Imperial College London

Imperial College London
Department of Aeronautics

Distributed Optimization in Multi-agent Systems with Coupling Constraints

Alexander Popov

Supervisor: Dr. Eric Kerrigan
Second marker: Dr. Thulasi Mylvaganam

A thesis submitted in partial fulfillment of the requirements for the degree of
MEng, Aeronautical Engineering

June 2021

Abstract

An effective strategy to accelerate the numerical solving of constrained optimization problems is to split the global problem into a set of smaller problems, each of which can be solved in a different parallel process. Since the resulting local problems may be coupled to each other through global constraints, a number of distributed schemes exist to enforce coordination between these processes through an exchange of information. However, these methods often require many iterations to converge, demanding extensive communication between agents, which may be impractical in a real physical network. To address this limitation, we propose connected dominating set sweeping methods, which exploit the structure of a network of agents to solve optimization problems in a fully decentralized manner. As well as constructing a series of local problems that are significantly simpler than those of competing schemes, these algorithms can also be implemented with an efficient communication protocol, making them ideal for real cyber-physical systems. We test the methods on a set of dynamic optimization problems, demonstrating impressive performance in comparison to alternative strategies.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Eric Kerrigan, for his invaluable advice and support over the past year. His suggestions and ideas not only enabled me to explore some fascinating avenues in the world of optimization, but also provided the necessary tools to escape from the many rabbit holes in which I found myself throughout the project.

I would also like to thank Lucian Nita for sharing with me his wisdom on distributed optimization, and for helping me wrestle with the difficulties of multi-processing and parallel computing. Additionally, I must thank Dr. Marta Zagorowska and Eduardo Vila for introducing me to a number of concepts which proved very useful in my own research, and indeed all the other members of Dr. Kerrigan's research group, from whom I learnt a vast amount through their presentations and discussions at weekly meetings.

Finally I would like to thank Jeanique and my family, all of whom are practically perfect in every way.

Contents

Chapter 1

Introduction

In mathematical optimization, we consider problems of the form

$$\min_x f(x) \qquad \text{subject to } \begin{cases} c^j(x) = 0, & j \in \mathcal{P} \\ c^j(x) \leq 0, & j \in \mathcal{Q}, \end{cases} \tag{1.1}$$

with a decision variable $x \in \mathbb{R}^n$ and an objective function $f\colon \mathbb{R}^n \to \mathbb{R}$. $\mathcal{P}$ and $\mathcal{Q}$ are finite sets of indices, and the constraints $c^j(x),\ j \in \mathcal{P} \cup \mathcal{Q}$ define the set of feasible solutions to the problem.

In recent decades, a great amount of research has been dedicated to developing fast and efficient algorithms for numerically solving various forms of (1.1). Such extensive interest stems from the vast array of applications for these methods, ranging from machine learning [1] and control [2] to signal processing [3], economics [4] and operations research [5]. With the remarkable progress in computer performance since the turn of the century, the field has matured beyond the realm of pure academic curiosity, with these algorithms now increasingly applied in real systems and software. Despite this success, there remain plenty of applications limited in scope by an inability to solve optimization problems at a sufficient rate. This is a particularly prevalent obstacle for real-time embedded systems requiring fast, online optimization. For example, model predictive control (MPC) [6, 7] requires a problem to be solved for each new update to a system input. For this reason, while MPC has been widely implemented in the petrochemical industry [8], as well in heating, ventilation and air condition (HVAC) systems [9], it remains impractical for many autonomous vehicles and robots, where system inputs may need multiple adjustments per second.

A good approach to accelerate the solving of large-scale problems is to distribute the computational work over a number of parallel processes, each solving a smaller problem. This is most appropriate if f is separable with respect to some partition of the decision variable,

$$f(x) = \sum_{i=1}^{N} f_i(x_i), \tag{1.2}$$

where $x = (x_1, \ldots, x_N)$ and $x_i \in \mathbb{R}^{n_i}$ are subvectors of x. In this form, we may then minimize f by solving N parallel problems. The solution $x^* \in \mathbb{R}^n$ can then be assembled as $x^* = (x_1^*, \ldots, x_N^*)$, where $x_i^* \in \mathbb{R}^{n_i}$ is the minimizer for f_i. This problem structure is common in optimization problems for multi-agent systems, such as power flow control in smart grids [10] and multi-robot path planning [11]. Then the problem decomposes very naturally, with each agent i finding the minimizer x_i^* for its local objective.

However, even if f is separable, the local decision variables may still be coupled through the constraints,

and so feasibility within each local problem may not guarantee feasibility in the global problem. One way to circumvent this is to formulate each local problem P_i in terms of the associated local variable x_i and a set of other subvectors $\{x_{ji} \mid j \in \mathcal{N}_i\}$, where x_{ji} represents the subvector x_j considered on the i^{th} process, and $\mathcal{N}_i$ defines the set of additional subvectors incorporated into P_i. We note that the native local variable can be equivalently written as x_{ii}. To ensure that P_i remains small relative to the centralized scheme, we desire $|\mathcal{N}_i| \ll N$. This can be enforced naturally if the set of coupling subvectors $\mathcal{C}_i$ is small, in which case we can just define $\mathcal{N}_i = \mathcal{C}_i$. Such a case can occur if, for example, the global constraints can be expressed as a sparse linear system. However, if there is strong coupling within x, then we may require that $\mathcal{N}_i$ be only a small subset of $\mathcal{C}_i$.

Having defined $\mathcal{N}_i$ for $i = 1, \ldots, N$, each process now considers the new problem

$$\min_{\widetilde{x}_i} \sum_{j \in \{i\} \cup \mathcal{N}_i} f_j(x_{ji}) \qquad \text{s.t.} \quad \begin{cases} c_i^l(\widetilde{x}_i) = 0, & l \in \mathcal{P}_i \\ c_i^l(\widetilde{x}_i) \leq 0, & l \in \mathcal{Q}_i, \end{cases} \tag{1.3}$$

where $\widetilde{x}_i$ is an aggregate vector formed by concatenating all x_{ji}, $j \in \{i\} \cup \mathcal{N}_i$ and the functions c_i^l, $l \in \mathcal{P}_i \cup \mathcal{Q}_i$ describe both local and coupling constraints on $\widetilde{x}_i$. To then assemble x^*, x_i^* is extracted from each aggregate solution $\widetilde{x}_i^*$. However, each x_i^* is only known to be feasible with respect to solutions for each coupling variable x_{ji}^*, as determined on the same local process. But the solution to the global problem (1.1) is in fact constructed from each locally determined x_i^*. Then, for a given $j \in \mathcal{N}_i$, we can only guarantee that x_i^* and x_j^* satisfy their associated coupling constraints if $x_{ji}^* = x_j^*$ (an equivalent condition, $x_{ij}^* = x_i^*$ must also hold on the j^{th} process). Therefore, for distributed optimization with coupling, most methods attempt to enforce some consensus between local problems by establishing communication between processes. We now briefly review some of these schemes, considering their application to (1.3).

Related Work

Many distributed algorithms are based on dual decomposition, first proposed in [12]. For a separable objective (1.2), the primal variable is updated in parallel via minimization of the (separable) Lagrangian, and then the assembled updates are used to form a gradient ascent step for the dual problem. An excellent review of dual methods can be found in [13, Ch.6]. Dual decomposition has been employed across a wide array of algorithms [14–22], and it can handle coupling constraints via a consensus-based approach [16], whereby global auxiliary variables are introduced to enforce agreement between local variables on different processes. However, these methods can only guarantee convergence for strictly convex problems, since the gradient ascent scheme relies on strict concavity of the dual problem. This condition can be relaxed with the primal-dual perturbed (PDP) subgradient method [23, 24], which can also be adapted into a distributed form using a consensus approach [10].

Another improvement on dual decomposition which has received a lot of recent attention is the alternating direction method of multipliers (ADMM) [25, 26], which blends dual ascent with augmented Lagrangian methods. Under mild assumptions, the method can be shown to converge for convex problems [27], without the additional need for strict convexity of f. Furthermore, convergence conditions for certain nonconvex problems have also been demonstrated [28]. As a result, in the past decade, ADMM has become perhaps the most widely used distributed algorithm for solving large optimization problems for a variety of applications [29–34]. However, despite plenty of research on its consensus form [27, 35–38], an ADMM consensus-based approach to (1.3) has generally considered only coupling equality constraints. Although an application to coupling inequality constraints has recently been explored [39], there remains a lack of detailed analysis of the strategy's efficiency and convergence properties for such problems.

The methods reviewed above are often assessed according to their ability to converge and the com-

plexity of each iteration. However, both dual decomposition methods and ADMM may require many iterations to converge for large problems [40]. We reason why this may be undesirable by considering optimization in multi-agent systems, where the problem structure naturally lends itself to distributed methods: each agent i finds the minimizer x_i^* for its local objective, subject to local constraints and coupling constraints with other agents. For all of the schemes mentioned, data is passed between agents at each iteration. However, in a real cyber-physical system, there will be some latency in this inter-agent communication. Therefore, with every iteration of the algorithm, there is an additional time delay, independent of the problem complexity.

This practical consideration prompts us to ask whether it is possible to formulate a distributed algorithm which can solve problems with many coupling constraints, whilst requiring only a small number of iterations. Furthermore, we wish to do so without introducing excessive complexity to the local problems.

Contributions

We address these questions by proposing a simple primal method to incrementally construct a feasible solution to the global problem. The strategy sweeps through the system's graph in a decentralized fashion by sequentially solving problems along a path. This is effectively a dynamic programming (DP) recursion [41, 42] through the global system. Although a similar approach to decentralized optimization has been proposed recently for tree graphs [43], this considers only coupling equality constraints, and uses an approximate DP method which may require several sweeps through the graph before sufficient convergence. In contrast, we suggest an exact method with a single sweep through a subgraph, capable of handling any coupling constraints. Our approach does not necessarily guarantee that a feasible solution can be found, and indeed its performance is dependent on the structure of the graph. However, there exist many problems for which this algorithm can work very effectively, achieving fast convergence without excessive communication. Furthermore, our proposal is presented in two forms: the first considers the *neighbourhood coupling* case where $\mathcal{C}_i \subseteq \mathcal{N}_i$, and the second handles *global coupling*, where each agent may be coupled to every other agent in the system. Distributed algorithms for the latter usually invoke some central coordinating process [44, 45], with very few fully decentralized schemes capable of handling such problems [43, 46].

Outline

The thesis is organized as follows. To provide comparison to our primal method, we first review the ADMM algorithm in Chapter 2, stating a consensus-based approach to solve coupled problems [39]. Additionally, we consider the convergence of this scheme, and a possible alternative stopping criterion is discussed. The main contribution of this thesis is then presented in Chapter 3, where we introduce our sweeping algorithm for contraint-coupled decentralized optimization, defining variants for different problem structures and graphs. As a possible improvement to these algorithms, the incorporation of an external active set method [47] is explored, and we propose an amendment to this method for improved performance. In Chapter 4, we describe the implementation of these strategies in Julia, with a particular focus on defining the required communication structure in a distributed computing framework. Furthermore, we discuss application to real systems with high communication latency. Numerical results are presented in Chapter 5, comparing the performance of the new algorithms with ADMM and the centralized solver for a series of dynamic optimization problems. The thesis concludes with Chapter 6, where we summarize the findings and suggest possible directions for future research.

Chapter 2

Alternating Direction Method of Multipliers

2.1 The Algorithm

ADMM can be applied to convex optimization problems of the form

$$\min_{x,z} \; f(x) + g(z) \qquad \text{s.t.} \;\; Ax + Bz = c, \tag{2.1}$$

where $x \in \mathbb{R}^n$, $z \in \mathbb{R}^m$, $c \in \mathbb{R}^p$ and A and B are matrices of appropriate dimensions. Although the method can also handle linear inequality constraints, we restrict ourselves to the formulation in (2.1) as this structure is sufficient for the consensus problem introduced in the next section. Furthermore, we note that this form can be constructed by splitting the decision variable x in the general problem (1.1), provided that $c^j,\ j \in \mathcal{P} \cup \mathcal{Q}$ are linear functions and f is convex. The augmented Lagrangian for (2.1) is defined as

$$L_\rho(x, z, \lambda) = f(x) + g(z) + \lambda^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2, \tag{2.2}$$

where $\lambda \in \mathbb{R}^p$ is the dual variable or Lagrange multiplier, and ρ is a positive scalar parameter. We observe that the right side of (2.2) is constructed from the Lagrangian $L(x, z, \lambda) = f(x) + g(z) + \lambda^T(Ax + Bz - c)$, and a penalty term $\frac{\rho}{2}\|Ax + Bz - c\|_2^2$. We will not discuss in detail the duality theory that underpins the algorithm, but instead the reader is referred to [13, 48] for background.

The ADMM iterations are then defined as:

$$x^{k+1} := \operatorname*{argmin}_x L_\rho(x, z^k, \lambda^k) \tag{2.3}$$

$$z^{k+1} := \operatorname*{argmin}_z L_\rho(x^{k+1}, z, \lambda^k) \tag{2.4}$$

$$\lambda^{k+1} := \lambda^k + \rho(Ax^{k+1} + Bz^{k+1} - c), \tag{2.5}$$

where the λ update (2.5) corresponds to gradient ascent on the dual problem

$$\max_\lambda \; \inf_{x,z} L_\rho(x, z, \lambda). \tag{2.6}$$

By including the penalty term, this can guarantee that the dual objective function in (2.6) is differentiable for a broader class of problems than would be possible with the unaugmented Lagrangian. Steps (2.3) and (2.4) recover the optimal points for the primal variables x and z respectively, given a dual optimal point λ^k. Crucially, x and z are not updated simultaneously, but rather in sequence, each with the other fixed. This ensures that, if f and g are separable, then the primal updates can be decomposed and solved as several independent parallel problems.

ADMM can be summarized as the combination of three key ingredients: dual decomposition, the augmented Lagrangian and splitting of the primal variable. Dual decomposition alone allows for the primal update to be parallelized over several processes. By augmenting the Lagrangian, the dual ascent can converge under milder conditions, but the addition of the penalty term compromises the function's separability. Finally, by splitting the decision variable, the sequential primal updates can recover this property. The result is a robust, parallelizable algorithm for convex optimization problems.

2.2 Consensus Form

We first consider the global consensus problem for N parallel processes

$$\begin{aligned} &\min_{\{x_i\}} \sum_{i=1}^{N} f_i(x_i) \\ &\text{s.t. } x_i - z = 0,\ i = 1, \ldots, N, \end{aligned} \tag{2.7}$$

where each $x_i \in \mathbb{R}^n$ is some local variable, $\{x_i\}$ is the set of all N local variables and $z \in \mathbb{R}^n$ is a global variable, common to all N processes. This problem is equivalent to the unconstrained (1.1), with separable f but x not partitioned. In such a case, we can still exploit the decomposability of the problem by minimizing each f_i in parallel, but then we additionally must enforce agreement between each local decision variable. In other words, x_i is the i^{th} problem's opinion of what x should be, and the auxiliary variable z ensures that all N opinions are the same.

The augmented Lagrangian for (2.7) is then

$$L_\rho(\{x_i\}, z, \lambda) = \sum_{i=1}^{N} \left(f_i(x_i) + \lambda_i^T (x_i - z) + \frac{\rho}{2} \|x_i - z\|_2^2 \right), \tag{2.8}$$

and the ADMM algorithm reduces to

$$x_i^{k+1} \coloneqq \operatorname*{argmin}_{x_i} \left(f_i(x_i) + (\lambda_i^k)^T (x_i - z^k) + \frac{\rho}{2} \|x_i - z^k\|_2^2 \right) \tag{2.9}$$

$$z^{k+1} \coloneqq \frac{1}{N} \sum_{i=1}^{N} x_i^{k+1} \tag{2.10}$$

$$\lambda^{k+1} \coloneqq \lambda^k + \rho(x_i^{k+1} - z^{k+1}). \tag{2.11}$$

The simplification of the z update (2.10) is detailed in [27]. The algorithm now has a very intuitive form: each process solves its local problem, then the global variable is set as the mean of these solutions, and the dual is updated accordingly. Conveniently, almost all of the required computations are in step (2.9), which is parallelizable.

2.3 Neighbourhood Consensus in Graphs

To illustrate how a consensus approach can be used in coupled problems, a multi-agent system with N agents is considered. The communication topology is described by a connected, undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the set of vertices (agents) and $\mathcal{E}$ is the set of edges (i, j), such that agent i and agent j can exchange information with each other. Then $\mathcal{N}_i = \{j \mid (i, j) \in \mathcal{E}\}$ is the set of neighbours of agent i, and $\mathcal{S}_i = \{i\} \cup \mathcal{N}_i$ is the i^{th} neighbourhood. Additionally we restrict ourselves to the neighbourhood coupling case $\mathcal{C}_i \subseteq \mathcal{N}_i$, meaning that each agent is coupled only with neighbours. An example graph with ten vertices is shown in Figure 2.1.

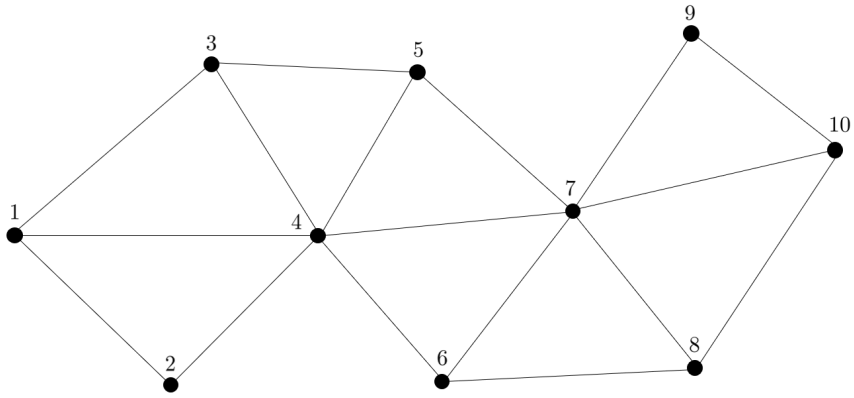


Figure 2.1: An undirected graph with $\mathcal{V} = \{1, \ldots, N\}$ and $N = 10$. According to our definitions, we can identify the neighbours of each vertex, e.g., $\mathcal{N}_4 = \{1, 2, 3, 5, 6, 7\}$.

Each agent i can then solve (1.3), which we restate here in the more concise form

$$\min_{\widetilde{x}_i \in \widetilde{\mathcal{X}}_i} \sum_{j \in \mathcal{S}_i} f_j(x_{ji}), \tag{2.12}$$

where $\widetilde{\mathcal{X}}_i$ defines the feasible region within which $\widetilde{x}_i$ must lie. However, we additionally require $x^*_{ji} = x^*_j$ and $x^*_{ij} = x^*_i$ for all $(i, j) \in \mathcal{E}$. Therefore, the local problem (2.12) is amended to

$$\begin{aligned} &\min_{\widetilde{x}_i \in \widetilde{\mathcal{X}}_i} \sum_{j \in \mathcal{S}_i} f_j(x_{ji}) \\ &\text{s.t. } x_{ji} = z_j,\ j \in \mathcal{S}_i, \end{aligned} \tag{2.13}$$

where z_j is the auxiliary variable associated with agent j. The distributed structure of (2.13) is known as the general form consensus problem [27], and we will equivalently refer to it as the neighbourhood consensus problem over $\mathcal{G}$. The ADMM iterations for (2.13) are as follows:

$$\widetilde{x}_i^{k+1} \coloneqq \operatorname*{argmin}_{\widetilde{x}_i \in \widetilde{\mathcal{X}}_i} \left(\widetilde{f}_i(\widetilde{x}_i) + (\widetilde{\lambda}_i^{\,k})^T(\widetilde{x}_i - \widetilde{z}_i^{\,k}) + \frac{\rho}{2}\|\widetilde{x}_i - \widetilde{z}_i^{\,k}\|_2^2\right) \tag{2.14}$$

$$z_i^{k+1} \coloneqq \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} x_{ij}^{k+1} \tag{2.15}$$

$$\widetilde{\lambda}_i^{\,k+1} \coloneqq \widetilde{\lambda}_i^{\,k} + \rho(\widetilde{x}_i^{k+1} - \widetilde{z}_i^{k+1}), \tag{2.16}$$

where $\widetilde{z}_i$ is the aggregate vector formed by concatenating $\{z_j \mid j \in \mathcal{S}_i\}$, and $\widetilde{f}_i(\widetilde{x}_i)$ is the summed objective in (2.13). Within each iteration k, there are two points of inter-agent communication: first, agent i sends each x_{ji}^{k+1} to the corresponding neighbour and in return receives x_{ij}^{k+1} before the z_i update (2.15). Then, agent i sends z_i^{k+1} to each neighbour and receives z_j^{k+1} from each neighbour before the $\widetilde{\lambda}_i$ update (2.16). The entire neighbourhood consensus ADMM scheme is stated in Algorithm 1.

Algorithm 1 Neighbourhood Consensus ADMM

for $i \in \mathcal{V}$, in parallel **do**
  Initialize: $k = 0$, $\widetilde{z}_i^0 = 0$, $\widetilde{\lambda}_i^0 = 0$.
  repeat
    Compute $\widetilde{x}_i^{k+1}$ using (2.14).
    Send x_{ji}^{k+1} to each node $j \in \mathcal{N}_i$.
    Receive x_{ij}^{k+1} from each node $j \in \mathcal{N}_i$.
    Compute z_i^{k+1} using (2.15).
    Send z_i^{k+1} to each node $j \in \mathcal{N}_i$.
    Receive z_j^{k+1} from each node $j \in \mathcal{N}_i$.
    Compute $\widetilde{\lambda}_i^{k+1}$ using (2.16).
    Set $k = k + 1$.
  until Stopping criterion satisfied.
end for

In summary, each agent i locally solves an optimization problem for its neighbourhood, then constructs a new auxiliary variable, representing the value of x_i to which all opinions in the neighbourhood should agree. Then these consensus variables are passed throughout the neighbourhood so that they are known to the relevant agents. Finally, each agent updates its dual variable according to the agent's disagreement with its neighbours, thus allowing it to construct a new local problem to solve in the next iteration.

2.4 Convergence and Stopping Criterion

Convergence results for the global consensus problem (2.7) are well-established. First we define at each iteration k the primal (consensus) residual $r^k = (x_1^k - z^k, \ldots, x_N^k - z^k)$ and the objective value f^k. Then, if the global objective f is closed, proper and convex, and the (unaugmented) Langrangian has a saddle point, ADMM guarantees that $r^k \to 0$ and $f^k \to f^*$, where f^* is the optimal value [27]. If we can further assume that, for each $i = 1, \ldots, N$, the local objective f_i is strongly convex and its gradient ∇f_i is Lipschitz continuous, then the global variable z and the dual variable λ converge linearly to their respective optima z^* and λ^* [35].

Currently, there is a lack of theoretical analysis of the neighbourhood consensus problem, particularly in the general case of coupling constraints between local variables. We discuss below some trivial extensions from the global consensus problem, as well as the challenges involved in developing more powerful results.

General form consensus can be treated as N consensus problems, each spanning a vertex i and its neighbours $\mathcal{N}_i$. By constructing each of these problems over a star graph $\mathcal{G}_i$ (see Figure 2.2), the above results can then be confirmed independently for each consensus problem $P_{\mathcal{G}_i}$, $i = 1, \ldots, N$. That is, within some neighbourhood $\mathcal{S}_i$, the opinions x_i^k and x_{ij}^k, $j \in \mathcal{N}_i$ will all converge towards agreement, provided that the global assumptions in [27] are valid over the analogous neighbourhood problem. However, this describes the behaviour of Algorithm 1 when each P_i is subject only to consensus

constraints. This is essentially equivalent to unconstrained minimization of the global objective f. When coupling between agents is introduced, the elegant solutions of [27, 35] cannot be invoked, since the local consensus problems $P_{\mathcal{G}_i}$, $i = 1, \ldots, N$, are no longer independent of each other. As an example, consider any $(i, j) \in \mathcal{E}$: then if agents i and j are coupled through their constraints, we know that x_i and x_{ji} will depend on each other. But according to our extension of the global consensus convergence results, x_i will appear in $P_{\mathcal{G}_i}$, whereas x_{ji} is associated with $P_{\mathcal{G}_j}$. Therefore $P_{\mathcal{G}_i}$ and $P_{\mathcal{G}_j}$ are themselves coupled, and so their respective convergence properties cannot be derived independently of one another.

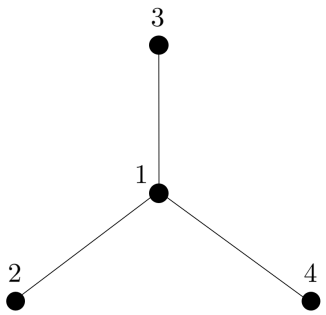


Figure 2.2: The star graph $\mathcal{G}_1$ formed from the neighbourhood $\mathcal{S}_1$ of the graph in Figure 2.1. This subgraph can be associated with a consensus problem $P_{\mathcal{G}_1}$, constrained by $x_1 = x_{21} = x_{31} = x_{41}$, via the auxiliary variable z_1.

Despite a lack of understanding of the interplay between the consensus problem and the coupled problem, it is actually perfectly reasonable to adopt the naive approach of simply applying ADMM and reaping any rewards that it may offer. This can be justified by considering some general (possibly nonconvex) optimization problem with a decision variable split into N constraint-coupled local variables. In an attempt to shape this problem into a form suitable for ADMM, we can restate it as an unconstrained problem, with the constraints implicitly included in a transformed objective function F, equivalent to f defined over a stricter domain [49]. Now we solve the problem

$$\min \ F(x), \tag{2.17}$$

where the domain of the new objective is defined as

$$\mathbf{dom}\, F = \{x \in \mathbf{dom}\, f \mid c^j(x) = 0,\ j \in \mathcal{P},\ c^j(x) \leq 0,\ j \in \mathcal{Q}\}. \tag{2.18}$$

Even if f was originally convex, the transformed problem may not preserve this convexity. Nevertheless, we have still obtained an unconstrained global problem which can be tackled in a distributed fashion via a general form consensus approach, i.e., splitting the objective and variables, and introducing linear consensus constraints. The problem structure is now well-suited to ADMM, but for the nonconvex objective function. Although convergence guarantees may not exist for this transformed problem, it has been observed that ADMM can often yield good performance over nonconvex functions. This may mean fast convergence to a local solution, or even just to a feasible suboptimal point with a better objective value than may be achieved by other methods [27].

Even if Algorithm 1 can converge for a given problem, exact consensus may not be achieved within a finite number of iterations. For this reason, a stopping criterion $\|r^k\|_2 \leq \epsilon$ is often introduced, where ϵ is some small positive upper bound and r^k is the primal residual of the overall neighbourhood consensus problem, with a squared norm

$$\|r^k\|_2^2 = \sum_{i=1}^{N} \|\widetilde{x}_i^k - \widetilde{z}_i^k\|_2^2. \tag{2.19}$$

To ensure fast termination of neighbourhood consensus ADMM, ϵ would ideally be relatively large, given that we are more interested in satisfying the coupling constraints than necessarily enforcing agreement between agents. However, it has already been reasoned that deviation between local opinions may lead to infeasible solutions. This tradeoff has been considered before [39], with feasibility of the global problem guaranteed by a robust feasibility method [50], in which the coupling constraints are relaxed and the local constraints tightening according to the size of ϵ.

For certain applications, it may be more appropriate to simply terminate the algorithm when a globally feasible solution is found. That is, $x^k \in \mathcal{X}$, where $x^k = (x_1^k, \ldots, x_N^k)$. At each iteration, the local problem always finds a locally feasible solution, so this stopping criterion can be checked by comparing the solutions of neighbouring nodes and confirming that their coupling constraints are satisfied. Although this strategy may result in a worse objective value than that obtained with the consensus residual criterion, it can guarantee a globally feasible suboptimal solution with potentially much fewer iterations. Furthermore, it is a very simple criterion to apply, whereas the consensus residual approach may yield great variation in performance, depending on the choice of ϵ.

We should also consider how these stopping criteria may be checked in practice. To verify a globally feasible solution, each agent i must compare its value x_i^k with the values x_j^k, $j \in \mathcal{N}_i$ from its neighbours, yielding a Boolean value to indicate whether coupling constraints have been satisfied. Then an additional protocol must be established to check all Boolean values across the graph. This procedure could be simplified by instead designating some central process which gathers and compares x_i^k, for all $i \in \mathcal{V}$, then indicates to all agents whether the criterion has been satisfied. Which of these strategies is more viable very much depends on the nature of the particular system.

In contrast, the consensus residual criterion can be redefined so that each agent i instead monitors $\|r_i^k\|_2 = \|\widetilde{x}_i^k - \widetilde{z}_i^k\|_2$. Then the central process only needs to gather a Boolean value from each agent to check whether all processes can terminate. If not, then agent must continue running the algorithm until all N criteria are satisfied. Although this protocol involves less communication than required for the feasibility check, we emphasise that, in a system with high latency, the benefit of fewer iterations may greatly outweigh that of less communication per iteration.

For all this discussion of the behaviour of ADMM, we above all should emphasize that the method often demonstrates very good convergence in practice. Beyond its theoretical guarantees, this empirical observation is perhaps the primary reason why it has been the focus of so much research interest in the last decade. However, much like a number of other primal-dual distributed methods, ADMM may only achieve sufficient convergence after a relatively large number of iterations. Since successive iterations tend to be quite fast, this characteristic may be of little consequence in many contexts. However, if we try to implement the neighbourhood consensus scheme over a real, physical multi-agent system in which the passing of data between agents is not instantaneous, then each iteration will be delayed by the transmission of local and auxiliary variables. Our suggestion to introduce a stopping criterion based only on feasibility can significantly reduce the total effect of this latency over the course of the algorithm, but it will do so at the expense of optimality.

These issues motivate us to consider a change of course: rather than simultaneously solving optimization problems at all agents (and hence requiring a number of iterations to enforce coordination), we instead explore the idea of sequentially solving one problem at a time, thus enabling us to guarantee that each successive problem does not conflict with its predecessors.

Chapter 3

Staged Optimization Over Connected Dominating Sets

The principle of solving an optimization problem in stages was first explored by Bellman in the 1940s [51] with the advent of dynamic programming (DP) [42]. The method considers processes which can be formulated as a sequence of decisions, and exploits the simple idea that if a sequence of decisions is optimal, then any smaller portion of that sequence is optimal for the corresponding subproblem [41] - this is Bellman's *principle of optimality*. Consider a multi-stage optimization problem, with each stage $k = 0, 1, \ldots, N$ characterized by a decision variable x_k and a value function $F_k(x_k)$, which tracks the cost of the decisions as the algorithm progresses. Then a DP scheme can be applied as a backward recursion [52], initialized with the value function for the final stage

$$F_N(x_N) = f_N(x_N). \tag{3.1}$$

By defining a relation between successive stages

$$x_{k+1} = g_k(x_k), \quad k = 0, 1, \ldots, N-1, \tag{3.2}$$

the algorithm can then proceed backwards from $k = N-1$ to $k = 1$, calculating the associated value function at each stage:

$$F_k(y) = \min_{x_k \in \mathcal{X}_k} f_k(x_k) + F_{k+1}\big(g_k(x_k)\big) \quad \text{s.t. } g_{k-1}(y) = x_k, \tag{3.3}$$

where $\mathcal{X}_k$ is the feasible region for x_k and y is the function argument. We notice that each F_k is dependent on the value function obtained from the previous step, F_{k+1}. Then the backward recursion terminates at $k = 0$ by solving the constructed optimization problem

$$F^* = F_0 = \min_{x_0 \in \mathcal{X}_0} f_0(x_0) + F_1\big(g_0(x_0)\big), \tag{3.4}$$

yielding the overall objective value F^* and the associated decision x_0^*. Then the entire sequence of decisions $x^* = (x_0^*, \ldots, x_N^*)$ can be determined by successive evaluation of (3.2), thus completing the algorithm. Although a backward recursion has been described here, the concept can of course be applied with a forward scheme instead.

Dynamic programming is very naturally suited to staged problems over a time interval (i.e., each stage corresponds to a discrete step within the interval $[0, N]$), but it can equally be applied over a graph, where a vertex corresponds to a stage in the recursion, and an edge connects consecutive stages. Then the DP recursion is akin to a sweep through the graph, passing through the vertices in sequence. Such a scheme seems ideal for a network of agents: as well as achieving optimality without invoking dual methods (and hence avoiding certain requirements on the nature of the objective f), an algorithm that considers local problems in sequence will not require any synchronization between the agents, thus avoiding the need for any centralized coordination.

However, there remains a potential pitfall: if each decision variable x_k is a vector, then, as the recursion progresses, it may become exponentially more difficult to calculate each value function. If the vectors are large, and if the graph has many vertices, then the scheme will be susceptible to the so-called *curse of dimensionality* [53]. This can be mitigated by constructing approximate value functions in place of the exact iteration (3.3), but then multiple sweeps through the graph may be required to achieve optimality [43]. As mentioned in Chapter 1, a multi-sweep algorithm only increases the total inter-agent communication, and so it is impractical for systems with high latency.

With this in mind, the algorithms presented in this chapter attempt to solve optimization problems in a decentralized fashion with only one sweep through the graph, whilst avoiding the effects of the curse of dimensionality. The proposed methods bear resemblance to the classic dynamic programming recursion (3.3), but several differences, inspired by further exploitation of the problem structure, will become apparent.

3.1 Sweeping with Neighbourhood Coupling

As explained in the previous chapter, for any scheme in which each agent solves an optimization problem over a constraint-coupled neighbourhood, we must be wary of conflicting solutions arising from differing assumptions of a given agent's decision. The consensus-based ADMM scheme addressed this by iteratively converging towards some agreement between x_i and x_{ij}, for all $j \in \mathcal{N}_i$. However, there is a much simpler way to ensure that, within some neighbourhoood $\mathcal{S}_i$, the set of extracted solutions $\{x_j^* \mid j \in \mathcal{S}_i\}$ satisfy coupling constraints with each other. If agent i solves its neighbourhood problem P_i, and then sends its opinion x_{ji}^* to the relevant neighbour $j \in \mathcal{N}_i$, then each neighbour can set its local decision to $x_j^* = x_{ji}^*$ without requiring any local computation. Each neighbour can also copy x_i^* from agent i, setting $x_{ij}^* = x_i^*$. In this way, the agent j now knows two of the solutions for its own neighbourhood, enabling it to greatly reduce the complexity of its own local problem P_j. Extending this concept, we may be able to solve a series of neighbourhood problems sequentially, such that each problem in the sequence can be simplified by incorporating solutions from the preceding problems. This is the fundamental idea behind the decentralized sweeping methods proposed in this chapter.

To formulate a general algorithm based on this notion, some additional properties of the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ must first be introduced. We define a dominating set $\mathcal{D} \subset \mathcal{V}$, such that every node $i \in \mathcal{V}$ which is not in $\mathcal{D}$ is neighboured by at least one node in $\mathcal{D}$. Furthermore, a connected dominating set (CDS) is a dominating set $\mathcal{D}$ with the additional property that there exists a path between any two nodes in $\mathcal{D}$, such that all the nodes passed on this path are themselves members of $\mathcal{D}$. This equivalently means that, from the nodes contained in $\mathcal{D}$, along with a subset of $\mathcal{E}$, a connected subgraph $\mathcal{G}^{\mathcal{D}}$ can be induced. Figure 3.1 depicts a graph in which two possible dominating sets are highlighted.

Additionally, we define a tree as a class of graph for which there is only one possible path between any two nodes. An undirected tree graph can be equivalently described as a connected graph wihout any cycles. Within a tree, a node with only one neighbour is known as a leaf, and we denote the set of leaves of a tree as $\mathcal{L}$. Figure 3.2 shows an example of a tree graph, from which we can observe that

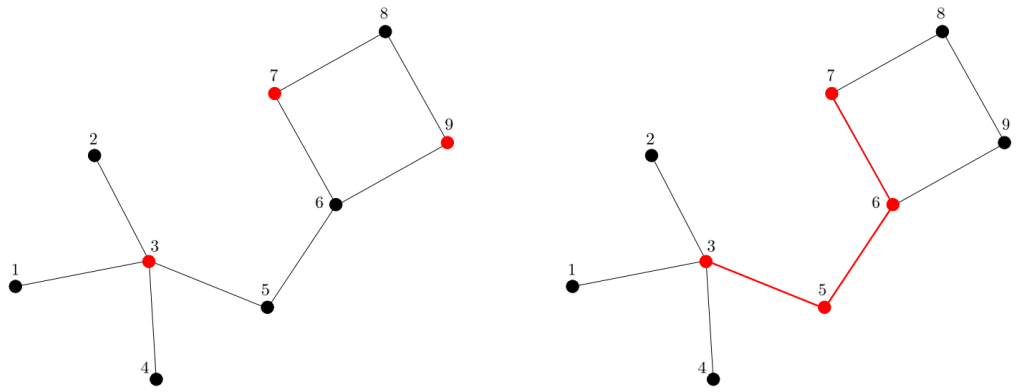


Figure 3.1: A graph with a possible (unconnected) dominating set $\{3, 7, 9\}$ (left) and a connected dominating set $\{3, 5, 6, 7\}$ (right), each indicated in red. Furthermore, the latter is a minimum connected dominating set for the graph, and the connecting edges which form the subgraph $\mathcal{G}^{\mathcal{D}}$ are highlighted in red.

there is only ever one minimum connected dominating set of a tree $\mathcal{G}(\mathcal{V}, \mathcal{E})$, and it is equal to $\mathcal{V} \setminus \mathcal{L}$. We also define a branch at some node $u \in \mathcal{V}$ as a subtree of $\mathcal{G}(\mathcal{V}, \mathcal{E})$ in which u is a leaf. Clearly, the number of branches at each node is equivalent to the number of neighbours. The distance $d(u, v)$ between two nodes $u, v \in \mathcal{V}$ is defined as the number of edges in the shortest path between them, and the graph eccentricity of u is given by $\max\{d(u, v) \mid u, v \in \mathcal{V}, u \neq v\}$. The smallest eccentricity value across all nodes of $\mathcal{G}$ is known as the graph's radius r, and the set of nodes with eccentricity equal to this radius constitutes the centre of the graph. It can be easily observed that the centre of any tree is either a single node or a pair of adjacent nodes. In the example graph shown in Figure 3.2, the radius is 3 and the centre is at node 6.

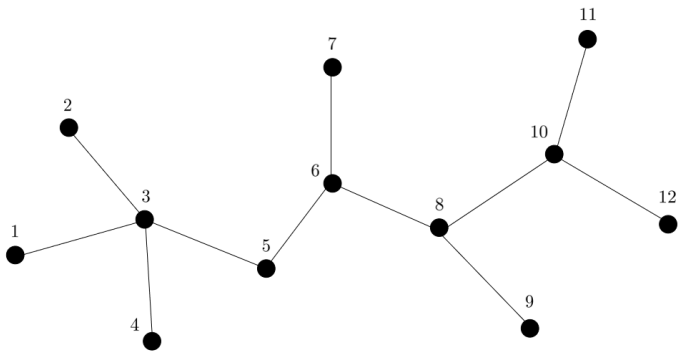


Figure 3.2: A tree with $N = 12$ nodes, a set of leaves $\mathcal{L} = \{1, 2, 4, 7, 9, 11, 12\}$ and a minimum connected dominating set $\mathcal{D} = \{3, 5, 6, 8, 10\}$.

We now introduce the *connected dominating set branch sweep* (CDS-BS) method for decentralized optimization over trees with neighbourhood coupling, i.e., $\mathcal{C}_i \subseteq \mathcal{N}_i$ for all $i \in \mathcal{V}$. In the following description, we express function arguments as sets, instead of as aggregate vectors, e.g., in place of $\widetilde{x}_i$, we use $\{x_{ji},\ j \in \mathcal{S}_i\}$. The procedure begins at some designated root node v_0, which solves the optimization problem P_{v_0}, defined as:

$$\begin{aligned} \min_{\{x_{jv_0},\, j \in \mathcal{S}_{v_0}\}} & \sum_{j \in \mathcal{S}_{v_0}} f_j(x_{jv_0}) \\ \text{s.t. } & c^l_{v_0}(\{x_{jv_0},\ j \in \mathcal{S}_{v_0}\}) = 0, \quad l \in \mathcal{P}_{v_0} \\ & c^l_{v_0}(\{x_{jv_0},\ j \in \mathcal{S}_{v_0}\}) \leq 0, \quad l \in \mathcal{Q}_{v_0}. \end{aligned} \tag{3.5}$$

From this root, a different sequence of problems can be constructed and solved along each branch, sweeping outwards to the leaves. To ensure that the longest sequence is as small as possible, v_0 can be set as a graph centre. For each node $i \in \mathcal{V} \setminus \{v_0\}$, the simplified problem P_i has a reduced set of decision variables $\bar{\mathcal{S}}_i = \mathcal{N}_i \setminus \{p_i\}$, where p_i is the unique 'parent' which sends $x^*_{ip_i}$ and $x^*_{p_i}$ to agent i. Equivalently, p_i is the sole member of the set $\{j \in \mathcal{N}_i \mid d(v_0, j) = d(v_0, i) - 1\}$. P_i can be constructed by first observing that the set of constraint functions in a non-reduced neighbourhood problem can be partitioned as follows:

$$\begin{aligned}\Big\{c_i^l(\{x_{ji},\ j \in \mathcal{S}_i\}),\ l \in \mathcal{P}_i \cup \mathcal{Q}_i\Big\} = &\Big\{c_i^l(\{x_{ji},\ j \in \mathcal{S}_i\}),\ l \in \bar{\mathcal{P}}_i \cup \bar{\mathcal{Q}}_i\Big\} \cup \\ &\Big\{c_i^l(x_i, x_{p_i i}),\ l \in \hat{\mathcal{P}}_i \cup \hat{\mathcal{Q}}_i\Big\},\end{aligned} \tag{3.6}$$

where $\hat{\mathcal{P}}_i \subset \mathcal{P}_i, \hat{\mathcal{Q}}_i \subset \mathcal{Q}_i$ are the sets of indices corresponding to constraint functions dependent only on the variables x_i and $x_{p_i i}$, and we define $\bar{\mathcal{P}}_i = \mathcal{P}_i \setminus \hat{\mathcal{P}}_i$, $\bar{\mathcal{Q}}_i = \mathcal{Q}_i \setminus \hat{\mathcal{Q}}_i$. In the sweeping procedure, each branch node i can trivially set $x_i = x^*_{ip_i}$ and $x_{p_i i} = x^*_{p_i}$ using the parent's solution, and so the constraints c_i^l, $l \in \hat{\mathcal{P}}_i \cup \hat{\mathcal{Q}}_i$ will take constant values and thus are redundant. Therefore, each simplified problem P_i, $i \in \mathcal{V} \setminus \{v_0\}$ is given by

$$\begin{aligned}\min_{\{x_{ji},\, j \in \bar{\mathcal{S}}_i\}} \ & \sum_{j \in \bar{\mathcal{S}}_i} f_j(x_{ji}) \\ \text{s.t. } & c_i^l(\{x_{ji},\ j \in \bar{\mathcal{S}}_i\}, x_i^*, x^*_{p_i i}) = 0, \quad l \in \bar{\mathcal{P}}_i \\ & c_i^l(\{x_{ji},\ j \in \bar{\mathcal{S}}_i\}, x_i^*, x^*_{p_i i}) \leq 0, \quad l \in \bar{\mathcal{Q}}_i.\end{aligned} \tag{3.7}$$

We note that (3.7) not only has fewer variables and constraints than the original local problem, but also that some of the remaining constraints will be simplified since the variables x_i and $x_{p_i i}$ are replaced by the constants x_i^* and $x^*_{p_i i}$. For example, a number of coupling constraints may now be reduced to bounds on a single variable. As soon as a branch node $i \in \mathcal{V} \setminus \{v_0\}$ has received the solutions from its parent p_i, the reduced problem P_i can be constructed and solved. From the complete set of solutions $\{x^*_{ji},\ j \in \mathcal{S}_i\}$, agent i can then send x_i^* and x^*_{ji} to each 'child' $j \in \bar{\mathcal{S}}_i$. By implementing this strategy at all branch nodes, we naturally cause the recursive scheme to propagate from the root, constructing an increasing set of agent decisions as the sweep moves towards the leaves of the tree. By the time all branches have been swept through, each agent $i \in \mathcal{V}$ will know its decision x_i^*, collectively constituting a solution to the global problem. The CDS-BS procedure at each node is summarized in Algorithm 2.

Algorithm 2 CDS Branch Sweep

```
for i ∈ V, in parallel do
  if i = v_0 then
    Solve P_{v_0} (3.5) and retain x*_{v_0}.
    For each j ∈ N_i, send x*_i and x*_{ji} to node j.
  else
    Wait until x*_{p_i} and x*_{ip_i} received from node p_i.
    Set x*_i = x*_{ip_i} and x*_{p_i i} = x*_{p_i}.
    if S̄_i ≠ ∅ then
      Solve P_i (3.7).
      For each j ∈ S̄_i, send x*_i and x*_{ji} to node j.
    end if
  end if
end for
```

We note some useful observations about Algorithm 2: since the leaves of the tree are the only nodes at which no optimization problem is ever solved, it is clear that the set of solving agents is equal to the minimum connected dominating set of the graph. Furthermore, by limiting ourselves to trees with neighbourhood coupling, the sweeping scheme can branch out from the root in multiple directions without the risk of ever solving for the same node twice with potential discrepancy between the two opinions.

A schematic diagram of a possible sweep over the example tree graph from Figure 3.2 is shown in Figure 3.3, starting at the root $v_0 = 6$. At the first stage, agent 6 solves its local problem and sends x_6^* and x_{j6}^* to the nodes $j \in \{5, 7, 8\}$. Now the set of agents $\mathcal{S}_6 = \{5, 6, 7, 8\}$ all know their local decisions, and indeed agents 6 and 7 are not required to do any further computation (since $7 \in \mathcal{L}$).

Then at the second stage, the reduced problems P_5 and P_8 are solved at agents 5 and 8 respectively. Agent 5 can then send x_5^* and x_{35}^* to agent 3, while agent 8 sends x_8^* and x_{j8}^* to each neighbour $j \in \{9, 10\}$. With this information communicated, the set of agents that know their local decisions has expanded to $\{3, 5, 6, 7, 8, 9, 10\}$, and agents 5, 8 and 9 no longer have any optimization problem to solve.

In the third stage, agents 3 and 10 each solve their reduced problems and distribute the various blocks of the solutions to the remaining neighbours, all of which are leaves. Hence the procedure is complete, and each node $i \in \mathcal{V}$ now stores its associated solution x_i^*, as indicated in the fourth panel of the figure.

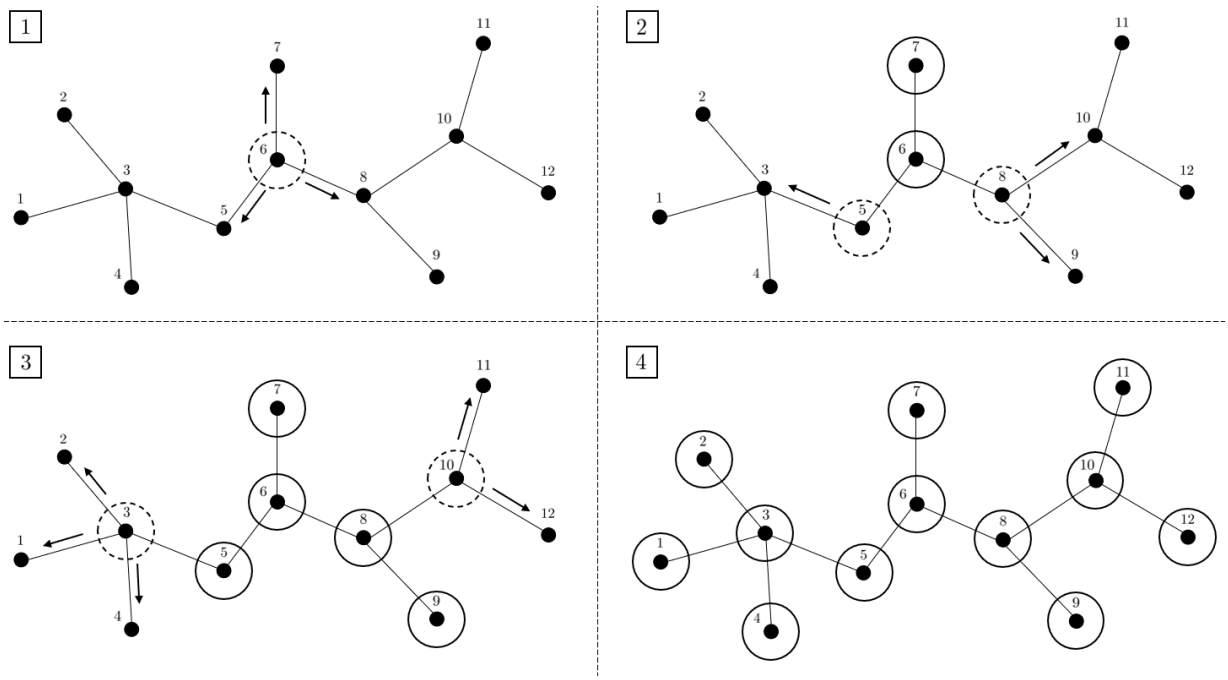


Figure 3.3: A schematic diagram of the branch sweep algorithm over a tree graph. Circles with dashed lines indicate that the enclosed agent is solving an optimization problem at that stage, and circles with solid lines illustrate that the corresponding node is storing its local solution. The arrows represent communication from solving agents to unsolved neighbours.

Algorithm 2 requires that each node $i \in \mathcal{V}$ know whether it is the root. Additionally, each node $i \in \mathcal{V} \setminus \{v_0\}$ should know its parent p_i. However, beyond these conditions, the agents do not require any further knowledge of the overall graph structure (we take for granted that each node is aware of its neighbours, as this is a trivial assumption of any graph-based algorithm). In fact, these conditions can be removed if we instead establish a new enumeration of $\mathcal{V}$, by labelling each vertex according to

its distance from the root. For the tree in Figures 3.2 and 3.3, this alternative enumeration is shown in Figure 3.4. Since the fact that a given node is aware of the numbering of its neighbours is a universal assumption for all graphs, an agent i can know if it is the root by verifying if $i = 0$. Furthermore, if $i \neq 0$, then its parent is simply the unique neighbour with value $i - 1$, and its reduced problem is in terms of all neighbours with value $i + 1$.

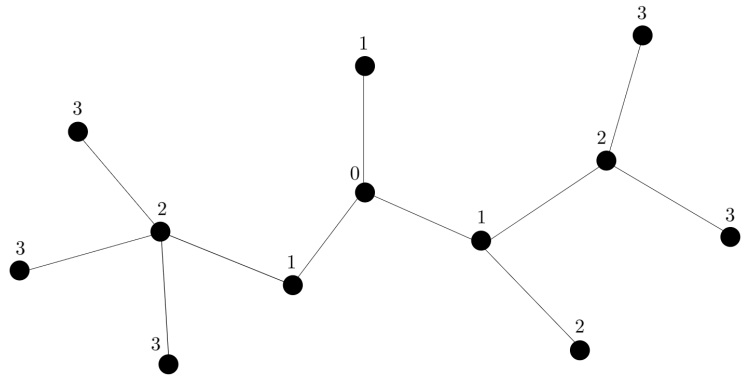


Figure 3.4: An alternative numbering of the vertices in our example tree graph, based on distance from the root.

While some resemblance to the dynamic programming recursion of (3.3) may be observed, we should beware that the proposed branch sweeping method is not necessarily being applied to multi-stage optimization problems. Therefore, in general, Bellman's principle of optimality will not hold, and hence a globally optimal solution may not be guaranteed. Furthermore, since we suggest CDS-BS for graphs in which local variables may be coupled by inequality constraints, then an exact relation (3.2) between stages may not exist. Not only does this prevent us from ensuring optimality, but it also has the potential to jeopardise the feasibility of our obtained solutions. For example, if there exist the coupling constraints $x_1 + x_2 \leq 1$ and $x_2 + x_3 \leq 1$ between the scalar decision variables associated with a set of nodes $\{1, 2, 3\}$, and some stage results in $(x_1^*, x_2^*) = (0, 1)$, then a later stage to determine x_3^* can only find a feasible solution if $\mathcal{X}_3' \cap \{x_3 \mid x_3 \leq 0\}$ is nonempty, where $\mathcal{X}_3'$ is the set of values of x_3 permitted by the combination of all other relevant constraints.

We are therefore limited by the following condition: CDS-BS can only be applied to a global problem if there exist feasible solutions to the root problem P_{v_0} and all subsequent simplified branch problems $P_i,\ i \in \mathcal{V} \setminus \{v_0\}$, where each branch problem is dependent on the solution of the parent problem. A formal definition of the class of global problems which satisfy this criterion is beyond the scope of this thesis, but the example presented in Chapter 5 demonstrates that there exist reasonable problems for which CDS-BS can find feasible and optimal solutions.

3.2 Sweeping with Global Coupling

The branch sweeping algorithm can be amended to handle the more general case of global coupling across the graph, i.e., if there exist constraints between non-adjacent nodes. We can preserve the idea of solving only at nodes within a connected dominating set, however we now have to ensure that every fixed solution is incorporated in all subsequent reduced problems. We adapt to this requirement by proposing the *connected dominating set path sweep* (CDS-PS) method, applicable to any graph for which there exists a connected dominating set that induces a path $\mathcal{G}^{\mathcal{D}}(\mathcal{D}, \mathcal{E}^{\mathcal{D}})$.

Instead of sweeping outwards from the centre along multiple paths, CDS-PS progresses along the path $\mathcal{G}^{\mathcal{D}}$, with the root node v_0 set as an endpoint of this path, such that the sweep terminates at the other endpoint v_s, where $s = |\mathcal{D}| - 1 = |\mathcal{E}^{\mathcal{D}}|$. The root node begins by solving its neighbourhood problem P_{v_0}, defined already in (3.5). Since the constraints of the global problem may couple any nodes of

the graph, the entire set of solutions to this problem $\{x^*_{jv_0},\ j \in \mathcal{N}_{v_0}\}$ are then passed forward to the next node in the path. Along the path, each node $i \in \mathcal{D} \setminus \{v_s\}$ has a unique 'child' node $r_i \in \mathcal{D}$ to which it sends messages, and each node $i \in \mathcal{D} \setminus \{v_0\}$ has a unique parent $p_i \in \mathcal{D}$ from which it receives messages.

After P_{v_0}, each subsequent problem $P_{v_1}, \ldots, P_{v_s}$ must incorporate the solutions to all preceding problems. We first define $\mathcal{F}_k$ as the set of all nodes for which solutions have been found in the problems up to and including P_{v_k}, that is,

$$\mathcal{F}_k = \bigcup_{m=0}^{k} \mathcal{S}_{v_m}, \qquad k = 0, 1, \ldots, s. \tag{3.8}$$

To ensure that each successive stage of the recursion is feasible with respect to all previous stages, we can construct the expanded problems $P'_{v_k},\ k = 1, \ldots, s$ as follows:

$$\begin{aligned} \min_{\{x_{jv_k},\, j \in \mathcal{F}_k\}} & \sum_{j \in \mathcal{F}_k} f_j(x_{jv_k}) \\ \text{s.t. } & c^l_{v_k}(\{x_{jv_k},\ j \in \mathcal{F}_k\}) = 0, \quad l \in \mathcal{P}_{v_k} \\ & c^l_{v_k}(\{x_{jv_k},\ j \in \mathcal{F}_k\}) \le 0, \quad l \in \mathcal{Q}_{v_k}, \end{aligned} \tag{3.9}$$

where $\mathcal{P}_{v_k}$ and $\mathcal{Q}_{v_k}$ are the sets of indices for all constraint functions in P'_{v_k}. We note that, if we set $k = 0$ in (3.9), then we obtain the root problem (3.5). As we move along the path, and hence as k increases to s, the size of P'_{v_k} will increase, incorporating decision variables associated with nodes outside the corresponding local neighbourhood $\mathcal{S}_k$. However, by constructing a sweeping procedure in which each path node $v_k,\ k = 0, \ldots, s-1$ sends forward the set of solutions $\{x^*_{jv_k},\ j \in \mathcal{F}_k\}$ to its child $r_{v_k} = v_{k+1}$, the expanded problems $P'_{v_k},\ k = 1, \ldots, s$ can be significantly simplified. This is because the solutions $\{x^*_{jv_{k-1}},\ j \in \mathcal{F}_{k-1}\}$ sent from v_{k-1} to v_k can then be copied at v_k, enabling it to set $\{x^*_{jv_k},\ j \in \mathcal{F}_{k-1}\}$ before even doing any computation. Therefore, the set of decision variables in the problem at v_k can be reduced to $\bar{\mathcal{S}}_{v_k} = \mathcal{N}_{v_k} \setminus \mathcal{F}_{v_{k-1}}$. To construct the simplified problem $P_{v_k},\ k = 1, \ldots, s$, we first make a similar observation to that in (3.6). That is, the set of constraint functions for the corresponding expanded problem P'_{v_k} can be partitioned as follows:

$$\begin{aligned} \Big\{c^l_{v_k}(\{x_{jv_k},\ j \in \mathcal{F}_k\}),\ l \in \mathcal{P}_{v_k} \cup \mathcal{Q}_{v_k}\Big\} = & \Big\{c^l_{v_k}(\{x_{jv_k},\ j \in \mathcal{F}_k\}),\ l \in \bar{\mathcal{P}}_{v_k} \cup \bar{\mathcal{Q}}_{v_k}\Big\} \cup \\ & \Big\{c^l_{v_k}(\{x_{jv_k},\ j \in \mathcal{F}_{k-1}\}),\ l \in \hat{\mathcal{P}}_{v_k} \cup \hat{\mathcal{Q}}_{v_k}\Big\}, \end{aligned} \tag{3.10}$$

where $\hat{\mathcal{P}}_{v_k} \subset \mathcal{P}_{v_k}, \hat{\mathcal{Q}}_{v_k} \subset \mathcal{Q}_{v_k}$ are the sets of indices corresponding to constraint functions dependent only on the set of variables $\{x_{jv_k},\ j \in \mathcal{F}_{k-1}\}$, and we define $\bar{\mathcal{P}}_{v_k} = \mathcal{P}_{v_k} \setminus \hat{\mathcal{P}}_{v_k},\ \bar{\mathcal{Q}}_{v_k} = \mathcal{Q}_{v_k} \setminus \hat{\mathcal{Q}}_{v_k}$. Since the sweep allows each agent v_k on the path to trivially set the solutions $\{x^*_{jv_k},\ j \in \mathcal{F}_{k-1}\}$, the functions $c^l_{v_k}, l \in \hat{\mathcal{P}}_{v_k} \cup \hat{\mathcal{Q}}_{v_k}$ can be evaluated as constants. Thus the simplified problem P_{v_k}, for each $k = 1, \ldots, s$, is given by

$$\begin{aligned} \min_{\{x_{jv_k},\, j \in \bar{\mathcal{S}}_{v_k}\}} & \sum_{j \in \bar{\mathcal{S}}_{v_k}} f_j(x_{jv_k}) \\ \text{s.t. } & c^l_{v_k}(\{x_{jv_k},\ j \in \bar{\mathcal{S}}_{v_k}\}, \{x^*_{jv_k},\ j \in \mathcal{F}_{k-1}\}) = 0, \quad l \in \bar{\mathcal{P}}_{v_k} \\ & c^l_{v_k}(\{x_{jv_k},\ j \in \bar{\mathcal{S}}_{v_k}\}, \{x^*_{jv_k},\ j \in \mathcal{F}_{k-1}\}) \le 0, \quad l \in \bar{\mathcal{Q}}_{v_k}. \end{aligned} \tag{3.11}$$

This reduced problem has significantly fewer variables and constraints than (3.9), and many of the remaining constraints may no longer be coupled. Through (3.5) and (3.11), we now have a mechanism

by which we can build up a set of solutions for the entire graph, implementing the problems sequentially along the path $\mathcal{G}^{\mathcal{D}}$. Once P_{v_s} is solved, a decision for every node $i \in \mathcal{V}$ will have been determined, since the path is induced by a connected dominating set for $\mathcal{G}$, meaning that $\mathcal{F}_s = \mathcal{V}$.

Unlike in CDS-BS, this sweep does not solve a problem at every node $i \in \mathcal{V}$. Therefore, we have to adapt the communication protocol to ensure that each node $i \in \mathcal{V} \setminus \mathcal{D}$ can find out its own decision. This can be done by assigning to each of these non-path nodes a parent $p_i \in \mathcal{D}$, given by

$$p_i = \min\{k \mid v_k \in \mathcal{N}_i\}, \quad i \in \mathcal{V} \setminus \mathcal{D}. \tag{3.12}$$

This definition ensures that, for an ideal system with no communication latency, each agent $i \in \mathcal{V}$ can set its own decision x_i^* as soon as it has been determined in the sweep. Having described the protocol for the entire graph, we can now summarize CDS-PS in Algorithm 3, where each $i \in \mathcal{D}$ is equivalently referred to as v_k, for its corresponding position k in the path.

Algorithm 3 CDS Path Sweep

for $i \in \mathcal{V}$, in parallel **do**
  if $i = v_0$ **then**
    Solve P_{v_0} (3.5) and retain $x^*_{v_0}$.
    For each $j \in \mathcal{N}_{v_0} \setminus \{v_1\}$, send $x^*_{jv_0}$ to node j.
    Send $\{x^*_{jv_0},\ j \in \mathcal{N}_{v_0}\}$ to node v_1.
  else if $i \in \mathcal{D}$ **then**
    Wait until $\{x^*_{jv_{k-1}},\ j \in \mathcal{F}_{k-1}\}$ received from node v_{k-1}.
    Set $x_i^* = x^*_{iv_{k-1}}$.
    Solve P_{v_k} (3.11).
    For each $j \in \bar{\mathcal{S}}_{v_k} \setminus \mathcal{D}$, send $x^*_{jv_k}$ to node j.
    if $k \neq s$ **then**
      Send $\{x^*_{v_k},\ j \in \mathcal{F}_k\}$ to node v_{k+1}.
    end if
  else
    Wait until $x^*_{ip_i}$ received from node p_i.
    Set $x_i^* = x^*_{ip_i}$.
  end if
end for

A visualization of the algorithm is shown in Figure 3.5, using an example graph with $N = 7$ nodes. The designated connected dominating set $\mathcal{D} = \{3, 4, 5\}$, with $v_0 = 3$ and $v_s = 5$. The first stage involves solving P_3 at agent 3. Since $v_1 = 4$, agent 3 then sends $\{x_3^*, x_{23}^*, x_{43}^*\}$ to agent 4, whilst agent 2 receives only x_{23}^*, setting $x_2^* = x_{23}^*$. At the end of this stage, the set of nodes $\{3, 4, 5\}$ have fixed solutions, each of which is stored locally.

In the second stage, agent 4 solves an optimization problem P_4, with the decision variable x_{54} and incorporated constant vectors $x_4^* = x_{43}^*$, $x_{34}^* = x_3^*$ and $x_{24}^* = x_{23}^*$. The structure of the problem ensures that the resulting decision for agent 5 will satisfy the coupling constraints with the set of agents $\{2, 3, 4\}$. This solution x_{54}^* can then be passed forward to agent 5 along with x_4^*, x_{34}^* and x_{24}^*.

In the third stage, agent 5 solves the problem P_5, with decision variables x_{65}, x_{75} and x_{15} and constant vectors represents the solutions of the set of agents $\{2, 3, 4, 5\}$. The resulting minimizers x_{65}^*, x_{75}^* and x_{15}^* can then be sent respectively to agents 6, 7 and 1. Since $v_s = 5$, the algorithm will terminate, with all nodes $i \in \mathcal{V}$ now storing their solution, as shown in the fourth panel of the figure.

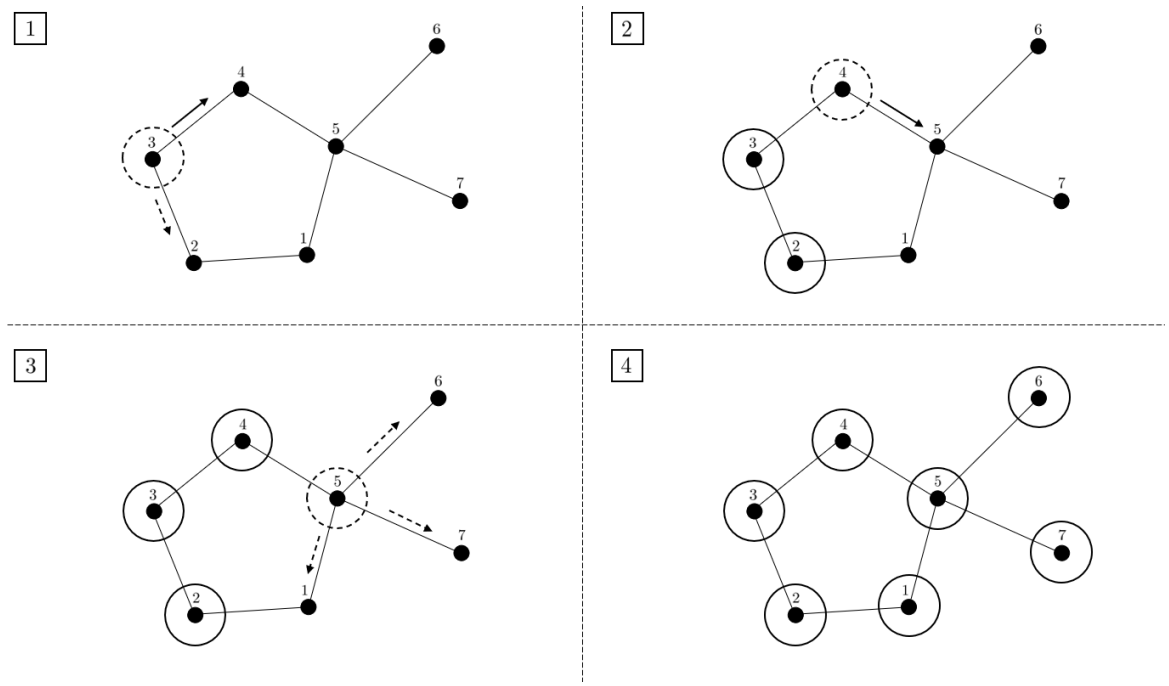


Figure 3.5: A schematic diagram of the branch sweep algorithm over a tree graph. Circles with dashed lines indicate that the enclosed agent is solving an optimization problem at that stage, and circles with solid lines illustrate that the corresponding node is storing its local solution. The solid arrows represent transmission of the accumulated solution set along the CDS path, and dashed arrows indicate the transfer of a single opinion from the solving agent to the relevant neighbour.

Just as in the branch sweep, Algorithm 3 does not require any agent to have full knowledge of the graph structure. However, each node $i \in \mathcal{V} \setminus \{v_0\}$ must know its parent p_i, and the receiver r_i must be known to each solving agent $i \in \mathcal{D} \setminus \{v_s\}$. A notable additional complexity of this algorithm arises from the fact that the sweeping mechanism cannot propagate naturally from the root, as was the case over a tree graph. Instead, the entire path must be set in order for the algorithm to proceed. The problem of searching for connected dominating sets within a graph is one that has received plenty of attention in the fields of graph theory and wireless networks [54], and so we will not delve further into this question here. Instead, we will simply assume that such a search algorithm can be readily implemented.

With the same reasoning as applied to CDS-BS, we note that the path sweep cannot necessarily find an optimal solution if the global problem cannot naturally be expressed in stages. Furthermore, depending on the chosen path, CDS-PS may not be able to find a feasible solution, since agents may be coupled by inequality constraints. However, we demonstrate the effectiveness of the algorithm with an example in Chapter 5, and we can reasonably claim that this procedure provides a possible route to efficiently solving certain problems which may otherwise require many iterations of a more conventional distributed algorithm.

3.3 An External Active Set Method

By building up a set of exact solutions that can be passed through the stages as constant parameters, both CDS-BS and CDS-PS will not succumb so readily to the curse of dimensionality that plagues many dynamic programming recursions. Indeed, by restricting its application to trees with global coupling, successive problems in the branch sweep procedure will not grow drastically in size. However, as the path sweep progresses, it may be more cumbersome to solve problems further along the sequence, even if the number of decision variables does not increase. This is due to the accumulation of decision values

from preceding stages, all of which need to be incorporated into subsequent problems via additional constraint functions. As the number of constraints increases, numerical solving methods can become very slow to implement, since more gradient computations need to be carried out.

We address this problem by applying the external active set (EAS) strategy proposed in [47] to each stage of CDS-PS. The steps of this procedure are summarized in Algorithm 4. The method is based on the idea that the solution to an optimization problem (1.1) with many inequality constraints may equivalently be obtained from a smaller problem with only a fraction of these constraints included. Specifically, since the feasibility at the optimal point is defined only by the equality constraints $c^j(x^*) = 0,\ j \in \mathcal{P}$ and active inequality constraints, $c^j(x^*) = 0,\ j \in \mathcal{A} \subset \mathcal{Q}$, the strategy attempts to infer the elements of the active set $\mathcal{A}$ to construct a smaller set of inequality constraints, ignoring as many functions as possible from the set $\{c^j,\ j \in \mathcal{Q} \setminus \mathcal{A}\}$. This is done by iteratively constructing a problem $P_{\mathcal{W}^k}$, defined as

$$\min_x\ f(x) \qquad \text{s.t.} \quad \begin{cases} c^j(x) = 0, & j \in \mathcal{P} \\ c^j(x) \leq 0, & j \in \mathcal{W}^k, \end{cases} \tag{3.13}$$

where $\mathcal{W}^k \subset \mathcal{Q}$ is the set of indices assumed to be in the active set at the k^{th} iteration of the scheme. Initializing $\mathcal{W}^0 = \varnothing$ the strategy then applies N_{iter} iterations of a recursive numerical solving method of the form $\xi^{l+1} = A(\xi^l)$. Even if a solution is not found before the recursion stops, the value of the decision variable at termination can be used to update $\mathcal{W}^{k+1}$. This update is constructed at each iteration with the function $w(x)$, defined by

$$w(x) = \{j \in Q \mid c^j(x) \geq \gamma(x)\}. \tag{3.14}$$

In [47], $\gamma(x)$ is set as

$$\gamma(x) = \epsilon(x) = \max\{0, \psi_+(x) - \bar{\epsilon}\}, \tag{3.15}$$

where $\bar{\epsilon} > 0$ is a constant scalar parameter, $\psi_+(x)$ is defined by

$$\psi_+(x) = \max\{0, \psi(x)\} \tag{3.16}$$

and $\psi(x)$ is the maximum inequality constraint value,

$$\psi(x) = \max_{j \in Q} c^j(x). \tag{3.17}$$

For reasons that will become clear, we suggest an alternative for $\gamma(x)$,

$$\gamma(x) = \min\{\delta(x), \epsilon(x)\}. \tag{3.18}$$

To define the new function $\delta(x)$, we first introduce the index set $\lambda(x)$ for all violated inequality constraints for a given x:

$$\lambda(x) = \{c^j(x) \mid j \in Q,\ c^j(x) > 0\}. \tag{3.19}$$

We additionally define the recursive function $s_k(A)$ to identify the k^{th} largest element in a set A:

$$s_k(A) = \max\left(A \setminus \{s_l \mid l \in \mathbb{N}_0,\ l < k\}\right), \quad s_0 = \{\}, \tag{3.20}$$

where $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ is the set of non-negative integers. Now we can define

$$\delta(x) = s_{\bar{\delta}|\lambda(x)|}(\lambda(x)), \tag{3.21}$$

where $\bar{\delta} \in [0, 1]$ is a constant scalar parameter. In this way, $w(x^k)$ constructs the set of indices corresponding to inequality constraints which, when evaluated at $x = x^k$, exceed some threshold. The function $\gamma(x^k)$, as stated in (3.18), chooses this threshold as the lower of two values: $\delta(x^k)$ sets the threshold such that all values above $\delta(x^k)$ are in the largest $\eta\%$ of the set of violated constraint values $\lambda(x^k)$, with $\eta = 100\bar{\delta}$. The second value $\epsilon(x^k)$ is $\bar{\epsilon}$ less than the maximum constraint value $\psi_+(x^k)$, or zero if $\psi_+(x^k) < \bar{\epsilon}$. The algorithm constructs the index set $\mathcal{W}$ with the reasonable guess that the most-violating constraints in the recursive numerical scheme are more likely to be active at the eventual solution point. Then $w(x)$ simply states how we select these constraints. The original threshold given by $\epsilon(x)$ will quickly build up $\mathcal{W}$ if $\psi_+(x^k)$ is relatively small at each iteration k. However, if $\psi_+(x^k)$ is very large, as may be the case in the first iteration, then the next largest constraint values may be distributed over a large range, in which case the bound $\bar{\epsilon}$ may fail to 'catch' them until a later iteration. Therefore, by introducing the alternative threshold $\delta(x)$, we can guarantee that a reasonable proportion of most-violated constraints will be included earlier in the scheme, thus potentially reducing the number of iterations of Algorithm 4 required to find a large enough set $\mathcal{W}$.

Algorithm 4 External Active Set Method

Required: x^0, N_{iter}, $w\colon \mathbb{R}^n \to \mathbb{R}_+$.
Set $k = 0$, $\mathcal{W}^0 = \varnothing$.
repeat
  Compute $\mathcal{W}^{k+1} = \mathcal{W}^k \cup w(x^k)$.
  Set $\xi^0 = x^k$.
  Perform N_{iter} iterations $\xi^{l+1} = A(\xi^l)$ on $P_{\mathcal{W}^k}$ (3.13).
  Set $x^{k+1} = \xi^{N_{\text{iter}}}$.
  Compute $\psi(x^{k+1})$ (3.17).
  Set $k = k + 1$.
until x^{k+1} is a local solution of $P_{\mathcal{W}^k}(x)$ and $\psi(x^{k+1}) \leq 0$.

We additionally note that the scheme's performance is dependent on the parameters N_{iter}, $\bar{\delta}$ and $\bar{\epsilon}$, and the best combination of values for a given problem may only be found through extensive numerical testing. Nevertheless, it is reasonable to suggest that this simple procedure can potentially be very effective for CDS-PS, as many of the constraints added at each stage of the sweep may not be critical to the new problem, and so only serve to add to the computation. By removing many of these functions, CDS-PS can better handle problems with extensive global coupling, instead of becoming absurdly slow for larger graphs.

Chapter 4

Implementation

We now briefly describe an approach for implementing the algorithms of the last two chapters, focusing on necessary distributed computing techniques to enable communication between processes. The methods were tested in Julia [55], and so we use this as the basis of our description, however there are a number of other programming languages which can exploit many of the same concepts to good effect. We then discuss the challenges of implementing CDS-BS and CDS-PS in high-latency networks, proposing an improved communication protocol to mitigate delays caused by information exchange between agents.

4.1 Distributed Computing in Julia

Although Message Passing Interface (MPI) [56] is perhaps the most widely used standard for message-passing between parallel processes, and good wrappers for Julia do indeed exist, we instead invoke Julia's native `Distributed` module. Beyond merely exploring the capabilities of a relatively new programming language, we can benefit from the one-sided communication structure inherent to Julia, whereby only one process may require explicit management in a two-process operation.

Multi-processing can be implemented with Julia by first initiating N workers, each responsible for one of the N parallel processes required. Although several workers may be able to run simultaneously on a single central processing unit (CPU), it is more ideal to enforce processor affinity, meaning that each worker is bound to a different CPU. Not only is this more representative of the computing architecture in a network of agents with on-board processing units, but affinity is also known to result in improved performance. For this reason, our distributed algorithms were ran on a cluster of processors, each implementing the work of one agent.

Algorithms 1-3 all require information to be *sent* from an agent and *received* by a neighbour. We instead reframe this strategy as follows: information is *made available* by an agent, and then *collected* by a neighbour. The advantage of this alternative approach is that the collecting agent can have full control in deciding which of the available data to collect, and thus can immediately label the received data in the same way that it would name and assign any local variable.

The function `remotecall_fetch` can be used to fetch data from one process onto another. However, a worker i can only fetch a value from another worker j if that value exists. Furthermore, once this value does exist on worker i, it may subsequently change as the algorithm runs. Therefore we require some mechanism by which the collecting agent waits until the correct data is available before retrieving it. In Julia, this can be done with a `Channel`, essentially a first in, first out queue of data onto which information can be uploaded and from which information can then be retrieved. Crucially, it is possible to delay tasks by first waiting until a channel contains data. Therefore, we can set up

at each sending agent a channel responsible for a particular data transfer, ensuring that the receiving agent can collect the correct data. Furthermore, if an agent requires data from multiple neighbours, it can asynchronously set up a wait at each relevant channel, and so the agent can immediately collect the required data from any source as soon as it has been made available by the sender.

4.2 Real Systems with Communication Latency

In a real network of agents, the transmission of a message between nodes is not an instant process. Instead, various factors may result in significant communication latency, which in turn delays the progress of an algorithm distributed across the network. For a given message between a transmitter and receiver, we define the latency L as the total time from the beginning of transmission to the receipt of the last bit of data. This can be calculated as

$$L = \tau + \frac{b}{s}, \tag{4.1}$$

where τ is the propagation delay, b is the size of the transmitted packet (in bits) and s is the bit rate (in bit/s). The propagation delay is dependent on the physical medium over which the message is passed, and so we cannot affect this in the algorithm design. Similarly, we will treat the bit rate as an intrinsic property of the system. Although s will vary for communication between different agent pairs, since it may be primarily dependent on physical distance between the two agents, we will instead assume a uniform s across all transmissions to simplify the analysis.

This leaves only the size of the message as a possible variable. Our proposed algorithms were designed to decrease the number of messages being passed through a given graph, but beyond this, we cannot reduce the total number of bits being transmitted without either sacrificing accuracy in the vectors or assuming that each node has additional knowledge about the graph structure. However, for both algorithms, the solving agent will pass forward some data which is independent of the optimization problem being solved. In Algorithm 2, an agent i can set $x_i^* = x_{ip_i}^*$ as soon as it receives $x_{ip_i}^*$ from its parent. Once it has solved P_i, the agent then sends x_i^* and x_{ji}^* to each unsolved neighbour j. But x_i^* has not changed since it was first copied from $x_{ip_i}^*$, so agent i can pass x_i^* to each agent $j \in \bar{\mathcal{S}}_i$ before the optimization is complete. If we consider that x_i^* and x_{ji}^* are vectors of the same size, with values stored in the same data type, then each can be described with $\bar{b}$ bits, where $\bar{b}$ depends on the number of values in the vector and the data type of each value. Then, using our initial communication protocol, the total delay in the global algorithm due to the message-passing between agents i and j can be expressed as

$$L_{ij} = \tau + \frac{2\bar{b}}{s}. \tag{4.2}$$

If we instead implement the new protocol, in which x_i^* is passed forward as soon as its value has been assigned by agent i, then, with the reasonable assumption that $\tau + \frac{\bar{b}}{s} \leq T_i$, where T_i is the time taken to solve P_i, we reduce the effective delay on the algorithm to

$$L'_{ij} = \tau + \frac{\bar{b}}{s}. \tag{4.3}$$

Retaining this assumption whilst also assuming that the propagation delay τ is negligible relative to the transmission time, we can estimate the total effective latency $\mathcal{L}_{\mathcal{G}}$ with the amended protocol. For a sweep rooted at the graph centre, we obtain

$$L_{\mathcal{G}} = \frac{r\bar{b}}{s}, \tag{4.4}$$

since the graph radius r is equivalent to the total number of delaying transmissions along the longest path from root to leaf.

The same idea can be applied in the path sweep to even greater effect. Algorithm 3 requires the CDS-induced path to accumulate a growing collection of fixed solutions to be used as constant parameters in subsequent problems. However, this set can expand significantly within very few stages, and so the size of the message passed forward from an agent $i \in \mathcal{D}$ to its child r_i will increase further along the path, causing greater latency between the successive stages. This can be mitigated by relaying fixed solutions through the path as soon as they become available. That is, newly-determined solutions can immediately be passed along through the remaining agents in the path, so that each of these agents will already know a lot of the necessary input values to their problem by the time the sweep reaches them.

To quantify the latency induced by this amended protocol, we consider a global problem over N agents, in which each local variable x_i, $i \in \mathcal{V}$, requires the same number of bits $\bar{b}$ to be described. We assume that the latency in a transmission from any agent $i \in \mathcal{D}$ to its receiver r_i is never greater than the time spent solving the optimization problem at agent i, and that the propagation delay τ is negligible in comparison to the transmission time for a given message. Then, with the relay protocol implemented, the effective delay $L_{\mathcal{G}}$ across the entire path sweep algorithm due to inter-agent communication is limited by the inequality

$$L_{\mathcal{G}} \leq \frac{N\bar{b}}{s}. \tag{4.5}$$

This result can be deduced by realizing that the solving procedure at each agent $i \in \mathcal{D} \setminus \{v_0\}$ is only delayed by the transmission time required for its parent p_i to send any newly-evaluated agent decisions from P_{p_i}. Since each of the N decisions is only found once, each decision only contributes to a critical transmission phase once over the course of the sweep. The last path node v_s may have to distribute new solutions to multiple neighbours, but each of these transmissions will be independent of the others, and so $\frac{N\bar{b}}{s}$ is merely an upper bound on $L_{\mathcal{G}}$.

Although amending Algorithm 3 with the new communication protocol will greatly reduce the adverse effects of latency, we should be cautious of the fact that the resulting expression (4.5) is dependent on the number of nodes in the graph. When N is large, this relation can be approximated to $L_{\mathcal{G}} \approx \frac{N\bar{b}}{s}$, and so the increasing effects of latency as the graph grows in size become unavoidable, regardless of the complexity of the global problem. The same is also true for Algorithm 3, as observed in (4.4). In contrast, a more conventional iterative scheme like ADMM may converge in very few iterations for simple problems, and so the cumulative latency over the course of the algorithm may be relatively small, even if the graph is large. While this is a notable comparison, we can equally argue that the exact nature of the sweeping methods enable them to solve problems which would require a significant number of iterations with alternative schemes. Ultimately, once the effects of a real system are taken into account, there is no such thing as a one-size-fits-all distributed algorithm.

Chapter 5

Numerical Experiments

The proposed algorithms were tested on a pair of dynamic optimization problems (DOPs) for multi-agent systems. A DOP is a specific form of (1.1) in which x represents the trajectory of a dynamical system over some time interval $\mathcal{T}$. Generally, the evolution of such a system can be described with a set of differential equations

$$\dot{x} = f(x, u), \tag{5.1}$$

where the variable u defines a sequence of control inputs over $\mathcal{T}$. Then a DOP can be restated as

$$\min_{x,u} f(x, u) \qquad \text{s.t.} \;\; x \in \mathcal{X}, \; u \in \mathcal{U} \tag{5.2}$$

where the feasible region $\mathcal{X}$ is defined such that the system dynamics (5.1) are always satisfied, along with any other other constraints on the states, while the set $\mathcal{U}$ enforces any limitations on the control inputs. We note that, since (5.2) is defined over a continuous time domain, a DOP must first be converted into a finite-dimensional problem through discretization over $\mathcal{T}$ before we can apply numerical methods to find a solution [57].

The examples presented below consider formations of moving agents, with each agent i governed by the dynamic model

$$\dot{s}_i = v_i, \qquad \dot{v}_i = u_i, \tag{5.3}$$

where $s_i, v_i \in \mathbb{R}^m$ respectively represent the agent's position and velocity vectors for an m-dimensional model, and $u_i \in \mathbb{R}^m$ is the associated control input, equivalent to acceleration. The state of the agent is then defined as

$$x_i = \begin{bmatrix} s_i \\ v_i \end{bmatrix} \in \mathbb{R}^{2m}. \tag{5.4}$$

We discretize the dynamics using a second-order Taylor approximation with a time step of t_s, obtaining a linear model

$$x_i(t+1) = Ax_i(t) + Bu_i(t), \tag{5.5}$$

where the state and input matrices are given by

$$A = \begin{bmatrix} 1 & t_s \\ 0 & 1 \end{bmatrix} \otimes I_m, \qquad B = \begin{bmatrix} \frac{t_s^2}{2} \\ t_s \end{bmatrix} \otimes I_m, \tag{5.6}$$

and t is the discrete-time variable. We now formulate the centralized DOP for this N-agent system. The input sequence will be determined over a discrete-time interval $[0, H-1]$. We treat the system state at $t = 0$ as a known quantity, and so the input sequence enables us to model the evolution of the system over an interval $[0, H]$. Then, constructing the vector $\widetilde{u} \in \mathbb{R}^{NHm}$ as the concatenation of all $u_i(t)$, $i = 1, \ldots, N$, $t \in [0, H-1]$, we define the system objective as

$$f(\widetilde{u}) = \sum_{i=1}^{N} \sum_{t=0}^{H-1} \|u_i(t)\|_2^2. \tag{5.7}$$

This is effectively a measure of total input cost of all agents over $[0, H-1]$.

Both examples presented below have a similar form: given some initial state of the system, find the lowest cost agent trajectories that satisfy some terminal condition at $t = H$, whilst ensuring that no two agents collide within the interval $[0, H]$. Furthermore, limits are imposed on the agents' speeds and inputs. We state the problem as follows:

$$\begin{aligned} \min_{\widetilde{u}} \ & f(\widetilde{u}) \\ \text{s.t.} \ & x_i(t+1) = Ax_i(t) + Bu_i(t) \\ & x_i(0) \in \mathcal{X}_i^0 \\ & x_i(H) \in \mathcal{X}_i^H \\ & \|v_i(t)\|_2 \leq \bar{v} \\ & \|u_i(t)\|_2 \leq \bar{u} \\ & \|s_i(t) - s_j(t)\|_2 \geq r_{\min} \\ & i, j \in \mathcal{V}, \ i \neq j, \ t \in [0, H-1], \end{aligned} \tag{5.8}$$

where $\mathcal{X}_i^0$ and $\mathcal{X}_i^H$ represent the initial and final conditions on the state of agent i, the speed and control limits are set $\bar{v}$ and $\bar{u}$, and $r_{\min}$ defines the minimum allowable distance between any two agents.

In order to solve (5.8) in a distributed fashion, we define the local problem P_i for an agent i with a set of neighbours $\mathcal{N}_i$ and $\mathcal{S}_i = \{i\} \cup \mathcal{N}_i$:

$$\begin{aligned} \min_{\widetilde{u}_i} \ & f_i(\widetilde{u}_i) \\ \text{s.t.} \ & x_j(t+1) = Ax_j(t) + Bu_j(t) \\ & x_j(0) \in \mathcal{X}_j^0 \\ & x_j(H) \in \mathcal{X}_j^H \\ & \|v_j(t)\|_2 \leq \bar{v} \\ & \|u_j(t)\|_2 \leq \bar{u} \\ & \|s_j(t) - s_l(t)\|_2 \geq r_{\min} \\ & j, l \in \mathcal{S}_i, \ j \neq l, \ t \in [0, H-1], \end{aligned} \tag{5.9}$$

where $\widetilde{u}_i$ is formed from all $u_i(t)$, $i \in \mathcal{S}_i$, $t \in [0, H-1]$, and f_i is the neighbourhood objective

$$f_i(\widetilde{u}_i) = \sum_{i \in \mathcal{S}_i} \sum_{t=0}^{H-1} \|u_i(t)\|_2^2. \tag{5.10}$$

We observe that the local optimization problem (5.9) is subject to inter-agent coupling constraints due to the collision avoidance requirement.

5.1 Neighbourhood Coupling

To assess the comparative performance of Algorithms 1 and 2, we consider a one-dimensional N-agent system, i.e., $s_i, v_i \in \mathbb{R}$, for $i = 1, \ldots, N$. The agents move as a platoon, and we define the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ as a path, such that each agent can communicate only with its immediate predecessor and follower. The nodes are numbered such that node N represents the leading agent, and node 1 represents the rear agent. The graph is illustrated in Figure 5.1 for a system with $N = 8$ agents. Although the centralized problem (5.8) imposes coupling constraints between all nodes of the graph, each agent in the platoon is effectively dependent only on its neighbours, and so both neighbourhood consensus ADMM and the CDS branch sweep are suitable.

Figure 5.1: The path graph for a platoon of 8 agents.

We formulate the problem for a time period $T = 100$s, discretized into $H = 1000$ time steps. The initial position and velocity (in SI units) for each agent $i = 1, \ldots, N$ are set as

$$s_i(0) = 8i, \qquad v_i(0) = N + 1 - 2i. \tag{5.11}$$

and the terminal conditions for each agent i are given by

$$s_i(H) = 8N + 500 - 2(N - i), \qquad v_i(H) = 0. \tag{5.12}$$

In other words, at the end of the time period, all agents must be stationary, the lead agent N must be 500m ahead of its starting position, and each pair of consecutive agents must be 2m apart. Furthermore, we set $r_{\min} = 0.5$, $\bar{v} = 10$ and $\bar{u} = 0.2$.

This distributed platoon problem was then used to test the performance of Algorithms 1 and 2, implemented in Julia with the solver Ipopt [58]. For further comparison, the equivalent centralized problem was also solved. Figure 5.2 shows the optimal positions over time for a system of 8 agents, according to the solution of the centralized problem.

Before presenting the comparative study, we confirm that the neighbourhood consensus ADMM algorithm can be applied to this problem by observing the convergence of both r^k and s^k, where we define s^k as the vector constructed by concatenating all non-negative elements of the set $\{c^j(\{x_i^k,\ i \in \mathcal{V}\}) \mid j \in \mathcal{P} \cup \mathcal{Q}\}$. The convergence of these residuals is shown in Figure 5.3 for $N = 8$. We note that the feasibility criterion $\|s^k\|_2 = 0$ terminates the algorithm after only five iterations, whereas

the stopping criterion $\|r^k\|_2 \leq \epsilon$ may result in 20-30 iterations for a reasonable threshold value of the order of 10^{-4}. Then, for all tests, the penalty parameter was set as $\rho = 1$.

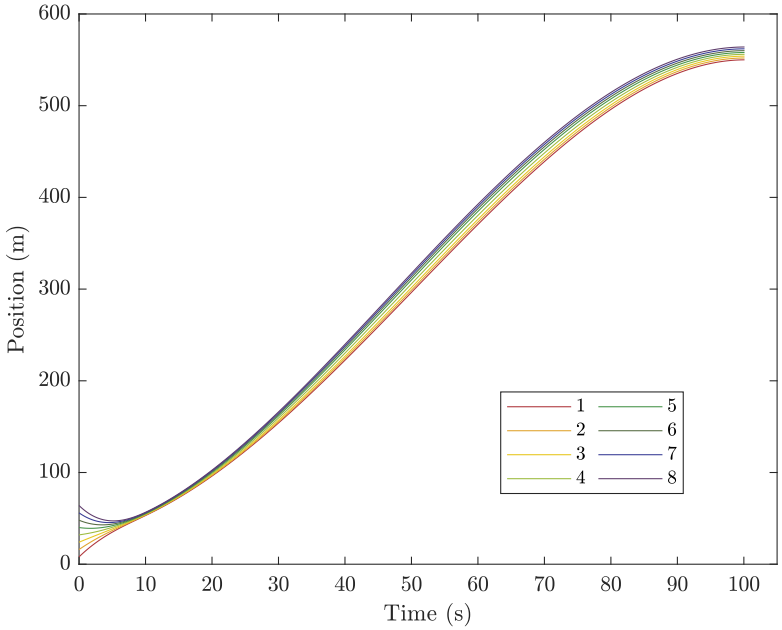
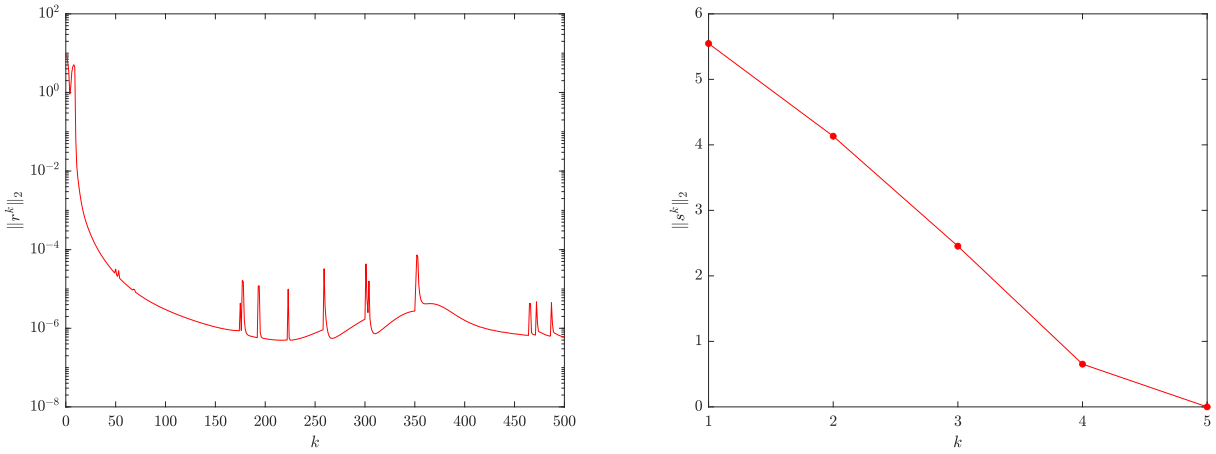


Figure 5.2: A visualization of the platoon problem solution for $N = 8$.



(a) $\|r^k\|_2$ for $k = 1, \ldots, 500$.

(b) $\|s^k\|_2$ for $k = 1, \ldots, 5$, with $\|s^5\|_2 = 0$.

Figure 5.3: The values of (a) $\|r^k\|_2$ and (b) $\|s^k\|_2$ after each iteration k of Algorithm 1.

To assess the effect of graph size on the various approaches, the centralized method, NC-ADMM and CDS-BS were each applied to platoon problems for $N = 4, 8 \ldots, 24$. For CDS-BS, the root node was always set as $v_0 = \frac{N}{2}$. In Figure 5.4 we compare the time required to reach a solution for each method. The latency due to message-passing between processing units has been subtracted from these timings to standardize the results. We observe that CDS-BS finds a solution significantly faster than the other methods, with this superiority becoming more obvious as the number of agents increases, to the point that, for $N = 24$, the elapsed time in CDS-BS is 25.5% of that in the centralized solver, and 39.5% of the time spent running NC-ADMM. By its nature, the time in the branch sweep algorithm will increase only by the time taken to solve any additional reduced problems as the graph increases. For the example of a path graph, we therefore expect the time to increase linearly as we add agents, since all the reduced problems are the same size. Indeed, the figure confirms this trend.

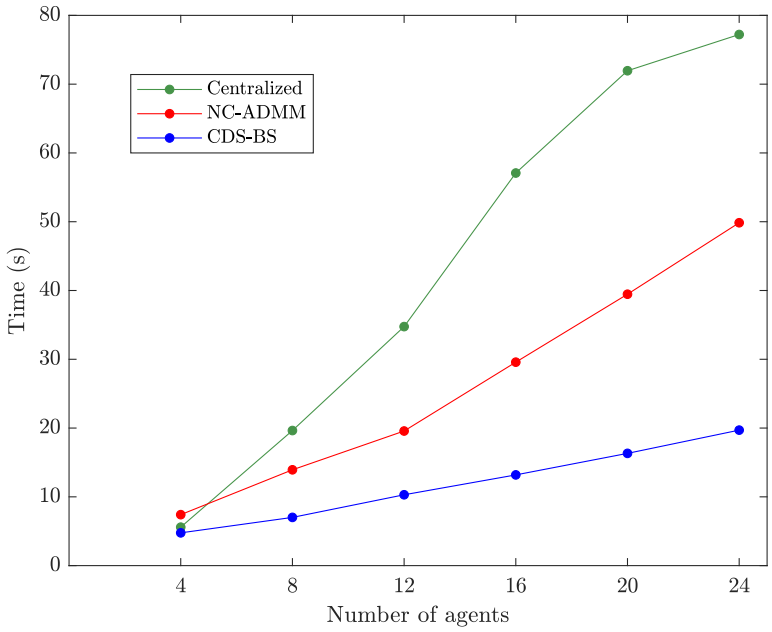


Figure 5.4: Elapsed time in the centralized and distributed schemes, for a range of N.

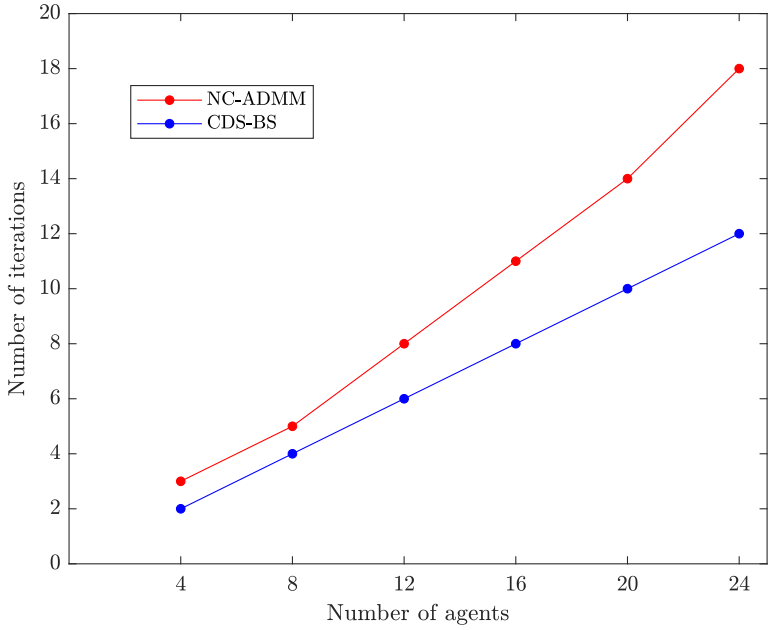


Figure 5.5: Number of iterations required for each distributed algorithm, for a range of N.

Figure 5.5 shows the number of iterations until completion, for each distributed algorithm. By extending a path graph, we naturally induce a linear increase in the number of stages of the branch sweep, since the number of stages is equal to r, the radius of the graph. As explained in the previous chapter, a series of reasonable assumptions enable us to estimate that r is the proportionality constant relating $L_{\mathcal{G}}$ and the transmission time of a single agent's state vector. A similar analysis of NC-ADMM would yield a constant of $2n_k$, where n_k is the total number of iterations, since two messages are passed in each iteration. Considering that, for this example, $r < n_k$ always, it is clear that CDS-BS is a much better candidate that NC-ADMM for high-latency systems.

Finally, in Figure 5.6 we compare the objective value f^* for each distributed algorithm relative to

that of the centralized scheme f_c^*. This demonstrates that the branch sweep algorithm can yield better objective values than NC-ADMM, comparing very well to the centralized solution. Although NC-ADMM would be able to reach better values after more iterations of the algorithm, the additional data transmissions would be a costly sacrifice.

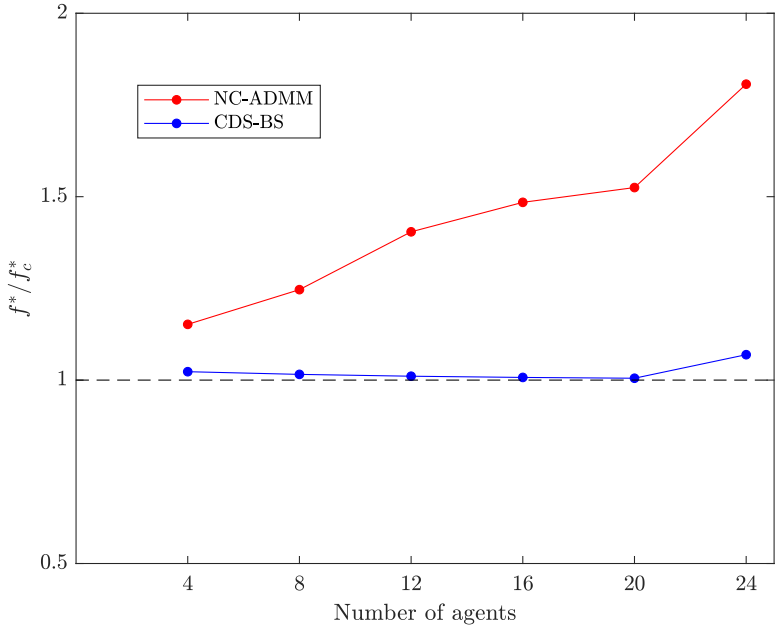


Figure 5.6: The cost f^* for both distributed algorithms relative to the centralized cost f_c^*, plotted for a range of N.

5.2 Global Coupling

The path sweep algorithm (CDS-PS) is assessed by considering a two-dimensional system, i.e., $s_i, v_i \in \mathbb{R}^2$, for $i = 1, \ldots, N$. The set of edges in the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is defined by

$$\mathcal{E} = \{(i,j) \mid \|s_i(0) - s_j(0)\|_2 \leq r_{\text{com}},\ i,j \in \mathcal{V},\ i \neq j\}, \tag{5.13}$$

where r_{com} is the communication radius within which two agents can exchange information. We construct an example for $N = 8$, setting the initial states as

$$\begin{aligned} x_1(0) &= (6.5, 1, 1, 1.5), & x_2(0) &= (5, 1, 1, 1.3), \\ x_3(0) &= (3, 1, 1, 1.2), & x_4(0) &= (1, 1, 1, 1), \\ x_5(0) &= (4, 2, 1, 1), & x_6(0) &= (2, 2, 1, 0.8), \\ x_7(0) &= (2, 3, 1, 0.6), & x_8(0) &= (0, 3, 1, 0.4). \end{aligned} \tag{5.14}$$

These conditions, when applied to (5.13) with $r_{\text{com}} = 2$, result in the graph shown in Figure 5.7. We select as a path the connected dominating set $\{2, 3, 6, 7\}$, starting at $v_0 = 2$. We then define the problem for a time period $T = 50$s, discretized into $H = 500$ time steps. The terminal position conditions for the agents, with $s_i(t) = (s_i^1(t), s_i^2(t))$, are given by

$$\begin{aligned} (s_i^1(H) - 30)^2 + (s_i^2(H) - 30)^2 &\leq 1, \\ \|s_i(H) - s_j(H)\|_2 &\geq r_{\min}, \end{aligned} \tag{5.15}$$

for $i, j \in \mathcal{V},\ i \neq j$. This means that, at $t = H$, all of the agents must be within a circle of radius 1, centred at the point $(30, 30)$. Furthermore, the collision avoidance constraints must still be satisfied. The terminal velocity condition is set as

$$v_i(H) = (1, 1.5), \qquad i \in \mathcal{V}, \tag{5.16}$$

ensuring a uniform velocity among the agents. We complete the problem formulation by setting $r_{\min} = 0.2$, $\bar{v} = 2$ and $\bar{u} = 0.2$.

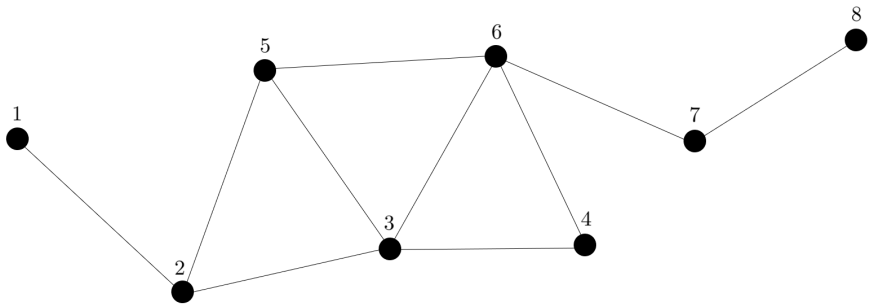


Figure 5.7: The graph for the two-dimensional example.

This problem was solved both in a centralized fashion and with CDS-PS. Figure 5.8 shows the optimal trajectories for the agents, as determined by the path sweep. Tests were also performed on this algorithm with the inclusion of the external active set strategy at each stage. The EAS parameters were fixed as $\bar{\epsilon} = 5$, $\bar{\delta} = 0.3$ and $N_{\text{iter}} = 100$. Although this iteration limit was deemed appropriate after some trial-and-error, it was observed that the strategy performed well for a range of $\bar{\epsilon}$ and $\bar{\delta}$ values. This robustness can be attributed to the fact that these two constants guarantee that the set $\mathcal{W}$ can be constructed in only a few updates. In contrast, defining the threshold $\gamma(x^k)$ with only $\epsilon(x^k)$, as suggested in [47], is not as suitable if there is a large range of constraint violation values at each stage of the scheme. This was verified by observing that, without the addition of the alternative threshold $\delta(x^k)$, the performance of EAS was very sensitive to the value of $\bar{\epsilon}$.

In Table 5.1, we present the comparative performance of these three methods, assessed by objective value f^* and elapsed time. The CDS-PS solution is very close to that of the centralized method, but the algorithm requires slightly longer to run. In this particular example, we might expect our algorithm to be relatively slow, since it is solving a sequence of four problems over a graph with only eight nodes. Naturally, we expect that, for larger graphs with $\frac{|\mathcal{D}|}{|\mathcal{V}|} \ll 1$, the sweep will become much more efficient than the centralized scheme.

Algorithm	f^*	Time (s)
Centralized	13.277	160.4
CDS-PS	13.631	199.0
CDS-PS, EAS	13.294	24.40

Table 5.1: Performance comparison between the centralized method, CDS-PS, and CDS-PS with the external active set strategy, applied to the 2-D multi-agent system example problem.

The most notable result from these tests is the drastic acceleration of CDS-PS when combined with the external active set strategy. The performance of EAS at each stage is detailed in Table 5.2, where n_k is the number of iterations required to terminate the scheme, $\mathcal{Q}$ is the index set of inequality

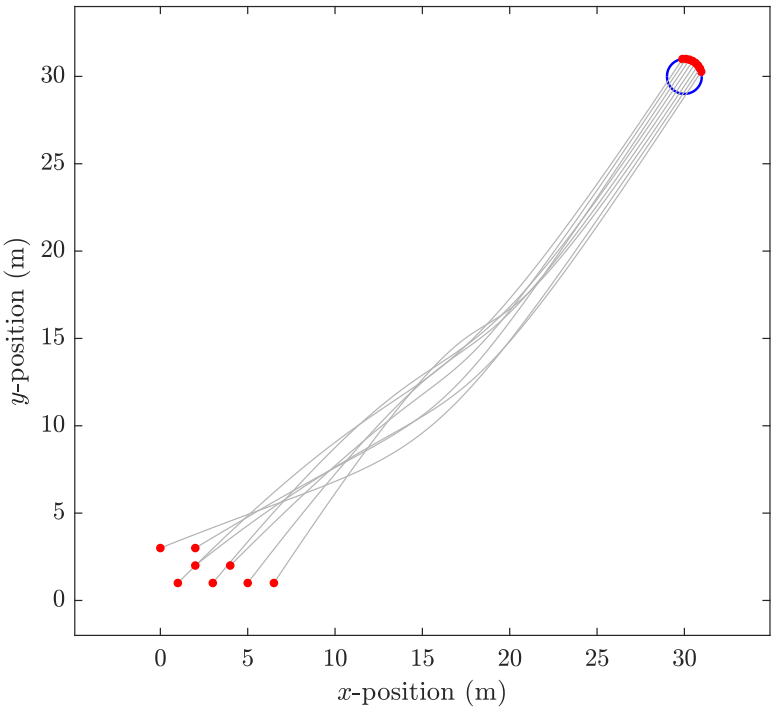


Figure 5.8: Optimal trajectories for the two-dimensional example, with the circular boundary indicated in blue.

constraints in the problem at each stage (for a path node v_k, this set is equivalent to $\bar{\mathcal{Q}}_{v_k}$ in (3.11)), $\mathcal{W} \subset \mathcal{Q}$ is the index set for all inequality constraints ultimately included in the numerical scheme, and $\mathcal{A} \subset \mathcal{Q}$ is the index set of all inequality constraints that are active at the solution point. Additionally, t is the time required to solve each problem with EAS incorporated, and we compare this to t_c, the time required to solve the same problem in its original form.

Node	n_k	$\lvert\mathcal{Q}\rvert$	$\lvert\mathcal{W}\rvert$	$\lvert\mathcal{A}\rvert$	t(s)	$\%t_c$
2	3	7010	331	21	7.04	5.86
3	4	6511	515	53	7.68	17.7
6	5	4007	390	9	5.45	32.7
7	4	4508	172	4	4.08	22.3

Table 5.2: Performance characteristics of the external active set strategy, applied to the problem at each path node.

This table shows that, for the problem at each stage of the sweep, $\frac{|\mathcal{A}|}{|\mathcal{Q}|} \ll 1$, confirming that these problems are well-suited to the external active set strategy. We observe that, at all path nodes, the set $\mathcal{W}$ ensures only a small fraction of the inequality constraints are included, thus greatly reducing the number of computations required in each recursion of the solver Ipopt. Nevertheless, we should also note that $|\mathcal{W}| \gg |\mathcal{A}|$ in all cases, suggesting that the scheme could be refined further. However, this excess is merely a consequence of choosing relatively large parameters $\bar{\epsilon}$ and $\bar{\delta}$. We could of course use smaller values to construct $\mathcal{W}$ more cautiously, but this would then result in more iterations of EAS before the stopping criteria are satisfied. Since each iteration may involve as many as 100 recursions in Ipopt, better performance can be obtained by prioritizing a faster, if slightly more reckless, construction of $\mathcal{W}$.

Chapter 6

Conclusion

We have proposed a pair of decentralized optimization schemes over graphs, CDS-BS and CDS-PS, based on the simple idea of passing solutions between nodes as exact parameters, and then sequentially solving simplified problems only within a connected dominating set of the graph. Although convergence guarantees have not been defined, the algorithms can potentially yield much better performance than conventional distributed schemes. Our strategies are not only applicable for both convex and non-convex problems, but furthermore, provided that each stage of the sweep is feasible, both CDS-BS and CDS-PS can obtain solutions to the global problem with significantly less intermediate communication than may be required in consensus-based strategies. We have reasoned that this is particularly beneficial in high-latency multi-agent systems. This practical advantage is in contrast to many other (perhaps more theoretically rigorous) distributed schemes, in which convergence guarantees are often sought without consideration for the number of iterations required.

The proposals discussed in this thesis have been likened to well-established dynamic programming methods, however our strategies construct fewer stages over a given graph by only solving problems in a subset of all nodes. By passing parameters along the sequence instead of functions, we avoid many of the adverse effects of the curse of dimensionality. In addition, by incorporating an external active set strategy at each stage of the path sweep, we can further suppress the curse by greatly simplifying stages over a possibly high-dimensional global problem.

To assess our suppositions, we tested the algorithms in Julia, implementing the necessary communication protocol over a distributed set of processors. For the dynamic optimization examples considered, we observed impressive performance in both CDS-BS and CDS-PS, demonstrating good speed with limited inter-agent communication. Additionally, the external active set strategy was shown to result in significant acceleration of the path sweep, thus adding support to our hypothesis that this method can effectively handle global coupling without introducing excessive complexity to each local problem.

Future Research

Future research should attempt to develop more formal conditions for the feasibility and optimality of the two sweeping schemes. Although such conditions may depend on the nature of the global problem, it would also be useful to formulate in greater detail the effect of the graph structure on performance. A clear understanding of this relationship may enable the construction of more effective strategies for choosing the root node in CDS-BS and the path in CDS-PS. Assuming that feasibility and optimality conditions can be characterized, we should then investigate possible improvements to the algorithms to expand the class of applicable problems. Possible avenues to explore could include starting at multiple root nodes, or developing strategies to infer the best route for the sweep, based on assessing conditions at each stage and then proceeding to the most suitable neighbour.

Bibliography

[1] S. Sra, S. Nowozin, and S. J. Wright. *Optimization for Machine Learning*. MIT Press, 2012.

[2] F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal Control*. John Wiley & Sons, 2012.

[3] J. Mattingley and S. Boyd. Real-time convex optimization in signal processing. *IEEE Signal Processing Magazine*, 27(3):50–61, 2010.

[4] K. Wainwright. *Fundamental Methods of Mathematical Economics*. McGraw-Hill/Irwin, 2005.

[5] R. L. Rardin. *Optimization in Operations Research*. Prentice Hall, 1998.

[6] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl. *Model Predictive Control: Theory Computation, and Design, 2nd Edition*. Nob Hill Publishing, 2017.

[7] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[8] M. L. Darby and M. Nikolaou. MPC: Current practice and challenges. *Control Engineering Practice*, 20(4):328–342, 2012.

[9] A. Afram and F. Janabi-Sharifi. Theory and applications of HVAC control systems - a review of model predictive control (MPC). *Building and Environment*, 72:343–355, 2014.

[10] T. Chang, A. Nedić, and A. Scaglione. Distributed constrained optimization by consensus-based primal-dual perturbation method. *IEEE Transactions on Automatic Control*, 59(6):1524–1538, 2014.

[11] S. Bhattacharya, V. Kumar, and M. Likhachev. Distributed optimization with pairwise constraints and its application to multi-robot path planning. In *Robotics: Science and Systems VI*, pages 87–94, 2010.

[12] H. Everett III. Generalized Lagrange multiplier method for solving problems of optimum allocation resources. *Operations Research*, 11(3):399–417, 1963.

[13] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2$^{\text{nd}}$ edition, 1999.

[14] P. Giselsson and A. Rantzer. Distributed model predictive control with suboptimality and stability guarantees. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC)*, pages 7272–7277, 2010.

[15] P. Giselsson, M. Dang Doan, T. Keviczky, B. De Schutter, and A. Rantzer. Accelerated gradient methods and dual decomposition in distributed model predictive control. *Automatica*, 49(3):829–833, 2013.

[16] B. Yang and M. Johansson. Distributed optimization and games: A tutorial overview. In A. Bemporad, M. Heemels, and M. Johansson, editors, *Networked Control Systems*, chapter 4. Springer-Verlag, 2010.

[17] A. Rantzer. Dynamic dual decomposition for distributed control. In *Proceedings of the 2009 American Control Conference*, pages 884–888, 2009.

[18] A. Falsone, K. Margellos, S. Garatti, and Maria Prandini. Dual decomposition for multi-agent distributed optimization with coupling constraints. *Automatica*, 84:149–158, 2017.

[19] H. J. Ferreau, A. Kozma, and M. Diehl. A parallel active-set strategy to solve sparse parametric quadratic programs arising in MPC. In *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference*, 2012.

[20] J. V. Frasch, S. Sager, and M. Diehl. A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computation*, 7(3):289–329, 2015.

[21] A. Kozma, J. V. Frasch, and M. Diehl. A distributed method for convex quadratic programming problems arising in optimal control of distributed systems. In *Proceedings of the 52nd IEEE Conference on Decision and Control*, pages 1526–1531, 2013.

[22] I. Necoara and J. A. K. Suykens. Interior-point Lagrangian decomposition method for separable convex optimization problems. *Journal of Optimization Theory and Applications*, 143:567–588, 2009.

[23] M. Kallio and A. Ruszczyński. Perturbation methods for saddle point computation. *International Institute for Applied Systems Analysis*, 1994.

[24] M. Kallio and C. H. Rosa. Large-scale convex optimization via saddle point computation. *Operations Research*, 47(1), 1999.

[25] R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité, d'une classe de problèmes de Dirichlet non linéaires. *Revue Française d'Automatique, Informatique et Recherche Opérationnelle*, 9:41–76, 1975.

[26] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers and Mathematics with Applications*, 2(1):17–40, 1976.

[27] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3:1–122, 2011.

[28] M. Hong, Z. Q. Luo, and M. Razaviyayn. Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM Journal on Optimization*, 26(1):337–364, 2016.

[29] Y. Yang, J. Sun, H. Li, and Z. Xu. Deep ADMM-net for compressive sensing MRI. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 10–18, 2016.

[30] W. Deng, M. J. Lai, Z. Peng, and W. Yin. Parallel multi-block ADMM with o(1/k) convergence. *Journal of Scientific Computing*, 71(2):712–736, 2017.

[31] T. Erseghe. Distributed optimal power flow using ADMM. *IEEE Transactions on Power Systems*, 29(5):2370–2380, 2014.

[32] S. H. Chan, X. Wang, and O. A. Elgendy. Plug-and-play ADMM for image restoration: Fixed-point convergence and applications. *IEEE Transactions on Computational Imaging*, 3(1):84–98, 2016.

[33] B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang. An ADMM algorithm for a class of total variation regularized estimation problems. *IFAC Proceedings Volumes*, 45(16):83–88, 2012.

[34] C. He, L. Wu, T. Liu, and M. Shahidehpour. Robust co-optimization scheduling of electricity and natural gas systems via ADMM. *IEEE Transactions on Sustainable Energy*, 8(2):658–670, 2016.

[35] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin. On the linear convergence of the ADMM in decentralized consensus optimization. *IEEE Transactions on Signal Processing*, 62(7):1750–1761, 2014.

[36] K. Huang and N. D. Sidiropoulos. Consensus-ADMM for general quadratically constrained quadratic programming. *IEEE Transactions on Signal Processing*, 64(20):5297–5310, 2016.

[37] T. H. Summers and J. Lygeros. Distributed model predictive consensus via the alternating direction method of multipliers. In *Proceedings of the 50th Annual Allerton Conference on Communication, Control and Computing*, pages 79–84, 2012.

[38] T. Chang, M. Hong, and X. Wang. Multi-agent distributed optimization via inexact consensus ADMM. *IEEE Transactions on Signal Processing*, 63(2):482–497, 2015.

[39] Y. Lyu, J. Hu, B. M. Chen, C. Zhao, and Q. Pan. Multivehicle flocking with collision avoidance via distributed model predictive control. *IEEE Transactions on Cybernetics*, 51(5):2651–2662, 2021.

[40] S. Koehler, C. Danielson, and F. Borrelli. A primal-dual active-set method for distributed model predictive control. *Optimal Control Applications and Methods*, 38:399–419, 2017.

[41] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[42] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[43] Y. Jiang, D. Kouzoupis, H. Yin, M. Diehl, and B. Houska. Decentralized optimization over tree graphs. *Journal of Optimization Theory and Applications*, 189:384–407, 2021.

[44] B. Houska, J. Frasch, and M. Diehl. An augmented Lagragian based algorithm for distributed nonconvex optimization. *SIAM Journal on Optimization*, 26(2):1101–1127, 2016.

[45] N. Chatzipanagiotis and M. M. Zavlanos. On the convergence of a distributed augmented Lagrangian method for non-convex optimization. *IEEE Transactions on Automatic Control*, 62(9):4405–4420, 2017.

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

[47] H. Chung, E. Polak, and S. Sastry. On the use of outer approximations as an external active set strategy. *Journal of Optimization Theory and Applications*, 146(1):51–75, 2010.

[48] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science & Business Media, 2nd edition, 2006.

[49] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[50] A. Richards and J. P. How. Model predictive control of vehicle maneuvers with guaranteed completion time and robust feasibility. In *Proceedings of the 2003 American Control Conference*, pages 4034–4040, 2003.

[51] S. Dreyfus. Richard Bellman on the birth of dynamic programming. *Operations Research*, 50(1):48–51, 2002.

[52] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 3rd edition, 2007.

[53] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons, 2007.

[54] J. Wu and H. Li. On calculating connected dominating sets for efficient routing in ad hoc wireless networks. In *Proceedings of the International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication*, pages 7–14, 1999.

[55] J.Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[56] The MPI Forum. MPI: A message passing interface. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 878–883, 1993.

[57] E. C. Kerrigan, Y. Nie, O. Faqir, C. H. Kennedy, S. A. Niederer, J. A. Solis-Lemus, P. Vincent, and S. E. Williams. Direct transcription for dynamic optimization: A tutorial with a case study on dual-patient ventilation during the COVID-19 pandemic. In *Proceedings of the 59th IEEE Conference on Decision and Control*, pages 2597–2614, 2020.

[58] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.