comments provided me with the means to improve my research.

I would also like to thank my wife, Ozgu, for giving me the strength to finish my research and my thesis, my parents for always standing behind me with their support, and my friends for their encouragement.

Taylan Ayken

Tokyo Institute of Technology

Contents

Chapter 1

Introduction

1.1 Motivation

Increase in computational power has led to more attention on complex models and controlling large-scale networked systems with many subsystems. One of the methods employed for solving such difficult control problems is Model Predictive Control (MPC), as it can take physical constraints on the system into account. However, as the size of systems gets larger and control problems get more complex, it requires longer time to solve these problems in a centralized manner if not impossible due to real time deadlines, increased memory, and processor usage. In addition, in a centralized system, the system itself is not robust to changes in the network graph, due to failure of subsystems or loss of connections between subsystems. Such scenarios can be prevented by using a switched model control system which stores multiple models of the system and uses the correct one according to the situation. However, although this option is feasible for small-scale and relatively simple systems, it is not really feasible for large-scale systems as the number of failure possibilities and model representations increase exponentially, which requires more effort for modeling and more storage space in the controller.

Distributed optimization methods have been researched extensively since at least 1980s, i.e. [1, 2] with wide array of applications such as power systems [3–5], water distribution [6], temperature control [7, 8], groups of aircraft [9, 10] and even medical decision making [11]. Distributed MPC is one of the effective methods for solving this kind of problems as the system is divided into smaller subsystems and a controller for each subsystem solves its own optimization problem in a distributed way, which results in the global optimum solution, see e.g., [12–25] for different methods. Dual Decomposition method, see e.g., [26] and [27], is one of the most efficient methods, which decomposes the general cost function into multiple local cost functions which, in turn, can be solved on local

controllers relatively easily. Advantage of it is that as each controller is solving a smaller part of the bigger problem, the optimum solution requires computation power for each controller. Also this method is robust to subsystem failures, similar to [28], as each subsystem, which is a part of the network and the computer system used for running the optimization algorithm on that part, is calculating the solution to the local optimization problem only. If a subsystem failure is detected by neighboring subsystems, they can make the necessary adjustments to their model.

But, the existing distributed algorithms require synchronous calculations followed by broadcasting their results at each iteration, which increases the communication cost. In [29] a solution to this problem is presented, but this approach is not suitable to be applied to single commodity network flow problems and the function used for triggering the update events is directly tied to system dynamics, which complicates the implementation. Also synchronization in networked systems is hard to implement and is a research topic on its own. Furthermore this synchronization means that some subsystems are staying idle, waiting for other subsystems to finish their calculation, which leads to unused processing power.

Also all of these methods require a supervisor, which collects the information about the convergence status of each subsystem and stops the optimization algorithms running on the subsystems when all of them satisfy a certain convergence criteria. This means the supervisor uses global information which contradicts with the essence of distribution. Finally this supervisor is connected to all subsystems with communication lines, which increases system setup cost and introduces a single point of failure to the system: If the supervisor malfunctions, the whole system malfunctions as subsystems will not be able to stop and get stuck in an infinite loop.

1.2 Purpose of this thesis

In this thesis we propose a novel asynchronous optimization method that solves the above mentioned problems: communication cost increase, implementation of synchronization and idle processes. We show a conventional dual decomposition based optimization method as a basis and identify its shortcomings. Then we implement event-triggered communication so that we can decrease the communication costs by disposing the need to announce new results to all neighbors by using local estimates. This novel method is still synchronous so we eliminate the need for synchronization and finally present our novel asynchronous optimization method. First two methods have been presented in our conference paper, [30], and journal paper, [31].

Also this thesis proposes a new stopping criterion for distributed systems, called diffusion based stopping criterion which eliminates the need for a supervisor. This way we truly implement a distributed optimization method where there is no need for the conventional supervisor which requires global information. In this method subsystems can determine the convergence status of the whole network. A preliminary version of this has been presented in our conference paper, [32]. In [33], we further developed and applied this method to event-triggered communication method and it is expected to be published shortly.

Although we focus on single commodity network flow problems, we believe with minor modifications these methods can be applied to a diverse selection of large-scale networked system optimization problems.

1.3 Outline

This thesis is organized as follows.

In chapter 2, first, we introduce single commodity network flow problems, then show the problem formulation that we will use throughout this thesis. We present possible connections types between subsystems and the exogenous signal that we are trying to satisfy. We also present some properties of the matrices that mathematically define these connections. We then define the problem and finally show a simple example system to show how the connections between subsystems are translated to mathematical equations.

In chapter 3, we introduce the dual decomposition based optimization method that will form the basis of other methods we developed. We show how we can modify the problem we are trying to solve so that all equations are decoupled and can be solved by individual subsystems. We then define a convergence criterion that is used for detecting when a solution has been found. Afterward we present the algorithm in easily understandable steps. However, this basic method has some shortcomings so we introduce the event-triggered communication method that can solve some of the problems of the previous method. We define local estimates, when and how they will be updated and show a simple example for the update operation. Finally we present the algorithm in steps.

In chapter 4, we talk about possible problems related to synchronization requirement of the previous methods and present the asynchronous optimization method as a solution. We redefine our local estimates by using local iterations and modify the update operation to use the local iteration. We then show the step by step algorithm for this method.

In chapter 5, we analyze the supervisor based stopping criterion that is used by almost all distributed optimization methods. As it requires global information, we define a new method named diffusion based stopping criterion that only uses local information and can determine the convergence status of all subsystems. We define the communication graph and the stopping criterion matrix that is used by this method. Then we show the algorithm for this new method and give an example that shows the changes on the stopping criterion matrices of a system as each subsystem converges.

In chapter 6, we combine these three distributed optimization methods and two stopping criteria and get six algorithms that we will analyze with numerical simulations. We show an example system and present the methods used to write the simulation codes. Finally we analyze calculation time, number of iterations, number of update events and cost at each time step.

Chapter 2

Problem Description

2.1 Single Commodity Network Flow Problems

In single commodity network flow problems we try to satisfy the demand created by users for a certain commodity. This commodity can be electricity, water, gas or other possible demanded good. Even car traffic can be used. User demand is already known or estimated for a certain horizon and the optimization algorithm tries to satisfy this demand while minimizing cost related to transporting the commodity and taking network constraints into account.

As an example if we look at a water system for a small town we can see that there are houses, businesses and other consumers which create the demand. This demand can be estimated by the water company by using past usage trends, weather reports and other available data. Then this system should be able to generate this commodity. This is usually done by wells, water purification and reverse osmosis plants. There are also storage subsystems in place as water tanks at houses, water towers for certain neighborhoods and cisterns. This system also has some constraints associated with it: Water pipes connecting these subsystems have a certain maximum flow capacity, generator subsystems have a certain limit they can produce at a certain time and storage subsystems can store a certain amount before filling their capacity. There may be even minimum storage limits in cases of emergencies. Finally there is a cost function associated with this system: Pumps use electricity to circulate water, reverse osmosis requires huge amounts of electricity to convert sea water to drinking water, everything in the system has to be periodically maintained and damaged parts need to be replaced. Such a system is the perfect candidate for optimal control.

2.2 Problem Formulation

We can represent a static optimization problem on a network system composed of M subsystems as,

$$f(x) = \sum_{i=1}^{M} f_i(x_i), \tag{2.1}$$

$$g_i(x_i) \leq 0^{m_i},\ i = 1, \ldots, M, \tag{2.2}$$

$$\sum_{i=1}^{M} F_i x_i + Gd = 0^p, \tag{2.3}$$

$$\begin{aligned}
&x = [x_1^T, \ldots, x_M^T]^T,\\
&x_i \in \mathbb{R}^{n_i},\\
&d = [d_1, \ldots, d_r]^T,\\
&d_j \in \mathbb{R},\\
&f : \mathbb{R}^{\sum n_i} \to \mathbb{R},\\
&f_i : \mathbb{R}^{n_i} \to \mathbb{R},\\
&g_i : \mathbb{R}^{n_i} \to \mathbb{R}^{m_i},\\
&F_i \in \{0,1\}^{p \times n_i},\\
&G \in \{0,1\}^{p \times r},
\end{aligned}$$

where (2.1) shows the cost function that we are trying to minimize, (2.2) shows the inequality constraints each subsystem has, (2.3) shows the equality constraints that connect the subsystems, d_j is the demand we are trying to satisfy, and 0^p denotes the zero column vector with p elements.

In addition, for constructing F_i and G matrices, that define the equality constraints, we consider here the two connection types shown in Fig. 2.1 because any network can be represented by some combinations of these two cases:

Fig. 2.1(a) This means there is an equality relationship that can be shown as $x_{i,a} + x_{j,b} = 0$, where $x_{i,a}$ is the ath element of x_i. Thus both F_i and F_j have an element of 1 on the same row of their ath and bth columns, respectively, but G doesn't have an element of 1 on the same row.

Fig. 2.1(b) This means there is an equality relationship that can be shown as $x_{i,a} + x_{j,b} + d_c = 0$, where d_c is the cth element of d. Thus F_i and F_j have an element

of 1 on the same row of their ath and bth columns, respectively, and G has an element of 1 on the same row of the kth column.

For any and every network, all elements of F_i and G are given by 0 or 1. The circles and square represent the subsystems and the demander, respectively.

Lemma 2.1. *F_i and G matrices have the following properties where $\|...\|_p$ is the p-norm, $\mathcal{N}_i$ represents the set of neighbors for subsystem i and $0^{a\times b}$ represents the a-by-b zero matrix.*

(i) $\|F_i\|_1 = 1$ *and* $\|F_i\|_\infty = 1$.

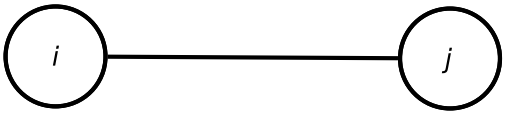
(ii) F_i^T *is the Moore-Penrose pseudoinverse of* F_i *for all* $i = 1, \ldots, M$.

(iii) $F_i^T F_j = 0^{n_i \times n_j}$ *if* $j \notin \mathcal{N}_i$.

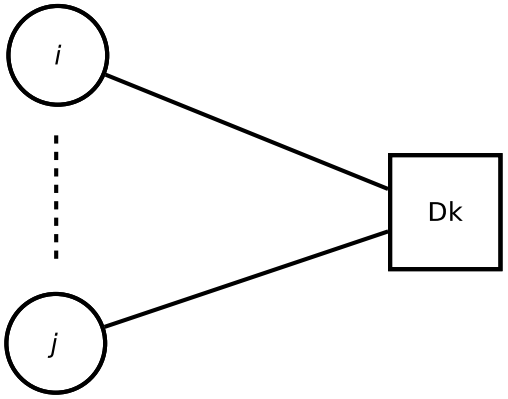
(iv) $\|G\|_1 = 1$ *and* $\|G\|_\infty = 1$.

(v) G is full rank and semi-orthogonal, i.e., a non-square matrix where the columns are orthonormal vectors.

Proof. (i) means that any row or column of F_i has a maximum of one element of 1. This is related to the construction method of F_i. (ii) means $F_i F_i^T$ maps all column vectors of F_i to themselves. This also means $F_i^T F_i$ maps all column vectors of F_i^T to themselves. This is also related to the construction method of F_i. (iii) follows from how F_i matrices



(a) Subsystem to subsystem connection.



(b) Subsystems to user connection.

FIGURE 2.1: Connection types that define (2.3).

are constructed. If $j \notin \mathcal{N}_i$, that means that either F_i or F_j or none has an element of 1 on each row. This means when we multiply F_i^T with F_j, elements of 1 do not line up and the result is a matrix with all 0 elements. (iv) means that any row or column of G_i has a maximum of one element of 1. This is related to the construction method of G_i. (v) means that $rank(G_i) = r$ and $G_i^T G_i = I_r$ as the number of demand signals are less than the number of inequality constraints, $r \leq p$. This is also related to the construction method of G_i. □

We then consider the following problem:

Problem 1. *Given d, find x minimizing* $f(x)$ *while satisfying* (2.2) *and* (2.3).

2.3 Example

To better represent the problem, we can show a simple example for the system in Fig. 2.2. Assume that subsystem 1 is a source, which supplies commodity such as a water well. It can be expressed by:

$$\begin{aligned} x_1(k+1) &= A_1 x_1(k) + B_1 u_1(k), \\ 0 &\leq x_{11}(k) + x_{12}(k) \leq 800, \end{aligned}$$

where $x_1(k) = [x_{11}(k)\ x_{12}(k)]^T$ is the state vector denoting the supplied water and $u_1(k)$ is the control variables such as the speed of the pump and valve settings.

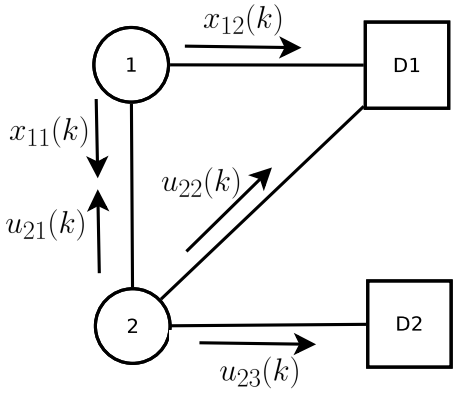


FIGURE 2.2: A simple example of three nodes.

In a similar way, suppose that subsystem 2 is a storage subsystem, which stores the commodity such as a water tower. It can be expressed by:

$$\begin{aligned} x_2(k+1) &= A_2 x_2(k) + B_2 \begin{bmatrix} u_{21}(k) \\ u_{22}(k) \\ u_{23}(k) \end{bmatrix}, \\ -1000 \leq u_{21}(k) + u_{22}(k) + u_{23}(k) &\leq 1000, \\ 100 \leq x_2(k) &\leq 5000, \end{aligned}$$

where $x_2(k)$ represents the stored commodity and $u_2(k) = [u_{21}(k)\ u_{22}(k)\ u_{23}(k)]^T$ is the control variable such as the water discharge rate at a certain pipe.

Finally, suppose that nodes D1 and D2 represent the demanders and contain the demands $d_1(k)$ and $d_2(k)$ respectively. In addition, we have the following equality constraints:

$$\begin{aligned} x_{11}(k) + u_{21}(k) &= 0, \\ x_{12}(k) + u_{22}(k) + d_1(k) &= 0, \\ u_{23}(k) + d_2(k) &= 0. \end{aligned}$$

We can rewrite this in the matrix form as:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0^{|u_1|} \\ 0 & 1 & 0^{|u_1|} \\ 0 & 0 & 0^{|u_1|} \end{bmatrix}}_{F_1} \begin{bmatrix} x_1(k) \\ u_1(k) \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{F_2} \begin{bmatrix} x_2(k) \\ u_2(k) \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{G} d(k) = 0.$$

Chapter 3

Distributed Optimization

3.1 Distributed Optimization Based on Dual Decomposition

In this section, we focus on a dual decomposition based distributed optimization method [26] that will form the basis of our other optimization methods. We begin by writing the Lagrangian of the cost function by merging the cost function $f(x)$ in (2.1) and the equality constraints in (2.3):

$$L = \sum_{i=1}^{M} f_i(x_i) + \lambda^T \left(\sum_{i=1}^{M} F_i x_i + Gd \right) \tag{3.1}$$

where the vector $\lambda \in R^p$ denotes the Lagrange variables (also called dual variables or "price").

Then we change the order of summation of (3.1) to get:

$$L = \sum_{i=1}^{M} \left(f_i(x_i) + \lambda_i^T x_i \right) + \lambda^T G d \tag{3.2}$$

where $\lambda_i = F_i^T \lambda \in R^{n_i}$ is the Lagrange variable for subsystem i.

From this, we can get the part inside the parentheses and use it as the local cost function as it contains only variables for subsystem i. So if we define the local cost function as

$$L_i = f_i(x_i) + \lambda_i^T x_i, \tag{3.3}$$

we get

$$L = \sum_{i=1}^{M} L_i + \lambda^T G d \tag{3.4}$$

for the global cost function.

Then instead of Problem 1, we consider the following local optimization problem for given λ_i:

Problem 2. *Given λ_i, find x_i minimizing L_i in* (3.3) *while satisfying* (2.2).

If the solution x_i^* to Problem 2 which minimizes the local cost functions L_i in (3.3) while satisfying the inequality constraint (2.2) for given λ_i, $i = 0, \ldots, M$, also satisfies the equality constraint (2.3), then it is also a solution to Problem 1 that minimizes the original cost function (2.1) while satisfying constraints (2.2) and (2.3).

Now, λ_i should also be optimized in order for the solutions x_i^* to satisfy the equality constraint (2.3). As the result of each optimization depends on the result of the other optimization, this requires using an iterative method for x_i and λ_i. As we start using an iterative method, x_i and λ_i depend on the iteration $\tau \in \{1, \ldots\}$. So the local cost function (3.3) becomes

$$L_i(\tau) = f_i(x_i(\tau)) + \lambda_i^T(\tau) x_i(\tau). \tag{3.5}$$

Because of this iteration process, the equality constraint in (2.3) will not be satisfied at every iteration. We can define an error related to the equality constraint in (2.3) as

$$\eta(\tau) := \sum_{i=1}^{M} F_i x_i(\tau) + Gd. \tag{3.6}$$

Next for given $x_i(\tau)$, we have to update $\lambda_i(\tau)$, for each subsystem at each iteration $\tau \in \{1, \ldots\}$. For this, we start with using a subgradient update rule for the global Lagrange variable $\lambda(\tau)$ as

$$\lambda(\tau + 1) = \lambda(\tau) + \alpha\eta(\tau), \tag{3.7}$$

where α expresses the step size for our subgradient update rule. There are many different methods for defining the step size, some depend on the iteration, τ. But for simplicity, we are using a fixed step size. Also there are different methods for updating $\lambda(\tau)$ but we use subgradient method as we base our approach on the dual decomposition method presented in [26].

By modifying (3.6), we can define a localized version of the equality constraint error in such a way that subsystem i only has to use x_i and x_j, where $j \in \mathcal{N}_i$, for calculating (3.7). We know that $F_i^T F_j = 0^{n_i \times n_j}$ when $j \notin \mathcal{N}_i$ from Lemma 2.1(iii). So now we can define the localized equality constraint error as

$$\eta_i(\tau) := F_i^T \left(F_i x_i(\tau) + \sum_{j \in \mathcal{N}_i} F_j x_j(\tau) + Gd \right), \tag{3.8}$$

and if we multiply both sides of (3.7) with F_i^T we find the localized gradient descent update rule for the Lagrange variable $\lambda_i(\tau)$:

$$\lambda_i(\tau + 1) = \lambda_i(\tau) + \alpha \eta_i(\tau). \tag{3.9}$$

As this is an iterative process, we have to define a stopping criterion for the iteration to stop so that we guarantee to get a result in finite time. Some algorithms define this as a certain threshold for the iteration, τ, but using the equality constraint error is better as the gradient descent is an asymptotically decreasing function and a limit on the error can be used for tuning between speed and accuracy of the optimization algorithm. So as a stopping criterion, we consider whether the global equality constraint in (2.3) is satisfied with a minimal relaxation parameter, ϵ. Now the convergence criterion can be defined as

$$\|\eta(\tau_{end})\|_\infty < \epsilon, \tag{3.10}$$

where $\|...\|_\infty$ is the infinity norm of a vector. Also, τ_{end} denotes the final iteration when the convergence criterion has been satisfied.

We can modify (3.10) and write a convergence criterion for each subsystem i as:

$$\|\eta_i(\tau_{end})\|_\infty < \epsilon. \tag{3.11}$$

Then this convergence criterion can be used by subsystem i to decide whether it converged to a valid solution or not. Then it checks whether the stopping criterion is satisfied and if so it stops the optimization algorithm and prepares for the next time step, k.

Lemma 3.1. (3.10) *is satisfied iff* (3.11) *is satisfied for all* $i = 1, \ldots, M$.

Proof. First we prove that (3.10) is satisfied if (3.11) is satisfied for all $i = 1, \dots, M$. Let's define $F = diag(F_1, \dots, F_M)$ and $\hat{\eta}(\tau) = [\eta_1^T(\tau), \dots, \eta_M^T(\tau)]^T$. Due to the definition of $\eta_i(\tau)$, we can say that $F\hat{\eta}(\tau) = FF^T\tilde{\eta}(\tau)$, where $\tilde{\eta}(\tau) = 1^M \otimes \eta(\tau)$ and 1^M is the all-ones column vector with M elements. We have $rank(FF^T) \geq p$ as $rank(FF^T) = rank(F)$ and $rank(F) = \sum_{i=1}^{M} rank(F_i)$. This means that each element of $\eta(\tau)$ will appear at least once in $F\hat{\eta}(\tau)$ and in $FF^T\tilde{\eta}(\tau)$. If $\|\eta_i(\tau)\|_\infty < \epsilon$ for all $i = 1, \dots, M$, then $\|\hat{\eta}(\tau)\|_\infty < \epsilon$. Due to the properties of F_i in Lemma 2.1(i), we can say that $\|F\hat{\eta}(\tau)\|_\infty = \|F\|_\infty \|\hat{\eta}(\tau)\|_\infty < \epsilon$ is also true. Using the equalities above, we can rewrite it as $\left\|FF^T\tilde{\eta}(\tau)\right\|_\infty < \epsilon$. As we know each row of $\eta(\tau)$ will appear at least once in $FF^T\tilde{\eta}(\tau)$ and using $\left\|FF^T\tilde{\eta}(\tau)\right\|_\infty = \|F\|_\infty \left\|F^T\right\|_\infty \|\tilde{\eta}(\tau)\|_\infty$, we can say that $\|\tilde{\eta}(\tau)\|_\infty < \epsilon$ is satisfied, so $\|\eta(\tau)\|_\infty < \epsilon$ is satisfied.

It is trivial to prove that (3.11) is satisfied for all $i = 1, \dots, M$ if (3.10) is satisfied. □

We summarize this method with the following algorithm:

Algorithm 1 (Distributed Algorithm)**.**

[Updating the solution candidates]

Each subsystem i, $i = 0, \dots, M$, performs the following steps with the initial condition $\tau \leftarrow 1$, $\lambda_i(1) \leftarrow 0^{n_i}$.

U1. $x_i(\tau) \leftarrow min_{x_i} L_i(\tau)$ *given* $\lambda_i(\tau)$.

U2. Send $x_i(\tau)$ *to subsystem* j $\forall j \in \mathcal{N}_i$, *receive* $x_j(\tau)$ *from all subsystems* $j \in \mathcal{N}_i$.

U3. $\eta_i(\tau) \leftarrow F_i^T \left(F_i x_i(\tau) + \sum_{j \in \mathcal{N}_i} F_j x_j(\tau) + Gd \right)$.

U4. $\lambda_i(\tau + 1) \leftarrow \lambda_i(\tau) + \alpha\eta_i(\tau)$.

[Check if the algorithm is terminated]

According to Algorithm 4 or 5.

3.2 Event-Triggered Distributed Optimization

With normal distributed optimization, there are a few shortcomings. One of them is that at each iteration all subsystems exchange $x_i(\tau)$. This increases the communication cost dramatically and if the subsystems are using radio communications, keeping the

communication at a minimum is essential for reducing power requirements. Also this algorithm is synchronous, some subsystems have to wait for more slower nodes which leads to wasted processing power. In addition to that, synchronization can be hard to implement in some networks. We can solve the first problem by using state value estimates and then updating these estimates when required. This way we can decrease the number of communication events. We presented this approach, event-triggered distributed optimization, in [31].

For this approach to work, each neighboring subsystem $j \in \mathcal{N}_i$, requires an estimate of the state of subsystem i. We can define this as:

$$\begin{aligned} &\hat{x}_i(0) = 0^{n_i}, \\ &\hat{x}_i(\tau) = \begin{cases} x_i(\tau) & \text{if updated,} \\ \hat{x}_i(\tau - 1) & \text{if not updated} \end{cases} \end{aligned} \tag{3.12}$$

where $\hat{x}_i(\tau)$ is the estimate for subsystem i during iteration τ, time step k.

This information must be common to subsystems i and $j \in \mathcal{N}_i$ as subsystems $j \in \mathcal{N}_i$ will use this estimate in their localized equality constraint error calculations and subsystem i will use it to calculate the deviation from the actual value $x_i(\tau)$ in order to update it when required. To update it we can define an error related to the state estimate and update it when a threshold value is reached. We represent the estimation error threshold function for subsystem i as $\delta_i(\tau)$. Whenever the estimation error is greater than the value of $\delta_i(\tau)$ at iteration τ, subsystem i will send a message to all $j \in \mathcal{N}_i$ at iteration τ to update $\hat{x}_i(\tau)$ so that

$$\|x_i(\tau) - \hat{x}_i(\tau)\|_\infty \leq \delta_i(\tau) \tag{3.13}$$

is satisfied at each iteration τ. We can guarantee this if the following condition is used to check if the estimate is within limits:

$$\|x_i(\tau) - \hat{x}_i(\tau - 1)\|_\infty \leq \delta_i(\tau). \tag{3.14}$$

If this condition is satisfied, we don't have to update, which means $\hat{x}_i(\tau) = \hat{x}_i(\tau - 1)$ so (3.14) becomes the same as (3.13) which means our condition is met. If, however, (3.14) is not satisfied, $\hat{x}_i(\tau) = x_i(\tau)$ according to the update rules. If we plug this into (3.13), we get $\|0^{n_i}\|_\infty \leq \delta_i(\tau)$ which is always true.

Now the only requirement for the estimation error threshold function is that it should be a positive and decreasing function. The simplest candidate is in the form of:

$$\delta_i(\tau) = \frac{K}{\tau} \tag{3.15}$$

where K is a constant.

Let's consider the optimization procedure at subsystem i. If we use the estimates $\hat{x}_j(\tau), j \in \mathcal{N}_i$ for the local equality constraint in (3.8) we get:

$$\hat{\eta}_i(\tau) := F_i^T \left(F_i x_i(\tau) + \sum_{j \in \mathcal{N}_i} F_j \hat{x}_j(\tau) + Gd \right), \tag{3.16}$$

where $\hat{\eta}_i(\tau)$ is called local equality constrain error estimate for subsystem i during iteration τ, time step k.

Also instead of the actual error, we now use this error estimate in the new convergence criterion for each subsystem i:

$$\|\hat{\eta}_i(\tau_{end})\|_\infty < \epsilon. \tag{3.17}$$

Then this convergence criterion can be used by subsystem i to decide whether it converged to a valid solution or not. Then it checks whether the stopping criterion is satisfied and if so it stops the optimization algorithm and prepares for the next time step, k.

However, there will be differences between the Lagrange variable vectors of the neighboring subsystems as we are using the estimate, $\hat{x}_j(\tau)$, instead of the actual value, $x_j(\tau)$, which will lead to big calculation errors as we iterate. To solve this problem, we can use an averaging operation:

$$\lambda_i(\tau) \leftarrow F_i^T W \left(F_i \lambda_i(\tau) + \sum_{j \in \mathcal{N}_i} F_j \lambda_j(\tau) \right), \tag{3.18}$$

where W is a unique diagonal weight matrix, satisfying the following property:

$$W \sum_{i=1}^{M} F_i 1^{n_i} = 1^p. \tag{3.19}$$

where 1^p denotes the all ones column vector with p elements.

After subsystem i updates its Lagrange variable vector, $\lambda_i(\tau)$, its neighboring subsystems $j \in \mathcal{N}_i$ have to update the corresponding values in their Lagrange variable vectors, $\lambda_j(\tau)$, so that the values are synchronized. This is done by the following replacement operation:

$$\lambda_j(\tau) \leftarrow F_j^T F_i \lambda_i(\tau) + \lambda_j(\tau) - F_j^T F_i F_i^T F_j \lambda_j(\tau). \tag{3.20}$$

As this replacement operation seems complex, it is best to show an example to better explain how this operation works.

3.2.1 Example

As an example, assume that we have a system with the following F_i matrices:

$$\begin{aligned} F_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad F_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ F_3 &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}, \qquad F_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \end{aligned} \tag{3.21}$$

Let's use the following λ_i vectors:

$$\begin{aligned} \lambda_1(\tau) &= \begin{bmatrix} \lambda_{1,1} \\ \lambda_{1,2} \end{bmatrix}, \quad \lambda_2(\tau) = \begin{bmatrix} \lambda_{2,1} \\ \lambda_{2,2} \\ \lambda_{2,3} \end{bmatrix}, \\ \lambda_3(\tau) &= \begin{bmatrix} \lambda_{3,1} \\ \lambda_{3,2} \end{bmatrix}, \quad \lambda_4(\tau) = \begin{bmatrix} \lambda_{4,1} \end{bmatrix}. \end{aligned} \tag{3.22}$$

By using (3.19), we can find a unique W matrix for this system:

$$W = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix}. \tag{3.23}$$

Now assume subsystem 2 had an update event. This results in an averaging operation presented in (3.18). At the end of this operation λ_2 becomes

$$\lambda_2(\tau) = \begin{bmatrix} \frac{\lambda_{1,1}+\lambda_{2,1}}{2} \\ \frac{\lambda_{1,2}+\lambda_{2,2}}{2} \\ \frac{\lambda_{2,3}+\lambda_{3,1}+\lambda_{4,1}}{3} \end{bmatrix}. \tag{3.24}$$

Then, as $\mathcal{N}_2 = \{1, 3, 4\}$, subsystems 1, 3 and 4 execute the replacement operation shown in (3.20). At the end of this operation we get

$$\lambda_1(\tau) = \begin{bmatrix} \frac{\lambda_{1,1}+\lambda_{2,1}}{2} \\ \frac{\lambda_{1,2}+\lambda_{2,2}}{2} \end{bmatrix}, \tag{3.25}$$

$$\lambda_3(\tau) = \begin{bmatrix} \frac{\lambda_{2,3}+\lambda_{3,1}+\lambda_{4,1}}{3} \\ \lambda_{3,2} \end{bmatrix}, \tag{3.26}$$

$$\lambda_4(\tau) = \begin{bmatrix} \frac{\lambda_{2,3}+\lambda_{3,1}+\lambda_{4,1}}{3} \end{bmatrix}. \tag{3.27}$$

We can summarize this method with the following algorithm:

Algorithm 2 (Event-Triggered Distributed Algorithm)**.**

[Updating the solution candidates]

Each subsystem i, $i = 0, \ldots, M$, performs the following steps with the initial condition $\tau \leftarrow 1$, $\lambda_i(1) \leftarrow 0^{n_i}$, $\hat{x}_i(0) \leftarrow 0^{n_i}$, $\hat{x}_j(0) \leftarrow 0^{n_j}$ $\forall j \in \mathcal{N}_i$.

U1. $x_i(\tau) \leftarrow min_{x_i} L_i(\tau)$ given $\lambda_i(\tau)$.

U2. Check if $\|x_i(\tau) - \hat{x}_i(\tau - 1)\|_\infty \leq \delta_i(\tau)$:

- *(a) If true: $\hat{x}_i(\tau) = \hat{x}_i(\tau - 1)$.*
- *(b) Else: $\hat{x}_i(\tau) = x_i(\tau)$. Send $x_i(\tau)$ to subsystem j $\forall j \in \mathcal{N}_i$, and get $\lambda_j(\tau)$ from subsystem j $\forall j \in \mathcal{N}_i$. Calculate new $\lambda_i(\tau)$ using $\lambda_i(\tau) \leftarrow F_i^T W \left(F_i \lambda_i(\tau) + \sum_{j \in \mathcal{N}_i} F_j \lambda_j(\tau) \right)$. Send $\lambda_i(\tau)$ to subsystem j $\forall j \in \mathcal{N}_i$,*

U3. Check if new $x_j(\tau)$ was received from subsystem j $j \in \mathcal{N}_i$:

- *(a) If new $x_j(\tau)$ was received: $\hat{x}_j(\tau) = x_j(\tau)$. Send $\lambda_i(\tau)$ and receive $\lambda_j(\tau)$. Calculate new $\lambda_i(\tau)$ using $\lambda_i(\tau) \leftarrow F_i^T F_j \lambda_j(\tau) + \lambda_i(\tau) - F_i^T F_j F_j^T F_i \lambda_i(\tau)$.*
- *(b) Else: $\hat{x}_j(\tau) = \hat{x}_j(\tau - 1)$.*

U4. $\hat{\eta}_i(\tau) \leftarrow F_i^T \left(F_i x_i(\tau) + \sum_{j \in \mathcal{N}_i} F_j \hat{x}_j(\tau) + Gd \right)$.

U5. $\lambda_i(\tau + 1) \leftarrow \lambda_i(\tau) + \alpha \hat{\eta}_i(\tau)$.

[Check if the algorithm is terminated]

According to Algorithm 4 or 5.

This algorithm works as long as the estimation error threshold function diminishes to 0.. We can prove this with the following theorem.

Theorem 3.2. *If* $\delta_i(\tau)$, $i \in \{1, \ldots, M\}$, *satisfies* $\lim_{\tau\to\infty} \delta_i(\tau) = 0$ *and* $\delta_i(\tau+1) < \delta_i(\tau)$ *for all* $\tau \geq 1$. *Suppose that the current time is* k. *Then the above Event-Triggered Distributed Algorithm stops at a finite iteration* τ.

Proof. As the update rule is $\|x_i(\tau) - \hat{x}_i(\tau-1)\| \leq \delta_i(\tau)$, after the update we have the inequality $\|x_i(\tau) - \hat{x}_i(\tau)\| \leq \delta_i(\tau)$. As $\|x_i(\tau) - \hat{x}_i(\tau)\| \leq \delta_i(\tau)$ we can say that $\hat{x}_i(\tau) = x_i(\tau) + O_i(\delta_i(\tau))$ where $O_i(\delta_i(\tau))$ is an error vector that shows the difference between the estimate and the real values. It has a lower bound of $-\delta_i(\tau)$ and an upper bound of $\delta_i(\tau)$. Then we can calculate the equality condition by

$$\hat{\eta}_i(\tau) = \bar{F}_i^T \bar{F}_i x_i(\tau) + \sum_{j\in\mathcal{N}_i} \bar{F}_i^T \bar{F}_j x_j(\tau)$$
$$+ \sum_{j\in\mathcal{N}_i} \bar{F}_i^T \bar{F}_j O_j(\delta_i(\tau)) + \sum_{j=1}^{r} \bar{F}_i^T H_j d_j.$$

Our stopping criteria is $\|\hat{\eta}_i(\tau)\| < \epsilon$, $k = k_0, \ldots, k_0 + N - 1$. We can easily see that $\hat{\eta}_i(\tau) = \eta_i(\tau) + \sum_{j\in\mathcal{N}_i} \bar{F}_i^T \bar{F}_j O_j(\delta_i(\tau))$ By using the triangle inequality, we can rewrite it as

$$\|\hat{\eta}_i(\tau)\| \leq \|\eta_i(\tau)\| + \left\| \sum_{j\in\mathcal{N}_i} \bar{F}_i^T \bar{F}_j O_j(\delta_i(\tau)) \right\|. \tag{3.28}$$

The boundaries of $O_i(\delta_i(\tau))$ can be used to find the boundaries of $\sum_{j\in\mathcal{N}_i} \bar{F}_i^T \bar{F}_j O_j(\delta_i(\tau))$

$$\left\| \sum_{j\in\mathcal{N}_i} \bar{F}_i^T \bar{F}_j O_j(\delta_i(\tau)) \right\| < |\mathcal{N}_i|\, \delta_i(\tau),$$

where $|\mathcal{N}_i|$ is the number of neighbors node i has. Then we can rewrite (3.28) as

$$\|\hat{\eta}_i(\tau)\| < \|\eta_i(\tau)\| + |\mathcal{N}_i|\, \delta_i(\tau). \tag{3.29}$$

For any $\epsilon > 0$, there exists a finite τ' such that

$$|\mathcal{N}_i|\, \delta_i(\tau) < \frac{\epsilon}{2} \qquad \forall \tau > \tau' \tag{3.30}$$

Also for any $\epsilon > 0$, there exists a finite τ'' such that

$$\|\eta_i(\tau)\| < \frac{\epsilon}{2} \qquad \forall \tau > \tau'' \tag{3.31}$$

Eqs. (3.30) and (3.31) imply that Eq. (3.29) holds true for all $\tau > max(\tau', \tau'')$. □

Chapter 4

Asynchronous Optimization

With event-triggered communication, we were able to solve the communication cost problem. However, it was still a synchronous method, leading to wasted processing power and requirement for hard to implement synchronization. We can get remove the synchronization requirement and design an asynchronous optimization method to solve these problems.

For this approach to work, each neighboring subsystem $j \in \mathcal{N}_i$, requires an estimate of the state of subsystem i. We can define this as:

$$\begin{aligned} &\hat{x}_i(0) = 0^{n_i}, \\ &\hat{x}_i(\tau_i) = \begin{cases} x_i(\tau_i) & \text{if updated,} \\ \hat{x}_i(\tau_i - 1) & \text{if not updated} \end{cases} \end{aligned} \tag{4.1}$$

where $\hat{x}_i(\tau_i)$ is the estimate for subsystem i during iteration τ_i. The important difference here is that the network wide iteration number τ is replaced with local iteration number τ_i.

Similar to event-triggered communication, we define an error related to the state estimate and update it when a threshold value is reached. This threshold function must use the local iteration number, so the estimation error threshold function for subsystem i is represented as $\delta_i(\tau_i)$. Whenever the estimation error is greater than the value of $\delta_i(\tau_i)$ at iteration τ_i, subsystem i will send a message to all $j \in \mathcal{N}_i$ at iteration τ_i so that

$$\|x_i(\tau_i) - \hat{x}_i(\tau_i)\|_\infty \leq \delta_i(\tau_i) \tag{4.2}$$

is satisfied instead of 3.13. To satisfy this condition we modify and get

$$\|x_i(\tau_i) - \hat{x}_i(\tau_i - 1)\|_\infty \leq \delta_i(\tau_i). \tag{4.3}$$

Again, the only requirement for the estimation error threshold function is that it should be a positive and decreasing function. The simplest candidate is similar to 3.15:

$$\delta_i(\tau_i) = \frac{K}{\tau_i} \tag{4.4}$$

where K is a constant.

As subsystem i also has the estimates of each neighboring subsystem $j \in \mathcal{N}_i$, we define the initialization and update rules as:

$$\begin{aligned} &\hat{x}_j(0) = 0^{n_j}, \\ &\hat{x}_j(\tau_i) = \begin{cases} x_j(\tau_j) & \text{if updated,} \\ \hat{x}_j(\tau_i - 1) & \text{if not updated} \end{cases} \end{aligned} \tag{4.5}$$

The important part here is that the estimate is represented as $\hat{x}_j(\tau_i)$, which means it now depends on the local iteration number of subsystem i, τ_i. But when we are updating it, we get the actual value from subsystem j at τ_j.

Now we consider the procedure at subsystem i. If we use the estimates $\hat{x}_j(\tau_i), j \in \mathcal{N}_i$ for the local equality constraint in (3.8) we get:

$$\hat{\eta}_i(\tau_i) := F_i^T \left(F_i x_i(\tau_i) + \sum_{j \in \mathcal{N}_i} F_j \hat{x}_j(\tau_i) + Gd \right), \tag{4.6}$$

where $\hat{\eta}_i(\tau_i)$ is called local equality constrain error estimate for subsystem i during iteration τ_i.

Also, similar to 3.17, we use this error estimate in the new convergence criterion for each subsystem i:

$$\|\hat{\eta}_i(\tau_{i,end})\|_\infty < \epsilon \tag{4.7}$$

where $\tau_{i,end}$ denotes the final iteration number of subsystem i when the convergence criterion has been satisfied.

Then this convergence criterion can be used by subsystem i to decide whether it converged to a valid solution or not. Then it checks whether the stopping criterion is satisfied and if so it stops the optimization algorithm.

Again, there will be differences between the Lagrange variable vectors of the neighboring subsystems as we are using the estimate, $\hat{x}_j(\tau_i)$, instead of the actual value, $x_j(\tau_j)$, which will lead to bigger calculation errors as we iterate as now the iteration numbers are different. So, we modify the averaging operation in 3.18 and get:

$$\lambda_i(\tau_i) \leftarrow F_i^T W \left(F_i \lambda_i(\tau_i) + \sum_{j \in \mathcal{N}_i} F_j \lambda_j(\tau_j) \right). \tag{4.8}$$

One thigh to consider here is that we are using $\lambda_j(\tau_j)$ as we require the Lagrange variable vector used by subsystem j at that iteration.

Again, after this update operation, the neighboring subsystems $j \in \mathcal{N}_i$ have to update the corresponding values in their Lagrange variable vectors, $\lambda_j(\tau_j)$, so that the values are synchronized. So, we modify the replacement operation in 3.20 and get:

$$\lambda_j(\tau_j) \leftarrow F_j^T F_i \lambda_i(\tau_i) + \lambda_j(\tau_j) - F_j^T F_i F_i^T F_j \lambda_j(\tau_j). \tag{4.9}$$

Another important thing to consider is Algorithm 2, subsystem i waits indefinitely at step U2.b until it gets $\lambda_j(\tau)$ from subsystem j $\forall j \in \mathcal{N}_i$. But as the iterations are not synchronized, a neighboring subsystem $j \in \mathcal{N}_i$ may have already stopped the iteration algorithm if subsystem i send a message stating that it converged to a solution at the previous iteration. This means that subsystem i will enter an infinite loop and never reach a solution. To solve this problem we have to check if the stopping criterion is satisfied while we are waiting for $\lambda_j(\tau)$ and terminate the optimization if it is satisfied. That means some parts of the stopping criterion algorithm is working in parallel to our optimization algorithm. We will talk more about these in the next chapter.

Now, we can summarize this method with the following algorithm:

Algorithm 3 (Asynchronous Algorithm)**.**

[Updating the solution candidates]

Each subsystem i, $i = 0, \ldots, M$, performs the following steps with the initial condition $\tau_i \leftarrow 1$, $\lambda_i(1) \leftarrow 0^{n_i}$, $\hat{x}_i(0) \leftarrow 0^{n_i}$, $\hat{x}_j(0) \leftarrow 0^{n_j}$ $\forall j \in \mathcal{N}_i$.

U1. $x_i(\tau_i) \leftarrow min_{x_i} L_i(\tau_i)$ given $\lambda_i(\tau_i)$.

U2. Check if $\|x_i(\tau_i) - \hat{x}_i(\tau_i - 1)\|_\infty \leq \delta_i(\tau_i)$:

(a) If true: $\hat{x}_i(\tau_i) = \hat{x}_i(\tau_i - 1)$.

(b) *Else:* $\hat{x}_i(\tau_i) = x_i(\tau_i)$. *Send* $x_i(\tau_i)$ *to subsystem* j $\forall j \in \mathcal{N}_i$, *and wait for* $\lambda_j(\tau_j)$ *from subsystem* j $\forall j \in \mathcal{N}_i$. *Check if the stopping criterion is satisfied according to Algorithm 4 or 5, if so, go to their last step. If not, calculate new* $\lambda_i(\tau_i)$ *using*
$\lambda_i(\tau_i) \leftarrow F_i^T W \left(F_i \lambda_i(\tau_i) + \sum_{j \in \mathcal{N}_i} F_j \lambda_j(\tau_j) \right)$. *Send* $\lambda_i(\tau_i)$ *to subsystem* j $\forall j \in \mathcal{N}_i$,

U3. Check if new $x_j(\tau_j)$ *was received from subsystem* j $j \in \mathcal{N}_i$*:*

(a) *If new* $x_j(\tau_j)$ *was received:* $\hat{x}_j(\tau_i) = x_j(\tau_j)$. *Send* $\lambda_i(\tau_i)$ *and receive* $\lambda_j(\tau_j)$. *Calculate new* $\lambda_i(\tau_i)$ *using*
$\lambda_i(\tau_i) \leftarrow F_i^T F_j \lambda_j(\tau_j) + \lambda_i(\tau_i) - F_i^T F_j F_j^T F_i \lambda_i(\tau_i)$.

(b) *Else:* $\hat{x}_j(\tau_i) = \hat{x}_j(\tau_i - 1)$.

U4. $\hat{\eta}_i(\tau_i) \leftarrow F_i^T \left(F_i x_i(\tau_i) + \sum_{j \in \mathcal{N}_i} F_j \hat{x}_j(\tau_i) + Gd \right)$.

U5. $\lambda_i(\tau_i + 1) \leftarrow \lambda_i(\tau_i) + \alpha \hat{\eta}_i(\tau_i)$.

[Check if the algorithm is terminated]

According to Algorithm 4 or 5 where τ *is replaced by* τ_i.

Chapter 5

Stopping Criteria

5.1 Supervisor Based Stopping Criterion

Supervisor based stopping criterion is the default stopping criterion that's used by nearly all distributed methods. The reason for that is its simplicity as single master–multiple slave systems are trivial to develop at this point. When the convergence criterion, (3.11) or (3.17), is satisfied for subsystem i, it sends a message to a supervisor stating that it converged to a solution. The supervisor keeps track of the whole system and decides when to stop according to the convergence messages it received. We can summarize this method with the following stopping algorithm:

Algorithm 4 (Supervisor Stopping Algorithm)**.**

[Checking if the update is terminated]

C1. Each i checks if $\|\eta_i(\tau)\|_\infty < \epsilon$ and sends True *or* False *to the supervisor.*

C2. Supervisor checks if all received messages are True or not:

- *(a) If all* True*: Send the* Stop *signal to all subsystems.*
- *(b) Else: Send the* Continue *signal to all subsystems.*

C3. Each i checks the message sent by the supervisor:

- *(a) If* Stop*: $\tau_{end} = \tau$ and $x_i(\tau_{end})$ for all $i = 0, \ldots, M$ are solutions that minimize the global cost function* (2.1)*.*
- *(b) If* Continue*: $\tau \leftarrow \tau + 1$, and go to step U1.*

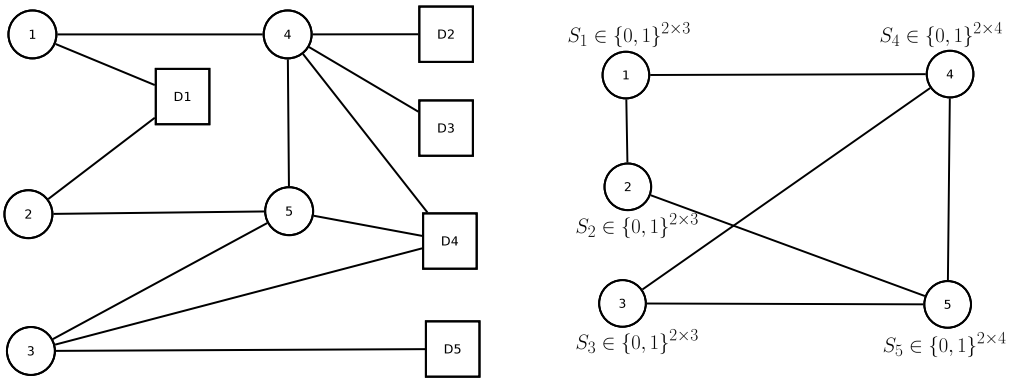
If we are using the asynchronous optimization method, there are a few minor changes. With asynchronous optimization, the master runs a separate thread where it checks if C2. is satisfied. Also subsystem i executes step C3. during U2.b to check if the stopping criterion is satisfied so that it does not get stuck in an infinite loop.

5.2 Diffusion Based Stopping Criterion

5.2.1 Communication Graph and Stopping Criterion Matrix

We mentioned that, the supervisor based stopping criterion is used by nearly all distributed methods but it has several shortcomings: It requires global information in the form of convergence criterion, causes an increase in system development costs as it requires communication lines between the supervisor and all subsystems, and affects the robustness of the whole system as the weakest link in the chain, if something happens to this supervisor, the optimization algorithm will not work for the whole system. Because of this, we developed a new stopping criterion for distributed optimization named diffusion based stopping criterion. This name was chosen as the convergence status of a subsystem "diffuses" through the whole network and can be found in each subsystem's stopping criterion matrix not by itself but mixed with other subsystems' convergence statuses.

For this method to work, we are assuming that convergence status of a subsystem is sent to only immediate neighbors. This information is considered to be local as a subsystem i has only the convergence status information about its immediate neighbors $j \in \mathcal{N}_i$ besides its own.



(a) G (b) G^*

FIGURE 5.1: An example graph G and its communication graph G^*

For this criterion, we create a new graph called communication graph G^* from the system representation G. If both subsystems i and j are connected by an edge in graph G, similar to Fig. 2.1(a), they are also connected in the communication graph G^*, which is fairly straightforward. But, if both subsystems i and j are connected to a demander by an edge in graph G, they are also connected in the communication graph G^*, similar to Fig. 2.1(b). Otherwise, they are not connected. This is because they all have and element of 1 on the same row of F_i and F_j which means that there is an equality constraint between their state values. An example of the communication graph G^* can be seen in Fig. 5.1. We can create this communication graph Fig. 5.1(b) from the system graph Fig. 5.1(a) as subsystems 1 and 4 are both connected by an edge so they are neighbors, and similarly for the subsystem pairs 2–5, 3–4 and 4–5. Also subsystems 1 and 2 are both connected through node U1 so they are neighbors, and similarly for the case of subsystems 3 and 4. The resulting neighbor sets are:

$$\begin{aligned}
\mathcal{N}_1 &= \{2, 4\}, \\
\mathcal{N}_2 &= \{1, 5\}, \\
\mathcal{N}_3 &= \{4, 5\}, \\
\mathcal{N}_4 &= \{1, 3, 5\}, \\
\mathcal{N}_5 &= \{2, 3, 4\},
\end{aligned}$$

which satisfy Lemma 2.1(iii),

For this method, each subsystem has a stopping criterion matrix, denoted $S_i \in \{0,1\}^{\varnothing G^* \times (|\mathcal{N}_i|+1)}$ for subsystem i, where $\varnothing G^*$ is the diameter of the communication graph G^*, defined as the greatest shortest distance between any two subsystems, and $|\mathcal{N}_i|$ is the number of neighbors of subsystem i which cannot be more than M. This matrix, S_i, stores information about the convergence status of the subsystem i and its neighboring subsystems $j \in \mathcal{N}_i$.

Let $S_i(a, b)$ denote the (a, b)th element of S_i and $[j]_{\mathcal{N}_i}$ denotes the index of subsystem $j \in \mathcal{N}_i$. This way the element $S_i(1, [j]_{\mathcal{N}_i})$, $j \in \mathcal{N}_i$ represents the convergence status of the subsystem j and the element $S_i(1, |\mathcal{N}_i| + 1)$ represents the convergence status of the subsystem i itself. The elements of the next rows of S_i represent the status of the previous rows of S_i and S_j $\forall j \in \mathcal{N}_i$. The element $S_i(a + 1, [j]_{\mathcal{N}_i})$, $j \in \mathcal{N}_i$ represents the status of the ath row of S_j and the element $S_i(a + 1, |\mathcal{N}_i| + 1)$ represents the status of the ath row of S_i. Only the elements $S_i(a, |\mathcal{N}_i| + 1)$, $a \in \{1, \ldots, \varnothing G^*\}$, are sent to the neighboring subsystems. If $S_i(1, |\mathcal{N}_i| + 1)$ is 1, it means the convergence criterion, (3.11) or (3.17), was satisfied for subsystem i. For $a \in \{2, \ldots, \varnothing G^*\}$, $S_i(a, |\mathcal{N}_i| + 1)$ is

calculated by

$$S_i(a, |\mathcal{N}_i| + 1) = \prod_{b=1}^{|\mathcal{N}_i|+1} S_i(a-1, b) \tag{5.1}$$

By checking the last row of S_i, we can find the convergence status of all subsystems, e.g., if some S_i has elements of all 1 on its last row, it means that all subsystems of network have converged to a solution. One can argue that this matrix contains global information as it has information about the convergence status of all subsystems. But only neighboring subsystems' convergence status is represented clearly on the first row, other rows contain the merged information ab out the other nodes and there is no way of finding out the convergence status of a particular subsystem if that subsystem is not a neighbor. So we say that the matrix S_i has only local information.

For Fig. 5.1(b), the stopping criterion matrices are:

$$\begin{aligned}
S_1 &= \begin{bmatrix} S_2(1,3) & S_4(1,4) & S_1(1,3) \\ S_2(2,3) & S_4(2,4) & S_1(2,3) \end{bmatrix}, \\
S_2 &= \begin{bmatrix} S_1(1,3) & S_5(1,4) & S_2(1,3) \\ S_1(2,3) & S_5(2,4) & S_2(2,3) \end{bmatrix}, \\
S_3 &= \begin{bmatrix} S_4(1,4) & S_5(1,4) & S_3(1,3) \\ S_4(2,4) & S_5(2,4) & S_3(2,3) \end{bmatrix}, \\
S_4 &= \begin{bmatrix} S_1(1,3) & S_3(1,3) & S_5(1,4) & S_4(1,4) \\ S_1(2,3) & S_3(2,3) & S_5(2,4) & S_4(2,4) \end{bmatrix}, \\
S_5 &= \begin{bmatrix} S_2(1,3) & S_3(1,3) & S_4(1,4) & S_5(1,4) \\ S_2(2,3) & S_3(2,3) & S_4(2,4) & S_5(2,4) \end{bmatrix},
\end{aligned} \tag{5.2}$$

where $S_1(1,3)$, $S_2(1,3)$, $S_3(1,3)$, $S_4(1,3)$ and $S_5(1,4)$ are 1 if the convergence criterion in (3.11), (3.17) or (4.7) is satisfied and 0 otherwise. Also

$$\begin{aligned}
S_1(2,3) &= S_2(1,3)S_4(1,4)S_1(1,3), \\
S_2(2,3) &= S_1(1,3)S_5(1,4)S_2(1,3), \\
S_3(2,3) &= S_4(1,4)S_5(1,4)S_3(1,3), \\
S_4(2,4) &= S_1(1,3)S_3(1,3)S_5(1,4)S_4(1,4), \\
S_5(2,4) &= S_2(1,3)S_3(1,3)S_4(1,4)S_5(1,4).
\end{aligned} \tag{5.3}$$

5.2.2 Algorithm

For writing the algorithms, we need the following definitions:

Definition 5.1. Define the following messages from subsystem i with the stopping criteria matrix S_i:

1. **convergence event message**: an arbitrary message stating that convergence criterion in (3.11) or (3.17) is satisfied for subsystem i. It sets $S_i(1, |\mathcal{N}_i| + 1)$.
2. **convergence event break message**: an arbitrary message stating that the convergence criterion in (3.11) or (3.17) is not satisfied for subsystem i. It resets $S_i(1, |\mathcal{N}_i| + 1)$.
3. a**th row event message**: an arbitrary message stating that the ath row contains all elements of 1s. It sets $S_i(a + 1, |\mathcal{N}_i| + 1)$.
4. a**th row event break message**: an arbitrary message stating that the ath row does not contain all elements of 1s. It resets $S_i(a + 1, |\mathcal{N}_i| + 1)$.

We can write down the algorithm for distributed optimization with the diffusion based stopping criterion for subsystem i with the stopping criterion matrix S_i as:

Algorithm 5 (Distributed Algorithm with Diffusion Based Stopping Criterion)**.**

[Checking if the update is terminated]

Each subsystem i, $i = 0, \ldots, M$, performs the following steps with the initial condition $S_i \leftarrow 0^{\varnothing G^* \times (|\mathcal{N}_i|+1)}$:

C1. If $\|\eta_i(\tau)\|_\infty < \epsilon$:

$S_i(1, |\mathcal{N}_i| + 1) \leftarrow 1$, *send* ***convergence event message*** *to all subsystems $j \in \mathcal{N}_i$.*

Else:

$S_i(1, |\mathcal{N}_i| + 1) \leftarrow 0$, *send* ***convergence event break message*** *to all subsystems $j \in \mathcal{N}_i$, $\tau \leftarrow \tau + 1$, and go to step U1.*

C2. If subsystem i receives ***convergence event message*** *from subsystem $j \in \mathcal{N}_i$:*

$S_i(1, [j]_{\mathcal{N}_i}) \leftarrow 1$.

Else if subsystem i receives ***convergence event break message*** *from subsystem $j \in \mathcal{N}_i$:*

$S_i(1, [j]_{\mathcal{N}_i}) \leftarrow 0$, $\tau \leftarrow \tau + 1$, *and go to step U1.*

C3. For $a \in \{1, \ldots, \varnothing G^* - 1\}$:

(a) If $\prod_{b=1}^{|\mathcal{N}_i|+1} S_i(a, b) = 1$*:*

$S_i(a+1, |\mathcal{N}_i| + 1) \leftarrow 1$ *and send a* ***th row event message*** *to all subsystems $j \in \mathcal{N}_i$.*

Else:

$S_i(a + 1, |\mathcal{N}_i| + 1) \leftarrow 0$ *and send a* ***th row event break message*** *to all subsystems $j \in \mathcal{N}_i$, $\tau \leftarrow \tau + 1$, and go to step U1.*

(b) If subsystem i receives a ***th row event message*** *from a subsystem $j \in \mathcal{N}_i$:*

$S_i(a+1, [j]_{\mathcal{N}_i}) \leftarrow 1$.

Else if receives a ***th row event break message*** *from a subsystem $j \in \mathcal{N}_i$:*

$S_i(a+1, [j]_{\mathcal{N}_i}) \leftarrow 0$, $\tau \leftarrow \tau + 1$, *and go to step U1.*

C4. If $\prod_{b=1}^{|\mathcal{N}_i|+1} S_i(\varnothing G^*, b) = 1$*:*

$\tau_{end} = \tau$ *and* $x_i(\tau_{end})$ *are solutions that minimize the global cost function.*

Else:

$\tau \leftarrow \tau + 1$, *and go to step U1.*

If we are using the asynchronous optimization method, again there are a few minor changes. With asynchronous optimization, all subsystems run a separate thread where they execute steps C2. and C3.b. Also subsystem i executes step C4. during U2.b to check if the stopping criterion is satisfied so that it does not get stuck in an infinite loop.

To guarantee that our algorithms stop if all subsystems have converged to a solution, we need the following theorem.

Theorem 5.2. *In the above algorithms, an arbitrarily chosen* $S_i \prod_{b=1}^{|\mathcal{N}_i|+1} S_i(\varnothing G^*, b) = 1$ *if and only if the algorithm of all subsystems have converged to a solution.*

Proof. Suppose that there is some i such that $\prod_{b=1}^{|\mathcal{N}_i|+1} S_i(\varnothing G^*, b) = 1$ and there is subsystem j in which the solution of the corresponding algorithm did not converge at iteration τ. This means that all subsystems $k \in N_j$ have $S_k(1, [j]_{\mathcal{N}_i}) = 0$. This, in turn, makes $S_k(2, |\mathcal{N}_k| + 1) = 0$ for all subsystems $k \in N_j$. This effect cascades in the whole graph and makes at least one of the elements of the $\varnothing G^*$th row of S_i 0. This contradicts with the assumption of $\prod_{b=1}^{|\mathcal{N}_i|+1} S_i(\varnothing G^*, b) = 1$. The converse is obvious. This completes the proof. □

As we already showed the convergence of the optimization methods, this theorem is enough to show that our algorithms using the diffusion based stopping criterion converge in finite iteration.

5.2.3 Example

As an example, consider the graph G^* in Fig. 5.1, where the resulting $\varnothing G^*$ is 2. That means we have $S_1 = 0^{2\times 3}$ for subsystem 1, $S_2 = 0^{2\times 3}$ for subsystem 2, $S_3 = 0^{2\times 3}$ for

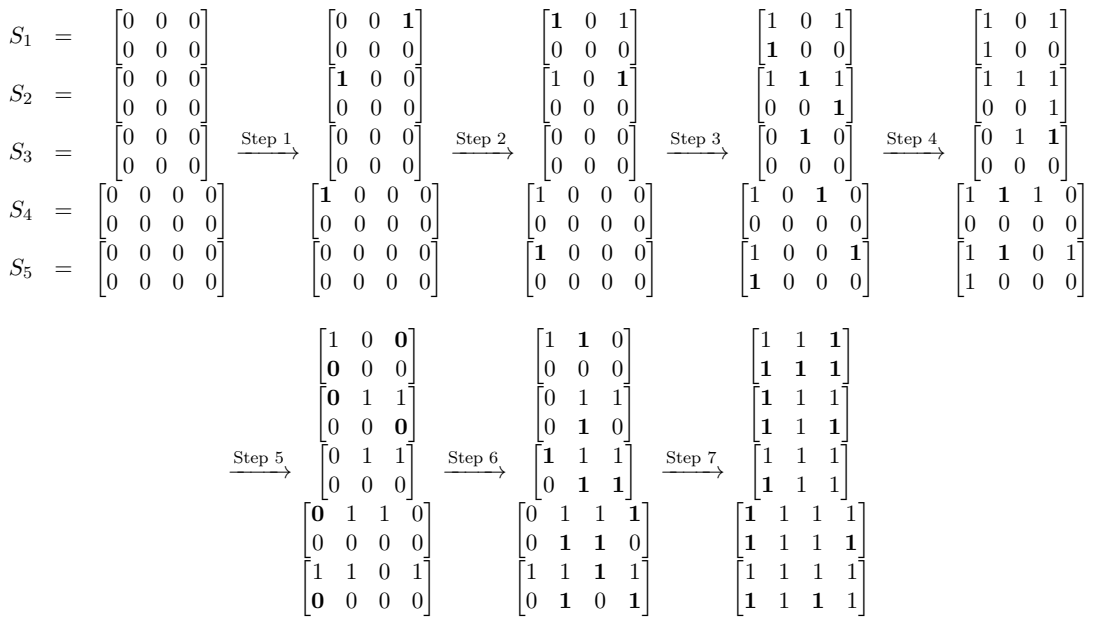


FIGURE 5.2: Example of behavior of stopping criterion matrices of subsystems.

subsystem 3, $S_4 = 0^{2\times 4}$ for subsystem 4 and $S_5 = 0^{2\times 4}$ for subsystem 5 initially. Let's follow stopping criterion matrices of all subsystems, as shown in Fig. 5.2. The stopping criterion matrices have the elements represented in (5.2).

Initially suppose that S_i for all subsystems i have all zero. Then let us explain how stopping criterion matrices behave under the following fictional story:

Step 1 (Subsystem 1 converges) $S_1(1,3)$ becomes 1. Subsystem 1 sends out *convergence event message*, which causes $S_2(1,1)$ and $S_4(1,1)$ to become 1.

Step 2 (Subsystem 2 converges) $S_2(1,3)$ becomes 1. Subsystem 2 sends out *convergence event message*, which causes $S_1(1,1)$ and $S_5(1,1)$ to become 1.

Step 3 (Subsystem 5 converges) $S_5(1,4)$ becomes 1. Subsystem 5 sends out *convergence event message*, which causes $S_2(1,2)$, $S_3(1,2)$ and $S_4(1,3)$ to become 1. At this time, $S_2(2,3)$ becomes 1 as all elements in the first row of S_2 are 1, it sends out *1st row event message*, which causes $S_1(2,1)$ and $S_5(2,1)$ to become 1.

Step 4 (Subsystem 3 converges) $S_3(1,3)$ becomes 1. Subsystem 3 sends out *convergence event message*, which causes $S_4(1,2)$ and $S_5(1,2)$ to become 1.

Step 5 (Subsystem 1 stops satisfying (3.11)) $S_1(1,3)$ becomes 0. Subsystem 1 sends out *convergence event break message*, which causes $S_2(1,1)$ and $S_4(1,1)$ to become 0. At this time $S_2(2,3)$ becomes 0 as not all elements in the first row of S_2 are 1, it sends out *1st row event break message*, which causes $S_1(2,1)$ and $S_5(2,1)$ to become 0.

Step 6 (Subsystem 4 converges) $S_4(1,4)$ becomes 1. Subsystem 4 sends out *convergence event message*, which causes $S_1(1,2)$, $S_3(1,1)$ and $S_5(1,3)$ to become 1. At this time, $S_3(2,3)$ becomes 1 as all elements in the first row of S_3 are 1, it sends out *1st row event message*, which causes $S_4(2,2)$ and $S_5(2,2)$ to become 1. Also, $S_5(2,4)$ becomes 1 as all elements in the first row of S_5 are 1, it sends out *1st row event message*, which causes $S_2(2,2)$, $S_3(2,2)$ and $S_4(2,3)$ to become 1.

Step 7 (Subsystem 1 converges) $S_1(1,3)$ becomes 1. Subsystem 1 sends out *convergence event message*, which causes $S_2(1,1)$ and $S_4(1,1)$ to become 1. At this time, $S_1(2,3)$ becomes 1 as all elements in the first row of S_1 is 1, it sends out *1st row event message*, which causes $S_2(2,1)$ and $S_4(2,1)$ to become 1. Also, $S_2(2,3)$ becomes 1 as all elements in the first row of S_2 is 1, it sends out *1st row event message*, which causes $S_1(2,1)$ and $S_5(2,1)$ to become 1. Finally, $S_4(2,4)$ becomes 1 as all elements in the first row of S_4 are 1, it sends out *1st row event message*, which causes $S_1(2,2)$, $S_3(2,1)$ and $S_5(2,3)$ to become 1.

Step 8 Now $\prod_{b=1}^{|\mathcal{N}_i|+1} S_i(2,b) = 1$ for all subsystems i and the controllers will stop optimizing as the whole network G has converged at iteration $\tau_{end} = \tau$.

Chapter 6

Numerical Simulations

6.1 Problem Settings of Numerical Simulation

We apply the proposed algorithm to the network system in Fig. 6.1. In these simulations, instead of solving for a single discrete time, we solve the problem for a sequence of discrete time steps, k, by using Model Predictive Control (MPC).

In MPC we have the following equations:

$$x_i(k+1) = A_i x_i(k) + B_i u_i(k), \tag{6.1}$$

$$C_i x_i(k) + D_i u_i(k) \leq E_i, \tag{6.2}$$

$$\sum_{i=1}^{M} \left(\hat{F}_i x_i(k) + \tilde{F}_i u_i(k) \right) + Gd(k) = 0, \tag{6.3}$$

$$J_{i,k_0} = \sum_{k=k_0}^{k_0+N-1} x_i^T(k) Q_i x_i(k) + u_i^T(k) R_i u_i(k), \tag{6.4}$$

where (6.1) represents the system model, i.e., $x_i(k+1) = x_i(k) + u_i(k)$; (6.2) represents the system constraints, i.e., $x_i(k) - u_i(k) \leq 100$, similar to (2.2); (6.3) represents the equality constraints, similar to (2.3) and (6.4) represents the cost function, similar to $f_i(x_i)$ in (2.1). Q_i is a positive semi-definite diagonal cost matrix, R_i is a positive definite diagonal cost matrix, N is the prediction horizon and k is the discrete time. Values of Q_i and R_i are selected at random for the simulations. This problem is reduced into Problem 1. We give more details about how these equations are modified to be solved by CPLEX in Appendix B.

We will run six algorithms: the distributed algorithm with supervisor (denoted as Distributed in figures), the event-triggered distributed algorithm with supervisor (denoted as Event in figures), the asynchronous algorithm with supervisor (denoted as Asynch.

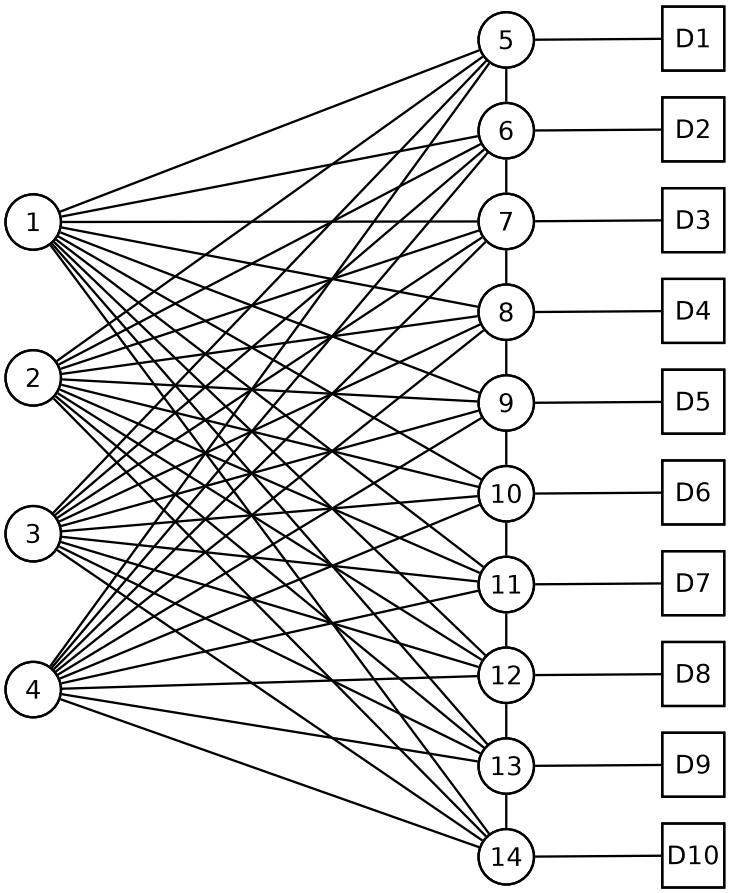


FIGURE 6.1: Physical system used for simulation

in figures), the distributed algorithm with diffusion based stopping criterion (denoted as Distributed D. in figures), the event-triggered distributed algorithm with diffusion based stopping criterion (denoted as Event D. in figures), and the asynchronous algorithm with diffusion based stopping criterion (denoted as Asynch. D. in figures).

Suppose that subsystems $1-4$ are source subsystems that supply the commodity; subsystems $5-14$ are storage subsystems that store the commodity and D1−D10 are consumers that denote the exogenous signal. The demand profiles can be seen in Fig. 6.2, where $d_6 - d_{10}$ have the exact same profile with $d_1 - d_5$. We consider Problem 1 with randomly created cost and inequality constraint functions. To use MPC, we set the prediction horizon, N, to 10 and $x(0) = 0$ for the initial time step, $k = 0$. For the design parameters of the algorithm, we set $\alpha = 0.05$, $\epsilon = 3$ and $K = 18$.

For distributed and event-triggered algorithms, our simulation code is written as a sequential process, ie. for each subsystem we execute one step of the algorithm, switch to the next subsystem and move onto the next step when all subsystems have completed the same step. This sequential method ensures that only one CPLEX process is running

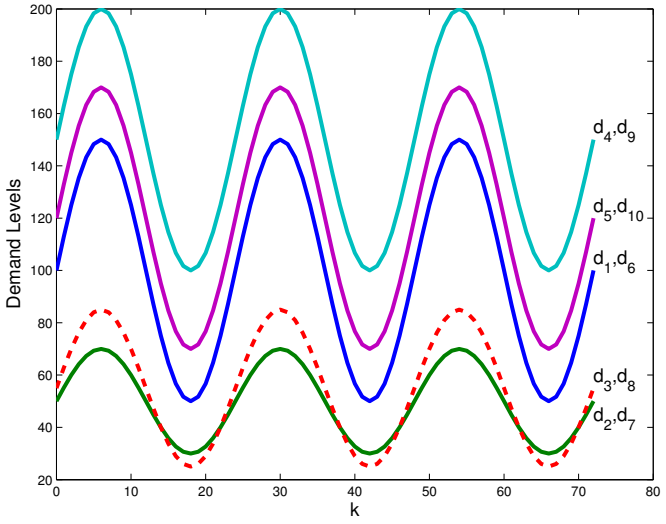


FIGURE 6.2: Demand profiles used during numerical simulation.

at any given time and insures that the steps are synchronized. Also we don't have to develop a method for passing messages between subsystems. For asynchronous algorithm executions, we have individual optimization programs for each subsystem and message passing between them is achieved by memory maps to files. This ensures that reliable message passing between subsystems is achieved but it forces multiple threads to be run on the same processor, which may affect our calculation time measurements.

We will not be comparing these algorithms with the centralized method as this has been widely investigated. However, we can quickly compare the base method, the distributed algorithm with supervisor, with the centralized method in order to see it's advantages. Optimal cost value and calculation time at each time step in this case are shown in Figs. 6.3 and 6.4, respectively. As one can see in Fig. 6.3, there is a little difference between the centralized and distributed methods so we can say that the path taken by the distributed method is different from the centralized one, which is expected. Also the distributed method's cost is usually greater than the centralized method as it cannot reach the optimum solution due to the relaxation parameter, ϵ. However, an interesting aspect is that from time to time the distributed method's cost is less than the centralized case. We believe the distributed method stores more commodity at certain time steps and when it starts using this extra stored commodity, the source subsystems can provide less commodity to satisfy the demand which lowers the cost.

For Fig. 6.4, the calculation time for the centralized method is more or less the same during each time step. However, for a small networked system, we would expect the calculation time of the distributed method to be greater than the calculation time of the centralized method. This is true for the initial time step, , $k = 0$, and most of

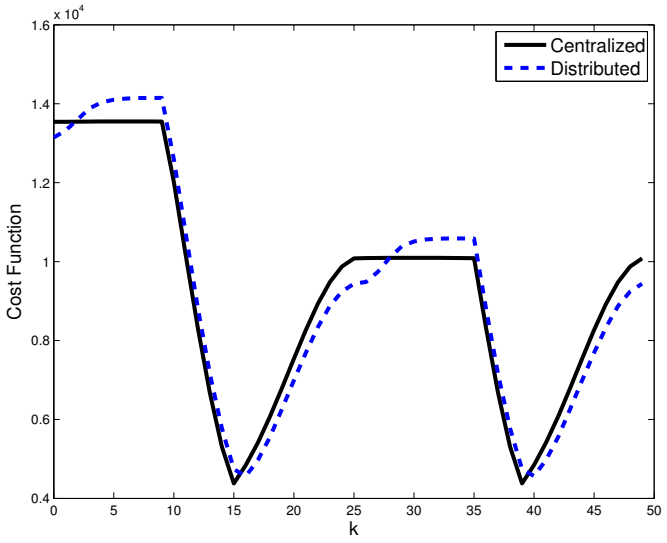


FIGURE 6.3: Cost function comparison of centralized and distributed algorithms.

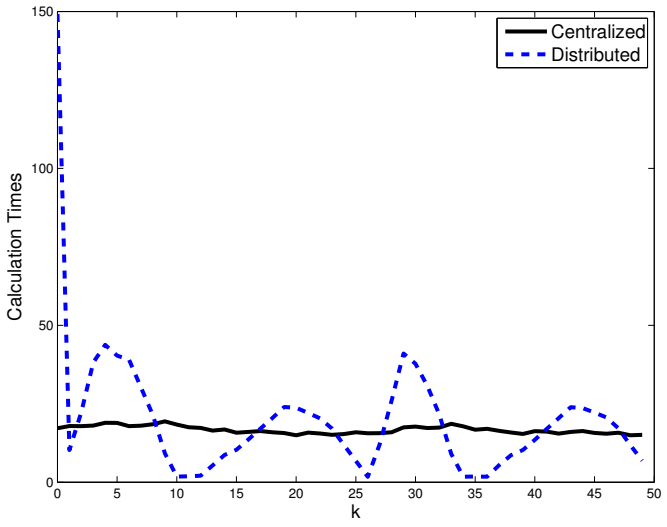


FIGURE 6.4: Calculation time comparison of centralized and distributed algorithms.

the simulation. However, there are certain intervals where the calculation time of the distributed method is less than the calculation time of the centralized method. We believe this is due to how MPC works: After the optimal solution is found, the solution for that time step is applied as the control output, then the result vector is shifted one time step in order to provide the initial values for the next time step. We use this for our Lagrange variable vectors, $\lambda_i(\tau)$, so at the end of each time step, k, we don't have to initialize them again. This saves some valuable time as at $k = 0$, the calculation time is around 150 ms. for the distributed method but during other time steps, the calculation time has a maximum value of about 45 ms.

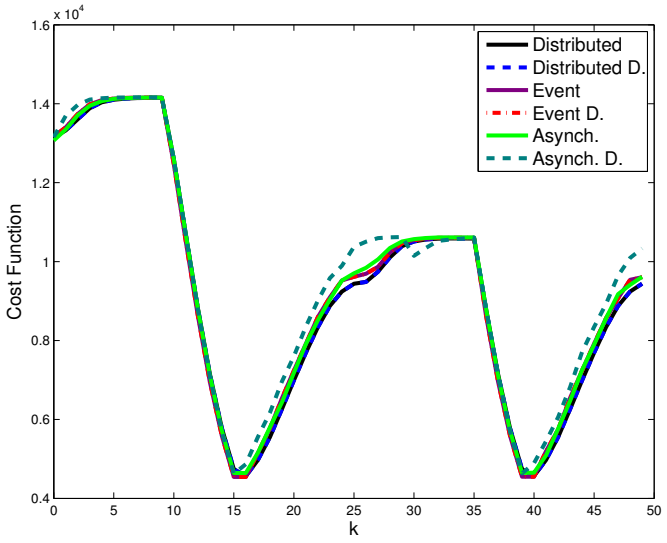


FIGURE 6.5: Cost function comparison of algorithms.

6.2 Comparison Between Algorithms

The resulting optimal cost value and calculation time at each time step in this case are shown in Figs. 6.5 and 6.6, respectively. As one can see in Fig. 6.5, there is little difference between all six algorithms so we can say that the chosen algorithm has minimal effect on the optimality of the solution, although the asynchronous algorithm with diffusion based stopping criterion has a higher cost at certain time steps.

For Fig. 6.6, the calculation time shown for the algorithms is the time at which each stopping criterion determines the optimization algorithm has converged without the communication delays. One can see that the use of diffusion based stopping criterion has no effect on the calculation time as expected for distributed and event-triggered communication algorithms. For asynchronous algorithms, some difference is expected as each run of the simulation is different due to asynchronous execution of optimization algorithm. We can also see that the calculation time for asynchronous algorithms is about three times higher than other algorithms.

Figs. 6.7 and 6.8 show the number of iterations and number of communication events, respectively. Fig. 6.7 has the same shape as the calculation time graph in Fig. 6.6 for distributed and event-triggered communication algorithms so we can say that calculation time mainly depends on the number of iterations τ. For the asynchronous algorithms, we can see that the calculation time at each iteration is higher than other algorithms. However, in Fig. 6.8 we see that the event-triggered communication and asynchronous algorithms have far less communication events. Also again the use of diffusion based

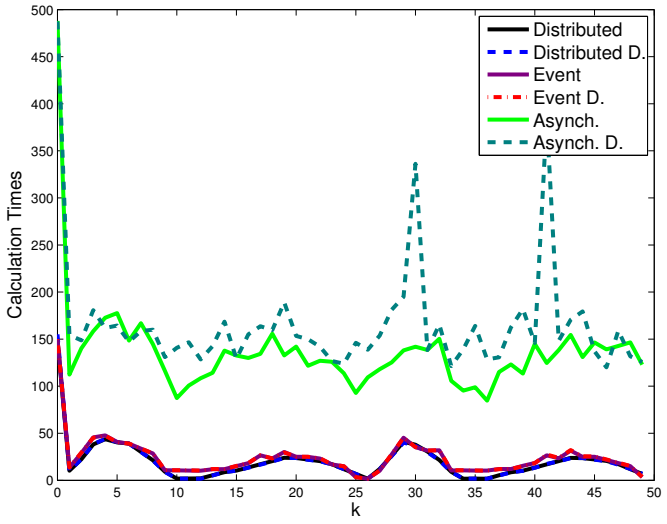


FIGURE 6.6: Calculation time comparison of algorithms.

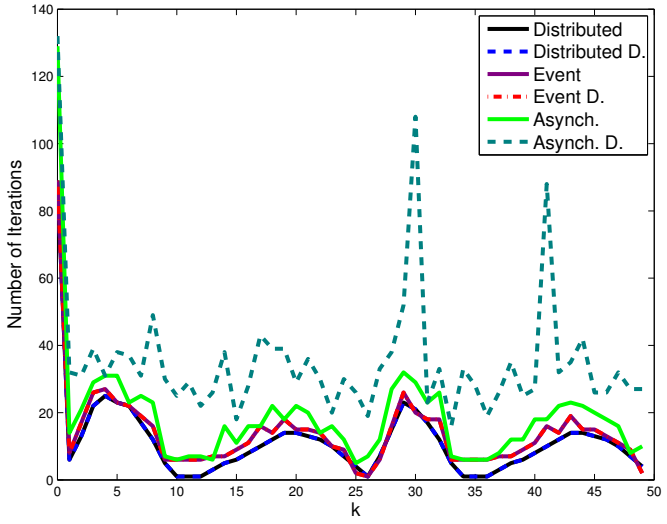


FIGURE 6.7: Number of iterations comparison of algorithms.

stopping criterion has no effect on the number of communication events as expected for distributed and event-triggered communication algorithms.

Table 6.1 shows the sum of maximum calculation times, maximum number of iterations and maximum number of communication events for the whole simulation excluding the time step 0 which is the initialization part where all dual variable vectors are zero initially. This is done in order to better judge the behavior of the algorithms during normal operation. It also shows the average iteration time for the whole simulation to judge how each method affects processor load. The calculation time difference between the supervisor based stopping criterion and the diffusion based stopping criterion are minimal for distributed and event-triggered communication algorithms and we can say

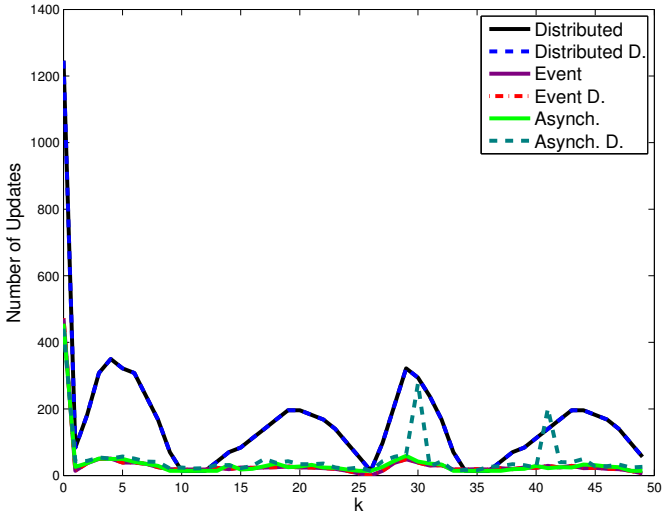


FIGURE 6.8: Number of updates comparison of algorithms.

TABLE 6.1: Numerical comparison of algorithms

Algorithm	Total Time	Max. Iterations	Max. Updates	Iteration Time
Distributed	848.8 ms.	25	350	1.66 ms.
Distributed D.	741.5 ms.	25	350	1.64 ms.
Event-Triggered	1037.6 ms.	27	51	1.63 ms.
Event-Triggered D.	1043.8 ms.	27	51	1.64 ms.
Asynch.	6343.2 ms.	32	59	4.45 ms.
Asynch. D.	7810.8 ms.	108	276	3.16 ms.

that these are caused by small load changes on the simulation machine. Also we can see that the stopping criteria has no effect on the maximum number of iterations and maximum number of updates for the same algorithms; these metrics depend purely on the optimization method used. If we look at average iteration time, we can see that the asynchronous algorithms put a significant load on the processor. We believe this is one of the reasons why the calculation time values were about three times higher than the other algorithms in Fig. 6.6.

To estimate how long the asynchronous algorithms would take if they were running on individual machines, we can scale the number of iterations by using the average iteration time values in Table 6.1. If we multiply the number of iterations with the average iteration time of 1.66 ms., we get the corrected calculation time graph in Fig. 6.9. Also total calculation time becomes 1338 ms. for the asynchronous algorithm with supervisor and 2732.4 ms. for the asynchronous algorithm with diffusion based stopping criterion.

From these result, we can say that two algorithms, the event-triggered distributed algorithm with diffusion based stopping criterion and the asynchronous algorithm with

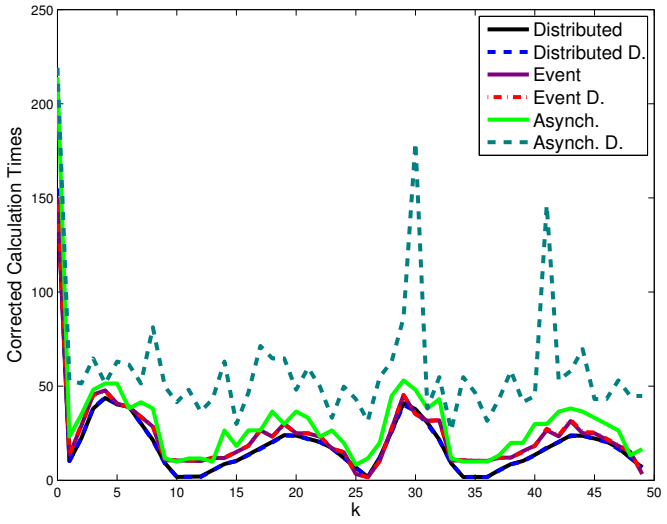


FIGURE 6.9: Corrected calculation time comparison of algorithms.

supervisor, are our candidates. The former has a low calculation time, does not need a supervisor and has a low communication cost due to low number of communication events. However, it is still a synchronous algorithm which means that the steps should be synchronized between subsystems. This can be advantageous as this synchronization is enough to determine when the optimization algorithm for each time step should start. The latter again has the low communication cost, and is an asynchronous method so it eliminates the need for synchronizing each subsystems' clock with each other. However, a supervisor is needed to keep track of the convergence status of each subsystem. Using a supervisor is not a bad thing for an asynchronous method as even when the stopping events are detected without a supervisor, a supervisor or a master node is needed to signal each subsystem to start the optimization algorithm for a certain time step.

Chapter 7

Conclusion

7.1 Summary

In this thesis, two new parallel distributed optimization algorithms and one novel stopping criterion for optimizing large-scale networked systems has been proposed. In chapter 3, we introduced the dual decomposition based optimization the event-triggered communication methods. In chapter 4, we presented the asynchronous optimization method by extending the previous methods. Chapter 5, focused on a new stopping criterion, named diffusion based stopping criterion, that can be used instead of the usual supervisor based stopping criterion.

In chapter 2, single commodity network flow problems have been discussed as the problem we are trying to solve. The properties of subsystem and exogenous signal connections types are shown, these connections are mathematically defined and an example to mathematical equations derived from these connections is given.

In chapter 3, the method that forms the basis of all our distribution optimization methods, the dual decomposition based optimization method, is introduced. A convergence criterion that can be used for detecting when a subsystem has converged to a solution has been defined. The shortcomings of this method is analyzed and the event-triggered communication method is introduced as a solution to some of these problems. Also step by step algorithms for these algorithms are given.

In chapter 4, possible problems related to synchronization requirement, mainly the tough implementation problem, are determined. A new distributed optimization method named asynchronous optimization is proposed as a solution and its algorithm is presented. for this method.

In chapter 5, initially, the supervisor based stopping criterion that is used by almost all distributed optimization methods is given. As this stopping criterion opposes the essence of distribution, a new dubbed diffusion based stopping criterion is nominated, the algorithm is shown and a simple example is provided.

In chapter 6, combination of three distributed optimization methods and two stopping criteria are analyzed with numerical simulations as six different algorithms. Some of the methods used to write the simulation codes are presented. Simulations are conducted for an arbitrary example system certain metrics are analyzed to determine the effectiveness of these algorithms.

In this thesis we tried to develop a parallel distributed optimization method which required less communication between subsystems, is asynchronous and has a decentralized stopping criterion. First requirement was for reducing the communication cost of the whole system, the second one was for easy real life implementation by eliminating the need for a synchronized clock, and the last one was for eliminating a node which reduced the fault tolerance of the whole system. After analyzing the simulation results, we can say that the event-triggered distributed algorithm with diffusion based stopping criterion and the asynchronous algorithm with supervisor are good candidates for using in real systems. The former has a close calculation time to our base algorithm, requires less communication events which lowers the communication cost, and does not need a supervisor node thus being a completely decentralized method. However it has the disadvantage of being a synchronous method, requiring some sort of clock synchronization between each subsystem, which itself is a widely researched topic. But if this synchronization is implemented, the subsystems can use this synchronized clock to start the optimization algorithm for each time step. The former requires nearly the same number of communication events as the event-triggered distributed algorithm and is an asynchronous method which does not require a synchronized clock so it is easier to implement in real life. However, there is a supervisor which has to keep track of each subsystem' convergence status and signal them to stop or continue the optimization algorithm. But this supervisor can have other uses in the system where some events, such as starting the optimization algorithm for a time step, should be done at the same time.

The algorithm which satisfies all the above requirements, the asynchronous algorithm with diffusion based stopping criterion, is not suggested as some subsystems stop unexpectedly while others continue optimization and all the algorithms have to be restarted from the beginning which causes the peaks in the number of iterations and calculation cost graphs. Also real life application of this algorithm is hard as although it does not require synchronized clocks and a supervisor for stopping the optimization algorithms, it still needs a way to start the optimization algorithms at the same time for each time

step. Falling back to either using a synchronized clock or a supervisor eliminates this problem, which results in using one of the two algorithms that we suggest.

7.2 Future Works

Distributed optimization is an attractive topic and distributed MPC is an effective method to solve large-scale optimization problems on networked systems. There are many possible extensions of this research.

First, although this thesis focused on single commodity network flow problems, distributed MPC can be used to solve many other types of distributed optimization problems. We believe with some modifications to the methods presented in this thesis, many such systems can be controlled optimally.

Another is on the simulation aspect. We have simulated only small networks due to time constraints and simulation system limitations. Possibly larger networks and different complex topologies can be simulated to analyze these algorithms. Also all simulations are done on a single simulation machine. New simulations with a cluster of simulation servers can be run for better analysis and determining the effects of communication delays.

Also although we have developed these algorithms, we have no idea about the effect of system size or neighbor numbers to different metrics as iteration time, number of iterations and number of updates. It should be possible to derive relations between system parameters and metrics by theoretical work.

Finally although the algorithms are verified by numerical simulations, application of theoretical work is also important. Case studies where these algorithms are applied to real world systems to optimally control them; such as utility systems, supply-demand chains and even networked robotics systems; can be conducted.

Appendix A

Decomposition Explanation

In this appendix, we will look at two decomposition techniques, primal decomposition and dual decomposition, and explain why dual decomposition is better for single commodity network flow problems. For this we will look at a real simple case, such as two subsystems connected as shown in Fig. 2.1(a) named as subsystem 1 and subsystem 2.

A.1 Primal Decomposition

We consider an optimization problem in the form of

$$\begin{aligned} &\text{minimize } f(x_1, x_2, y) = f_1(x_1, y) + f_2(x_2, y) \\ &\text{subject to } g_1(x_1, y) \leq \mathbf{0},\ g_2(x_2, y) \leq \mathbf{0}. \end{aligned} \tag{A.1}$$

where x_i is the local variables for subsystem i and y is the complicating variables that couple both subsystems. If we fix y, we can divide this problem into individual subproblems in the form of

$$\begin{aligned} &\text{minimize}_{x_1}\ f_1(x_1, y) \\ &\text{subject to } g_1(x_1, y) \leq \mathbf{0}, \end{aligned} \tag{A.2}$$

$$\begin{aligned} &\text{minimize}_{x_2}\ f_2(x_2, y) \\ &\text{subject to } g_2(x_2, y) \leq \mathbf{0}, \end{aligned} \tag{A.3}$$

and the results of this optimization are x_1^* and x_2^* respectively. Now we have another subproblem that has to be solved in the form of

$$\begin{aligned} &\text{minimize}_{y}\ f_1(x_1^*, y) + f_2(x_2^*, y) \\ &\text{subject to } g_1(x_1^*, y) \leq \mathbf{0},\ g_2(x_2^*, y) \leq \mathbf{0}. \end{aligned} \tag{A.4}$$

and the result of this optimization is y^*.

We can solve (A.2) and (A.3) individually and even at the same time, then a master solves (A.4). After this, we plug y^* into (A.2) and (A.3) again to find the new x_1^* and x_2^*. So our algorithm becomes:

Algorithm 6 (Primal Decomposition Algorithm)**.**

Fix y^* to any initial value. Each subsystem i executes the following step:

> *Find x_i^* that minimizes $f_i(x_i, y^*)$ for a given y^*, subject to $g_i(x_i, y^*) \leq \mathbf{0}$. Send this x_i^* to the master.*

The master executes the following step:

> *Find y^* that minimizes $f_1(x_1^*, y) + f_2(x_2^*, y)$ for given x_1^* and x_2^* from the previous step, subject to $g_i(x_i, y) \leq \mathbf{0}$ for all i. Send this y^* to subsystems.*

Repeat

A.2 Dual Decomposition

In primal decomposition, we give the job of calculating the complicating variable to a master which means this method required a master. Or it has to be solved by each subsystem which means we are not using our processing power effectively as finding a solution to an optimization problem uses significant computation power. However, if we were to introduce new variables y_1 and y_2, we can rewrite A.1 as

$$\begin{aligned} &\text{minimize } f(x_1, x_2, y_1, y_2) = f_1(x_1, y_1) + f_2(x_2, y_2) \\ &\text{subject to } g_1(x_1, y_1) \leq \mathbf{0},\ g_2(x_2, y_2) \leq \mathbf{0},\ y_1 + y_2 = \mathbf{0}. \end{aligned} \tag{A.5}$$

by introducing the equality constraint $y_1 + y_2 = \mathbf{0}$ which requires two local versions of the complicating variable. Now the cost equations do not contain any complicating terms. Now we can write the Lagrangian as

$$L(x_1, x_2, y_1, y_2, \lambda) = f_1(x_1, y_1) + f_2(x_2, y_2) + \lambda^T y_1 + \lambda^T y_2, \tag{A.6}$$

where λ^T is the Lagrangian variable. We can divide this problem into individual subproblems in the form of

$$\begin{aligned} &\text{minimize}_{x_1,y_1}\ f_1(x_1,y_1)+\lambda^T y_1 \\ &\text{subject to } g_1(x_1,y_1)\leq \mathbf{0}, \end{aligned} \tag{A.7}$$

$$\begin{aligned} &\text{minimize}_{x_2,y_2}\ f_2(x_2,y_2)+\lambda^T y_2 \\ &\text{subject to } g_2(x_2,y_2)\leq \mathbf{0}, \end{aligned} \tag{A.8}$$

and the results of this optimization are x_1^*, y_1^*, x_2^* and y_2^* respectively. Now we have to update the Lagrange variable by using the subgradient of the dual problem. The dual function can be written as

$$h(\lambda)=h_1(\lambda)+h_2(\lambda), \tag{A.9}$$

where $h_i(\lambda)=\inf\limits_{x_i,y_i}\big(f_i(x_i,y_i)+\lambda^T y_i\big)$. Then the dual problem becomes

$$\text{maximize } h_1(\lambda)+h_2(\lambda). \tag{A.10}$$

which is the main problem we are trying to solve in dual decomposition.

As $-h_i(\lambda)=-\sup\limits_{x_i,y_i}\big(-\lambda^T y_i-f_i(x_i,y_i)\big)$, the subgradient of $-h_i(\lambda)$ at λ for a given x_i^* and y_i^* is $-y_i^*$. This means the subgradient of the negative dual function $-h_1(\lambda)-h_2(\lambda)$ is equal to $-y_1^*-y_2^*$. So we can update λ by using the subgradient method:

$$\lambda=\lambda+\alpha(y_1^*+y_2^*), \tag{A.11}$$

where α is the step size for the subgradient algorithm.

We can solve (A.7) and (A.8) individually and even at the same time, then either a master or all subsystems solve (A.11). It is not a problem for all subsystems to solve (A.11) as it contains two vector summations and one multiplication. After this, we plug the new λ into (A.7) and (A.8) again to find the new x_1^*, y_1^*, x_2^* and y_2^*. So our algorithm becomes:

Algorithm 7 (Dual Decomposition Algorithm)**.**

Fix λ to any initial value. Each subsystem i executes the following steps:

1. *Find x_i^* and y_i^* that minimizes $f_i(x_i,y_i)+\lambda^T y_i$ subject to $g_i(x_i,y_i)\leq \mathbf{0}$. Send y_i^* to other subsystem.*
2. *Find new λ using $\lambda=\lambda+\alpha(y_1^*+y_2^*)$.*

Repeat

Appendix B

MPC Implementation with Dual Decomposition

B.1 Step 1:

Let's start by writing the cost function:

$$L_{i,k_0} = \sum_{k=k_0}^{k_0+N-1} x_i^T(k)Q_i x_i(k) + u_i^T(k)R_i u_i(k) + \lambda_{x,i}^T(k)x_i(k) + \lambda_{u,i}^T(k)u_i(k). \tag{B.1}$$

Then we can make the following definitions:

$$\begin{aligned} X_{i,k_0} &\triangleq \begin{bmatrix} x_i(k_0) \\ \vdots \\ x_i(k_0+N-1) \end{bmatrix}, \qquad U_{i,k_0} \triangleq \begin{bmatrix} u_i(k_0) \\ \vdots \\ u_i(k_0+N-1) \end{bmatrix}, \\ \Lambda_{x,i,k_0} &\triangleq \begin{bmatrix} \lambda_{x,i}(k_0) \\ \vdots \\ \lambda_{x,i}(k_0+N-1) \end{bmatrix}, \quad \Lambda_{u,i,k_0} \triangleq \begin{bmatrix} \lambda_{u,i}(k_0) \\ \vdots \\ \lambda_{u,i}(k_0+N-1) \end{bmatrix}. \end{aligned} \tag{B.2}$$

If we plug these definitions into the (B.1), we get:

$$L_{i,k_0} = X_{i,k_0}^T \begin{bmatrix} Q_i & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q_i \end{bmatrix} X_{i,k_0} + U_{i,k_0}^T \begin{bmatrix} R_i & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R_i \end{bmatrix} U_{i,k_0} + \Lambda_{x,i,k_0}^T X_{i,k_0} + \Lambda_{u,i,k_0}^T U_{i,k_0}. \tag{B.3}$$

We can rewrite it as:

$$L_{i,k_0} = X_{i,k_0}^T \bar{Q}_i X_{i,k_0} + U_{i,k_0}^T \bar{R}_i U_{i,k_0} + \Lambda_{x,i,k_0}^T X_{i,k_0} + \Lambda_{u,i,k_0}^T U_{i,k_0}. \tag{B.4}$$

B.2 Step 2:

Let's open all the individual system models and write them in terms of $x_i(k_0)$ and $u_i(k_0)$:

$$\begin{aligned} x_i(k_0) &= I x_i(k_0) \\ x_i(k_0+1) &= A_i x_i(k_0) + B_i u_i(k_0) \\ x_i(k_0+2) &= A_i x_i(k_0+1) + B_i u_i(k_0+1) \\ &= A_i^2 x_i(k_0) + A_i B_i u_i(k_0) + B_i u_i(k_0+1) \\ &\vdots \end{aligned} \tag{B.5}$$

If we write (B.5) in matrix form, we get:

$$\begin{bmatrix} x_i(k_0) \\ x_i(k_0+1) \\ x_i(k_0+2) \\ \vdots \\ x_i(k_0+N-1) \end{bmatrix} = \begin{bmatrix} I \\ A_i \\ A_i^2 \\ \vdots \\ A_i^{N-1} \end{bmatrix} x_i(k_0) + \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ B_i & 0 & \cdots & 0 & 0 \\ A_i B_i & B_i & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ A_i^{N-2} B_i & A_i^{N-3} B_i & \cdots & B_i & 0 \end{bmatrix} \begin{bmatrix} u_i(k_0) \\ u_i(k_0+1) \\ u_i(k_0+2) \\ \vdots \\ u_i(k_0+N-1) \end{bmatrix}. \tag{B.6}$$

We can rewrite it as:

$$X_{i,k_0} = G_{i,0} x_i(k_0) + G_{i,1} U_{i,k_0}. \tag{B.7}$$

B.3 Step 3:

From (B.4), we have:

$$L_{i,k_0} = X_{i,k_0}^T \bar{Q}_i X_{i,k_0} + U_{i,k_0}^T \bar{R}_i U_{i,k_0} + \Lambda_{x,i,k_0}^T X_{i,k_0} + \Lambda_{u,i,k_0}^T U_{i,k_0}.$$

Also from (B.7), we have:

$$X_{i,k_0} = G_{i,0} x_i(k_0) + G_{i,1} U_{i,k_0}.$$

If we plug (B.7) into (B.4), we get:

$$\begin{aligned} L_{i,k_0} = \quad & (G_{i,0}x_i(k_0) + G_{i,1}U_{i,k_0})^T \bar{Q}_i (G_{i,0}x_i(k_0) + G_{i,1}U_{i,k_0}) + U_{i,k_0}^T \bar{R}_i U_{i,k_0} + \\ & \Lambda_{x,i,k_0}^T (G_{i,0}x_i(k_0) + G_{i,1}U_{i,k_0}) + \Lambda_{u,i,k_0}^T U_{i,k_0}. \end{aligned} \tag{B.8}$$

Let's rearrange the terms:

$$\begin{aligned} L_{i,k_0} = \quad & U_{i,k_0}^T (G_{i,1}^T \bar{Q}_i G_{i,1} + \bar{R}_i) U_{i,k_0} + \\ & 2U_{i,k_0}^T (G_{i,1}^T \bar{Q}_i G_{i,0} x_i(k_0) + G_{i,1}^T \Lambda_{x,i,k_0}/2 + \Lambda_{u,i,k_0}/2) + \\ & x_i^T(k_0)(G_{i,0}^T \bar{Q}_i G_{i,0}) x_i(k_0) + \Lambda_{x,i,k_0}^T G_{i,0} x_i(k_0). \end{aligned} \tag{B.9}$$

We can rewrite it as:

$$\begin{aligned} L_{i,k_0} = \quad & U_{i,k_0}^T M U_{i,k_0} + 2U_{i,k_0}^T (L x_i(k_0) + G_{i,1}^T \Lambda_{x,i,k_0}/2 + \Lambda_{u,i,k_0}/2) + \\ & x_i^T(k_0) N x_i(k_0) + \Lambda_{x,i,k_0}^T G_{i,0} x_i(k_0). \end{aligned} \tag{B.10}$$

This is the QP problem that we are trying to solve.

B.4 Step 4:

We now have to write our inequality constraints for our QP problem. Let's start by writing the inequality constraint for the whole prediction horizon:

$$\begin{aligned} C_i x_i(k_0) + D_i u_i(k_0) &\leq E_i \\ C_i x_i(k_0+1) + D_i u_i(k_0+1) &\leq E_i \\ C_i x_i(k_0+2) + D_i u_i(k_0+2) &\leq E_i \\ \vdots & \\ C_i x_i(k_0+N-1) + D_i u_i(k_0+N-1) &\leq E_i. \end{aligned} \tag{B.11}$$

This can be rewritten by using the definitions in (B.2) as:

$$\underbrace{\begin{bmatrix} C_i & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_i \end{bmatrix}}_{\bar{C}_i} X_{i,k_0} + \underbrace{\begin{bmatrix} D_i & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & D_i \end{bmatrix}}_{\bar{D}_i} U_{i,k_0} \leq \underbrace{\begin{bmatrix} E_i \\ \vdots \\ E_i \end{bmatrix}}_{\bar{E}_i} \tag{B.12}$$

If we plug in (B.7), we get:

$$\bar{C}_i (G_{i,0}x_i(k_0) + G_{i,1}U_{i,k_0}) + \bar{D}_i U_{i,k_0} \leq \bar{E}_i \tag{B.13}$$

Let's rearrange the terms:

$$\left(\bar{C}_i G_{i,1} + \bar{D}_i\right) U_{i,k_0} \leq \bar{E}_i - \bar{C}_i G_{i,0} x_i(k_0) \tag{B.14}$$

We can rewrite it as:

$$S_i U_{i,k_0} \leq \Gamma_i + T_i x_i(k_0) \tag{B.15}$$

This is the inequality constraint for our QP problem.

Bibliography

[1] John N. Tsitsiklis, Dimitri P. Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.

[2] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, 1989.

[3] K. Mani Chandy, Steven H. Low, Ufuk Topcu, and Huan Xu. A simple optimal power flow model with energy storage. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 1051–1057, 2010.

[4] Huan Xu, Ufuk Topcu, Steven H. Low, Christopher R. Clarke, and K. Mani Chandy. Load-shedding probabilities with hybrid renewable power generation and energy storage. In *Proceedings of the 48th Annual Allerton Conference on Communication, Control, and Computing*, pages 233–239, 2010.

[5] Huan Xu, Ufuk Topcu, Steven H. Low, Christopher R. Clarke, and K. Mani Chandy. Distributed subgradient-based coordination of multiple renewable generators in a microgrid. *IEEE Transactions on Power Systems*, 29(1):23–33, 2014.

[6] Sebastian Hentzelt, Andreas Klingler, and Knut Graichen. Experimental results for distributed model predictive control applied to a water distribution system. In *ISIC*, pages 1100–1106. IEEE, 2014.

[7] Sarah Koehler and Francesco Borrelli. Building temperature distributed control via explicit mpc and "trim and respond" methods. In *Proceedings of European Control Conference*, pages 4334–4339, 2013.

[8] Zhe Liu, Xi Chen, Xingtian Xu, and Xiaohong Guan. A decentralized optimization method for energy saving of hvac systems. In *CASE*, pages 225–230. IEEE, 2013.

[9] Yoshiaki Kuwata and Jonathan How. Decentralized cooperative trajectory optimization for uavs with coupling constraints. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6820–6825, 2006.

[10] Gokhan Inalhan, Dusan M. Stipanovic, and Claire J. Tomlin. Decentralized optimization, with application to multiple aircraft coordination. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 1147 – 1155, 2002.

[11] David Mateos-Nunez and Jorge Cortes. Distributed online convex optimization over jointly connected digraphs. *IEEE Transactions on Network Science and Engineering*, 1(1):23–37, 2014.

[12] Angelia Nedic and Asuman E. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[13] Bjorn Johansson, Tamas Keviczky, Mikael Johansson, and Karl Henrik Johansson. Subgradient methods and consensus algorithms for solving convex optimization problems. In *Proceedings of the 47th IEEE Conference on Decision and Control*, pages 4185–4190, 2008.

[14] Angelia Nedic, Asuman E. Ozdaglar, and Pablo A. Parrilo. Constrained consensus and optimization in multi-agent networks. *IEEE Transactions on Automatic Control*, 55(4):922–938, 2010.

[15] Ermin Wei, Asuman E. Ozdaglar, and Ali Jadbabaie. A distributed newton method for network utility maximization. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 1816–1821, 2010.

[16] Pontus Giselsson and Anders Rantzer. Distributed model predictive control with suboptimality and stability guarantees. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 7272–7277, 2010.

[17] Yudong Ma, Garrett Anderson, and Francesco Borrelli Borrelli. A distributed predictive control approach to building temperature regulation. In *American Control Conference (ACC)*, pages 2089–2094, 2011.

[18] Masahiro Ono and Brian C. Williams. Decentralized chance-constrained finite-horizon optimal control for multi-agent systems. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 138–145, 2010.

[19] Minghui Zhu and Sonia Martínez. On distributed convex optimization under inequality and equality constraints. *IEEE Transactions on Automatic Control*, 57(1): 151–164, 2012.

[20] Dusan Jakovetic, João Manuel Freitas Xavier, and José M. F. Moura. Fast distributed gradient methods. *IEEE Trans. Automat. Contr.*, 59(5):1131–1146, 2014.

[21] Guodong Shi, Alexandre Proutiere, and Karl Henrik Johansson. Continuous-time distributed optimization of homogenous dynamics. In *Allerton*, pages 520–527. IEEE, 2013.

[22] Minghui Zhu and Sonia Martínez. An approximate dual subgradient algorithm for multi-agent non-convex optimization. *IEEE Trans. Automat. Contr.*, 58(6):1534–1539, 2013.

[23] Ruggero Carli and Giuseppe Notarstefano. Distributed partition-based optimization via dual decomposition. In *CDC*, pages 2979–2984. IEEE, 2013.

[24] Ermin Wei, Asuman E. Ozdaglar, and Ali Jadbabaie. A distributed newton method for network utility maximization-i: Algorithm. *IEEE Trans. Automat. Contr.*, 58 (9):2162–2175, 2013.

[25] Nader Motee and Ali Jadbabaie. Distributed multi-parametric quadratic programming. *IEEE Transactions on Automatic Control*, 54(10):2279–2289, 2009.

[26] Stephen Boyd, Lin Xiao, Almir Mutapcic, and Mattingley Jacob. Notes on decomposition methods. Available at http://see.stanford.edu/materials/lsocoee364b/08-decomposition_notes.pdf, 2008.

[27] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[28] Jing Wang and Nicola Elia. Distributed averaging under constraints on information exchange: Emergence of lévy flights. *IEEE Transactions on Automatic Control*, 57 (10):2435–2449, 2012.

[29] Minyi Zhong and Christos G. Cassandras. Asynchronous distributed optimization with event-driven communication. *IEEE Transactions on Automatic Control*, 55 (12):2735–2750, 2010.

[30] Taylan Ayken and Jun-ichi Imura. Asynchronous distributed optimization of smart grid. In *Proceedings of SICE Annual Conference (SICE)*, pages 2098–2102, 2012.

[31] Taylan Ayken and Jun-ichi Imura. Event triggered distributed optimization based on dual decomposition. *SICE Journal of Control, Measurement, and System Integration*, 8(3):221–227, 2015.

[32] Taylan Ayken and Jun-ichi Imura. Diffusion based stopping criterion for distributed optimization. In *Proceedings of the 19th IFAC World Congress*, pages 10512–10517, 2014.

[33] Taylan Ayken and Jun-ichi Imura. Diffusion based stopping criterion for event-triggered distributed optimization. *SICE Journal of Control, Measurement, and System Integration*, 2015. Accepted.