Privacy preserving car-parking: a distributed approach

ELISABETTA ALFONSETTI

KTH Electrical Engineering

Degree project in
Automatic Control
Master's Thesis
Stockholm, Sweden, January 2013

XR-EE-RT 2013:002

Abstract

There has been a substantial interest recently in privacy preserving problems in various application domains, including data publishing, data mining, classification, secret voting, private querying of database, playing mental poker, and many others. The main constraint is that entities involved in the system are unwilling to reveal the data they hold or make them public. However, they may want to collaborate and find the solution of a bigger computational problem without revealing the privately held data. There are several approaches for addressing such issues, including cryptographic methods, transformation methods, and parallel and distributed computation techniques. In this thesis, these three methods are highlighted and a greater emphasis is placed on the last one. In particular, we discuss the theoretical backgrounds of optimization decomposition techniques. We further point out key literature associated with the privacy preserving problems and provide basic classifications of their treatments. We focus to a particular interesting application, namely the car parking problem, or parking slot assignment problem. To solve the problem in a privacy preserving manner, a new parallel and distributed computation method is proposed. The goal is to allocate the parking slots to the cars, but without revealing anyone else the intended destinations. We apply decomposition techniques together with projected subgradient method to address this problem and the result is a decentralized privacy preserving car parking algorithm. We compare our algorithm with three other methods and numerically evaluate the performance of the proposed algorithm, in terms of optimality and as well as the computational speed. Despite the reduced computational complexity of the proposed algorithm, it provides close-to-optimal performance.

Acknowledgments

I would like to thank my supervisor, Dr. Chathuranga Weeraddana, for having followed me step by step in the preparation of this thesis. He really helped me and I have learned a lot from him. I would also like to thank Professor Carlo Fischione for giving me the privilege to write my thesis at the Automatic Control Laboratory of KTH. Another thanks to all the PhDs in the department, in particular to Piergiuseppe Di Marco and Damiano Varagnolo, for their advices and for their support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Optimization problems are very common in the business world and the development of solution methods for these problems are of crucial importance from a theoretical, as well as from a practical perspective. Much of the data involved in the optimization problem is constrained by privacy and security concern, preventing the sharing and centralization of data needed to apply optimization techniques. Some potential applications in which the concept of privacy preserving is very important are for example the electronic voting, the scientific and statistical computations, e-commerce, auctions, privacy preserving data mining and many others. So, one of the complicating factors of this area is that we have together two different fields: security and optimization. It is not common for researchers to have expertise in both of these areas. A potential approach for solving these types of problems is to make sure that the parties involved can cooperate with each other to reduce waste and improve efficiency. In the field of business, for example, corporations could have mutual gain with the sharing of some information, like reducing logistic costs. But, generally, companies are unwilling to share their sensitive information for fear of revealing company secrets, their financial strategies or their financial health, breaching privacy or anti-trust legislation. As a result, the collaborative optimization is very difficult to implement. A compromise solution could be to introduce a *trusted third party*.

A trusty third party is an external agent to whom you entrust all your private data. From this type of approaches however there can be various problems. In particular, they must ensure that the data storage system of the third party is secure and they must trust the third party to behave fairly. Also, this type of approach can be costly and there is not guarantee to optimality. Therefore, the use of the third party is not a good idea when there are private information between competitors. However there are some existing solutions to handle such barriers, while exploiting the collaborative bene-

fits of the cooperation between entities. The main classification for privacy preserving methods is as follows:

- **cryptographic methods**: hide the private data by using cryptographic techniques. At each step in the algorithm the data is encrypted/decrypted and the concept of privacy is extrinsically acquired;
- **transformation based methods**: "disguise" the original problem using cryptographic sub-protocols and then solve the problem in the disguised domain. Encryption data happens only once, at the beginning and again privacy is extrinsically acquired.
- **decomposition methods**: the private data can be partitioned between the agents and they can collaborate to solve the problem in a parallel and distributed fashion, without reveal their private variables. Here the privacy is intrinsic.

We will deepen these approaches below, highlighting the advantages and disadvantages for each of them. We will focus mainly on the decomposition technique, which works better under different points of view.

Contribution

We believe that the privacy preserving optimization based approaches are still to be investigated and can be tuned to many application domains. They possess many appealing aspects, which are desirable in practice, e.g., efficiency, scalability, natural (geographical) distribution of problem data. Therefore, in this thesis, we restrict ourselves to privacy preserving optimization based solution methods and do not consider the treatments based on well investigated cryptographic primitives. Of course, there are many survey papers, e.g. [25], that describe in details such methodologies and are extraneous to the main focus. In Section 2, we present main background in optimization theory and in Section 3, we discuss in detail dual decomposition in optimization. In Section 4, we point out key literature associated with the privacy preserving problems and provide basic classifications of their treatments. In Section 5, we consider a specific problem, car parking problem and provide a decentralized method to solve the problem in a *privacy preserving manner*, where we attempt to minimize distance between cars' destinations and the parking slots without revealing the destination information. Section 6 provides numerical results to evaluate the performance of the proposed solution method and Section 7 is the conclusions.

Chapter 2

Optimization Theory

In this chapter we present main background in optimization theory. The origin of the material presented in this chapter is based on reference [1] and we reproduce them for the completeness.

2.1 Mathematical optimization

A mathematical optimization problem has the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i,\ i = 1, \ldots, m \end{array} \tag{2.1}$$

Here the vector $x = (x_1, ..., x_n)$ is the optimation variable of the problem, the function $f_0 : R^n \to R$ is the objective function, the functions $f_i : R^n \to R, i = 1, ..., m$, are the (inequality) constraint functions, and the constants $b_1, ..., b_m$ are the limits, or bounds, for the constraints. A vector x^* is called optimal if for any z with $f_i(z) \leq b_i$, we have $f_0 \geq f_0(x^*)$. The variable x represents the choice made; the constraints $f_i(x) \leq b_i$ represent firm requirements or specifications that limit the possible choices, and the objective value $f_0(x)$ represents the cost of choosing x. A solution of the optimization problem (2.1.1) corresponds to a choice that has minimum cost (or maximum utility), among all choices that meet the firm requirements. Variety of practical problems involving decision making can be cast in the form of a mathematical optimization problem. Mathematical optimization is an important tool in many area such as engineering, electronic design, automation, automatic control systems, civil, chemical, mechanical, and aerospace engineering, network design and operation, finance, supply chain management, scheduling, and many others.

2.2 Convex optimization

The convex problem should satisfy the following requirements:

- the objective function must be convex;
- the inequality constraint functions must be convex;
- the equality constraint functions $h_i(x) = a_i^T x - bi$ must be affine.

Thus we have the problem of the form:

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0,\ i = 1, \ldots, m \\ & a_i^T = b_i,\ i = 1, \ldots, p\ , \end{array} \tag{2.2}$$

where the variable is x and $f_0, ..., f_m$ are convex functions. Here we minimize a convex objective function over a convex set. A fundamental property of convex optimization problems is that any locally optimal point is also globally optimal. Therefore, by using local information one can determine whether a point is locally (and therefore globally) optimal or not. Moreover, there is a rich theory for characterizing the optimality for convex optimization problems, i.e., necessary and sufficient conditions for the optimality. For example, in the case of an unconstrained convex problem $x^* = \arg\min_x f(x)$, if and only if $\nabla f(x^*) = 0$, where ∇f denotes the gradient of the function f. We refer the reader to [1, Chapter 4-5] for details.

2.2.1 Linear optimization

Linear programmin (LP) is the classical example for an convex problem. When the objective and constraint functions are all affine, the problem is called a linear program (LP) . A general linear program has the form

$$\begin{array}{ll} \text{minimize} & c^T x + d \\ \text{subject to} & Gx \preceq h \\ & Ax = b, \end{array} \tag{2.3}$$

where the variable is x and the problem data $G \in R^{m \times n}$ and $A \in R^{p \times n}$. Linear programs are, of course, convex optimization problems. It is common to omit the constant d in the objective function, since it does not affect the optimal (or feasible) set. Since we can maximize an affine objective $c^T x + d$, by minimizing $-c^T x - d$, a maximization problem with affine objective and constraint functions are again an LP. Note that the feasible set of the LP (2.2.2) is a polyhedron. Therefore, the problem is to minimize the affine function $c^T x + d$ over the polyhedron.

2.3 Convex optimization algorithms

Convexity of the problem allows efficient solution methods for convex problems, which is not often the case for non-convex problems. There are many cases where we can use the optimality conditions to achieve closed form solution. Otherwise there are several well studied iterative algorithms to find the solution within a specified tolerances. The rest of this section presents an overview of some important methods.

2.3.1 Descent methods

Recall that any local optimum is also globally optimal for convex problems. Therefore,one strategy is to rely on local information and generate a sequence of points with decreasing cost function value. Such algorithms are called descent methods. As long as the cost function is bounded from below, any such sequence will converge to an optimal point. One natural local information is the descent direction, which is obtained from the gradient of the cost function. The resulting algorithms are called gradient methods. One special variant is the steepest descent method, where the descent direction is found with respect to a given norm $||\cdot||$. These descent methods are usually applied for unconstrained convex optimization problems.

2.3.2 Gradient and subgradient methods

The gradient methods are applied whenever the function f_0 we want to minimize is differentiable. Otherwise, the corresponding algorithm is called the subgradient method. Let us first focus to the differentiable case.

Formally, we call a direction s is descent direction at x, if it fulfills the following

$$\nabla f_0(x)^T s < 0 \ . \tag{2.4}$$

That is, it must make an acute angle with the negative gradient. It is possible to find a step size t, which is sufficiently small and positive so that $f_0(x+ts) < f_0(x)$, which in turn yields a descent method.

Obviously, if $s = -\nabla f_0(x)$, then we have $\nabla f_0(x)^T s = -||\nabla f_0(x)||_2^2 < 0$, and therefore a natural choice for s is the negative gradient of function f_0 evaluated at x. The decent algorithm with this choice is called the gradient method. The $(k+1)$th iteration of the gradient algorithm is given by

$$x(k+1) = x(k) - t(k)\nabla f_0(x(k)) \ , \tag{2.5}$$

where $t(k)$ is the step size that should be chosen appropriately. We refer the reader to [1, Section 9.2] for details.

When the problem is constrained, e.g., $x \in X$, where X is a closed convex set, the gradient algorithm (2.5) can easily be modified as follows:

$$x(k+1) = P_X[x(k) + t(k)\nabla f_0(x(k))] \ , \tag{2.6}$$

where $P_X[x_0]$ represents the projection of point x_0 on to the set X. The orthogonal projection of a point x_0 on **to** X can be formally expressed as

$$P_X[x_0] = \arg\min_{z \in X} \| \ z - x_0 \ \|_2 \ , \tag{2.7}$$

where $\| \cdot \|_2$ is the Euclidian norm. The resulting algorithm (2.6) is called projected gradient method.

Now consider the case where the cost function f_0 is not differentiable. Then, the corresponding alternative is to use a **subgradient** instead of the gradient $\nabla f_0(x)$. A subgradient of a convex (possibly) non-differentiable function f_0 at x is formally defined as follows:

Definition 1 (Subgradient) *We say a vector $s \in R^n$ is a subgradient of convex function $f_0 : R^n \to R$ at $x \in \boldsymbol{dom} f_0$ if for all $z \in \boldsymbol{dom} f_0$,*

$$f_0(z) \geq f_0(x) + s^T(z - x). \tag{2.8}$$

The set of subgradients of f_0 at the point x is called the subdifferential of f at x, and is denoted $\partial f(x)$ and can be defined formally as follows:

Definition 2 (Subdifferential)

$$\partial f_0(x) = \bigcap_{z \in \boldsymbol{dom} f_0} \{s \mid f_0(z) \geq f_0(x) + s^T(z - x)\}. \tag{2.9}$$

The subdifferential $\partial f_0(x)$ is always a closed convex set, even if f_0 is not convex. This follows from the fact that it is the intersection of an infinite set of halfspaces. If f_0 is differentiable, then subdifferential is a singleton and we have $\nabla f_0(x) \in \partial f_0$.

The subgradient algorithm is identical to the **gradient method (2.5)** and the projected subgradient is identical to the projected gradient method (2.6), except that the gradient $\nabla f_0(x)$ is replaced by a subgradient s. However, contrary to the gradient algorithm, the cost function will not necessarily decrease for every iteration. Here, the arguments for convergence are instead based on the decrease of the distance between the iterate and the optimal solution.

2.3.3 Interior point algorithms

In the case of constrained convex optimization problems we have to rely on algorithms based on interior point methods. Interior point methods solve constrained convex problems by considering a sequence of unconstrained problems, where infeasibility translates to a penalty in the objective function. These subproblems are typically solved with Newton methods, and their optimal point is used as an initial point for the next iteration. For every step an approximation of the optimal point is achieved, and the as approximation becomes better, the solutions will converge to the optimal point of the original problem. For every iterate along the way, sub-optimality can be bounded with duality techniques. For a comprehensive treatment, we refer the reader to [1, Chapter 11]. Interior point methods are not as straightforward to implement, but their fast convergence makes them the method of choice for centralized optimization, where ready-made solvers exist, e.g., CVX (http://cvxr.com/cvx/).

2.4 Duality

Duality is a powerful machinery in optimization theory. Duality plays a major role in characterizing optimality of the solution, e.g., Karush Kuhn Tucker (KKT) conditions. One can use duality theory for computing a lower bound on the optimal value of the primal problem, even when it is nonconvex. In the case of convex problem, often the bound on the primal optimal value is tight, provided certain constraint qualifications holds (this known as zero duality gap). Moreover, duality sometimes leads to efficient or distributed method for solving the primal problem. In this chapter we briefly present some basic results of duality. The reader may refer to [1, Section 5] for details.

Throughout this section we will consider the problem of the canonical form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0,\ i = 1, \ldots, m \\ & h_i(x) = 0,\ i = 1, \ldots, p\ , \end{array} \tag{2.4.0.10}$$

where the variable is x.

2.4.1 The Lagrangian and Dual Function

The Lagrangian associated with problem (2.4.0.10) is defined as

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x). \tag{2.4.1.1}$$

where the new variables $\lambda \geq 0 \in R^m$ and $\nu \in R^p$ are called *Lagrange multipliers* or *dual variables.*

Let $f(x) = (f_1(x), \ldots, f_m(x))$, $h(x) = (h_1(x), \ldots, h_p(x))$, $\lambda = (\lambda_1, \ldots, \lambda_m)$, and $\nu = (\nu_1, \ldots, \nu_p)$ to simplify the presentation. Thus, we can compactly express (2.4.1.1) as

$$L(x, \lambda, \nu) = f_0(x) + \lambda^T f(x) + \nu^T h(x). \tag{2.4.1.2}$$

The *dual function* $g(\lambda, \nu)$ is obtained by minimizing the Lagrangian (2.4.1.2) over all x , i.e.,

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu) = \inf_x (f_0(x) + \lambda^T f(x) + \nu^T h(x)). \tag{2.4.1.3}$$

The function $g(\lambda, \nu)$ is the infimum of a family of affine functions (parameterized by x). Therefore, regardless of whether the original problem is convex or not, the dual function is concave. Let us denote p^* as the optimal value of (2.4.0.10). Since x^* is feasible and is a, (possibly non-unique) minimizer of f_0, $f(x^*) \geq 0$, $h(x^*) = 0$ and for any (λ, ν) such that $\lambda \geq 0$,

$$L(x^*, \lambda, \nu) = f_0(x^*) + \lambda^T f(x^*) + \nu^T h(x^*) \leq f_0(x^*) = p^* \ , \tag{2.4.1.4}$$

and

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu) \leq L(x^*, \lambda, \nu) \leq f_0(x^*) = p^*. \tag{2.4.1.5}$$

In other words, for any pair of feasible dual variables ($\lambda \geq 0$), $g(\lambda, \nu)$ is a lower bound on p^*, the optimal value of the original problem.

2.4.2 The Dual Problem

The goal of the *dual problem* is to find the largest lower bound (say d^*) for p^*, i.e.,

$$d^* = \sup_{\lambda \geq 0, \nu} g(\lambda, \nu). \tag{2.4.2.1}$$

Any feasible pair (λ^*, ν^*) that maximizes $g(\lambda, \nu)$ is called optimal Lagrange multipliers or dual optimal solution. Since g is concave regardless of the original problem, the dual problem is always a convex optimization problem.

Chapter 3

Distributed Optimization

In this chapter we present some classical as well as state-of-the-art distributed optimization methods. In particular, we discuss classical decomposition technique, including primal and dual decomposition. We also discuss the state-of-the-art, alternating direction method of multipliers (ADMM) and the recent F-Lipshitz framework for distributed optimization. In later chapters, we summarize the inherent privacy preserving properties of those methods. Moreover, our algorithm developments in this thesis for privacy preserving car parking slot optimization is essentially based on dual decomposition method. All of the material presented in this chapter are essentially reproduced from [2], [3], [12], [13].

3.1 Decomposition Method

Decomposition is a general approach for solving a problem by breaking it up into smaller ones and solving each of the smaller ones separately, either in parallel or sequentially. Problems for which decomposition works in one step are called (block) separable, or trivially parallelizable. For example, suppose the variable x can be partitioned into subvectors $x_1, \ldots, x_k$, the objective is a sum of functions of x_i, and each constraint involves only variables from one of the subvectors x_i. This means that we can solve each problem involving x_i separately, and then re-assemble the solution x. When there is some coupling between the subvectors, the problems cannot be solved independently. In such situations, we have to rely on decomposition methods for distributed optimization. These techniques essentially solve a sequence of smaller problems and coordinate those subproblems to achieve the solution of the original coupled problem. In other words, decentralized solution methods can be interpreted as, simple protocols that allow a collection of

subsystems to coordinate their actions to achieve global optimality. We start by describing the classical decomposition method, primal decomposition and dual decomposition.

3.1.1 Primal Decomposition

Let us consider an unconstrained optimization problem that splits into two subproblems with a shared variable,

$$\begin{array}{ll} \text{minimize} & f(x) = f_1(x_1, y) + f_2(x_2, y) \\ \text{subject to} & x_1 \in X_1, \qquad x_2 \in X_2 \ , \end{array} \tag{3.1.1.1}$$

where the variables are x_1, x_2, and y. Here, x_1 and x_2 can be considered as the *private variables* or *local variables* associated with the first and the second subproblems, respectively, and y can be considered as the *public variable* or *interface variable* between the two subproblems. When y is fixed the problem is separable. Let $\Phi_1(x)$ represent the optimal value of the problem

$$\begin{array}{ll} \text{minimize} & f_1(x_1, y) \\ \text{subject to} & x_1 \in X_1 \end{array} \tag{3.1.1.2}$$

with the variable is x_1 and $\Phi_2(x)$ represent the optimal value of the problem

$$\begin{array}{ll} \text{minimize} & f_2(x_2, y) \\ \text{subject to} & x_2 \in X_2 \end{array} \tag{3.1.1.3}$$

with the variable is x_2. Then the original problem is equivalent to the problem

$$\text{minimize} \quad \Phi_1(y) + \Phi_2(y) \ , \tag{3.1.1.4}$$

where the variable is y. This problem is called the master problem. Whenever the functions f_1 and f_2 are convex, then the functions $\Phi_1(y)$ and $\Phi_2(y)$ are convex as well. The master problem can be solved for example by using the subgradient method.

repeat

Solve the subproblems (possibly in parallel).

Find $\bar{x_1}$ that minimizes $f_1(x_1, y)$, and a subgradient $g_1 \in \partial\Phi_1(y)$.

Find $\bar{x_2}$ that minimizes $f_2(x_2, y)$, and a subgradient $g_2 \in \partial\Phi_2(y)$.

Update complicating variable

$$y := y - \alpha_k(g_1 + g_2) \ ,$$

Here α_x is a step length that can be chosen in any of the standard ways [14]. Note that very iteration generates a $x^k = (x_1^k, x_2^k)$ that is feasible in the original problem.

3.1.2 Dual Decomposition

Let us next consider the same problem (3.1.1.1), but an equivalent formulation where we introduce new variables and enforce a consistency constrain, i.e.,

$$\begin{array}{ll} \text{minimize} & f(x) = f_1(x_1, y_1) + f_2(x_2, y_2) \\ \text{subject to} & x_1 \in X_1, \qquad x_2 \in X_2 \\ & y_1 = y_2 \ , \end{array} \tag{3.1.2.1}$$

where the variables are x_1, x_2, y_1, and y_2. Note that we have introduced a *local version* of the complicating variable y, along with a consistency constraint that requires the two local versions to be equal. Note that the objective is now separable, with the variable partition (x_1, y_1) and (x_2, y_2). Let us now form the dual problem. The Lagrangian is given by

$$L(x_1, y_1, x_2, y_2, \lambda) = f_1(x_1, y_1) + f_2(x_2, y_2) + \ \lambda^T y_1 - \lambda^T y_2, \tag{3.1.2.2}$$

and is separable. Let $g_1(\lambda)$ represent the optimal value of the problem

$$\begin{array}{ll} \text{minimize} & f_1(x_1, y_1) + \lambda^T y_1 \\ \text{subject to} & x_1 \in X_1 \end{array} \tag{3.1.2.3}$$

with the variable is (x_1, y_1) and $g_2(\lambda)$ represent the optimal value of the problem

$$\begin{array}{ll} \text{minimize} & f_2(x_2, y_2) + \lambda^T y_2 \\ \text{subject to} & x_2 \in X_2 \end{array} \tag{3.1.2.4}$$

with the variable is (x_2, y_2). Then the dual function is given by

$$g(\lambda) = g_1(\lambda) + g_2(\lambda), \tag{3.1.2.5}$$

where λ is the variable. Thus, the dual problem is expressed as

$$\text{minimize} \quad g(\lambda) = g_1(\lambda) + g_2(\lambda) \ , \tag{3.1.2.6}$$

with the variable is λ. This is indeed the master problem in *dual decomposition*. Again, we can use the subgradient method to solve the master problem (3.1.2.6) as follows:

repeat

Solve the subproblems (possibly in parallel).

Find x_1 and y_1 that minimizes $f_1(x_1, y_1) + \lambda^T y_1$.

Find x_2 and y_2 that minimizes $f_2(x_2, y_2) + \lambda^T y_2$.

Update dual variables

$\lambda := \lambda - \alpha_k(y_2 + y_1)$.

Here α_x is a step length that can be chosen in any of the standard ways [14]. It is worth of noting that at each step of the dual decomposition algorithm, we have a lower bound on p^*, the optimal value of problem (3.1.2.1). Specifically, we have

$$p^* \geq g(\lambda) = f_1(x_1, y_1) + \lambda^T y_1 + f_2(x_2, y_2) - \lambda^T y_2 \ , \tag{3.1.2.7}$$

where x_1, y_1, x_2, y_2 are the subproblem (3.1.2.3)-(3.1.2.4) solutions during any iteration of the subgradient method. The consistency constraint $y_1 = y_2$ is not feasible in general, i.e., $y_2 - y_1 \neq 0$. However, a feasible point can be constructed from the iterate as

$$(x_1, \bar{y}), \qquad (x_2, \bar{y}) \ , \tag{3.1.2.8}$$

where $\bar{y} = (y_1 + y_2)/2$. As a result, an upper bound on p^* is simply given by

$$p^* \leq f_1(x_1, \bar{y}) + f_1(x_2, \bar{y}). \tag{3.1.2.9}$$

3.2 Alternating Direction Method of Multipliers (ADMM)

The material presented in this section is based on [3], so we refer the reader to [3], for details. The main motivation for using ADMM is that it combines the benefits of dual decomposition and augmented Lagrangian methods for constrained optimizations. The result is a distributed algorithm with fast (compared to the subgradient method) convergence properties. We start with a simple convex constrained optimization problem. Then, we describe the dual ascent algorithm and augmented Lagrangian method for solving this problem, which are the key ingredients for ADMM. Consider the following convex constrained optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b \ , \end{array} \tag{3.2.0.10}$$

where the variable is x. The associated lagrangian is

$$L(x, y) = f(x) + y^T(Ax - b) \ , \tag{3.2.0.11}$$

where y is the dual variable or Lagrange multiplier associated with the equality constraint. The dual problem is given by

$$\text{maximize} \qquad g(y) \ , \tag{3.2.0.12}$$

where $g(y) = \inf_x L(x, y)$.

In the dual ascent method, we solve the dual problem using gradient ascent. Assuming that g is differentiable, the gradient ∇g of g at y is given by

$$\nabla g(y) = Ax^+ - b \ , \tag{3.2.0.13}$$

where

$$x^+ = \arg\min_x L(x, y) \ . \tag{3.2.0.14}$$

Thus, the dual ascent algorithm is given by

$$x^{k+1} = \arg\min_x L(x, y^k) \tag{3.2.0.15}$$

$$y^{k+1} = y^k + \alpha^k(Ax^{k+1} - b) \ , \tag{3.2.0.16}$$

where α^k is the step size. Note that $(Ax^{k+1} - b)$ corresponds to the gradient of g at y^k, i.e., $\nabla g(y^k)$. The first step is an x-minimization step, and the second step is a dual variable update. The dual variable y can be interpreted as a vector of prices. This algorithm is called dual ascent since, with appropriate choice of α, the dual function increases in each step.

However, in the case of convergence, the dual ascent algorithm discussed above heavily relies on assumptions like strict convexity or finiteness of f [3]. Let us now describe the augmented lagrangian method, which gracefully achieves convergence without such assumptions, and there more robust. Here we consider the following equivalent problem formulation, instead of original problem (3.2.0.10)

$$\begin{array}{ll} \text{minimize} & f(x) + (\rho/2)||Ax - b||_2^2 \\ \text{subject to} & Ax = b \ , \end{array} \tag{3.2.0.17}$$

where the variable is x and ρ is a positive scalar. We denote by $L_\rho(x, y)$ the Lagrangian associated with problem (3.2.0.17), and is given by

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + (\rho/2)||Ax - b||_2^2 \ , \tag{3.2.0.18}$$

where y represents the dual variables associated with the equality constraint. Applying dual ascent to the modified problem (3.2.0.17) yields the algorithm:

$$x^{k+1} = \arg\min_x L_\rho(x, y^k) \tag{3.2.0.19}$$

$$y^{k+1} = y^k + \rho(Ax^{k+1} - b) \ , \tag{3.2.0.20}$$

which is known as the method of multipliers. Note that the x-minimization step uses the augmented Lagrangian $L_\rho(x, y)$, instead of $L(x, y)$ in (3.2.0.11). Moreover, the penalty parameter ρ is used in the place of α^k in (3.2.0.16). The conditions for convergence of the method of multipliers (3.2.0.19)-(3.2.0.20) is far more general compared to the dual ascent method (3.2.0.15)-(3.2.0.16) [3]. However, when f is separable, the augmented Lagrangian L_ρ is not separable, because of the quadratic term $||Ax - b||_2^2$. As a result, the x-minimization step (3.2.0.19) cannot be performed in parallel for each subsystems.

We next describe the ADMM method, which can be considered as a blend between the dual ascent algorithm and the method of multipliers. Let us consider the problem of the form

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c \ , \end{array} \tag{3.2.0.21}$$

with variables $x \in R^n$ and $z \in R^m$. The augmented Lagrangian for problem (3.2.0.21) is given by

$$L_\rho(x, z, y) = f(x)+g(z)+y^T(Ax+Bz-c)+(\rho/2)||Ax+Bz-c||_2^2 \ , \tag{3.2.0.22}$$

where y denotes the dual variables as usual. Note that the direct application of method of multiplier method (3.2.0.19)-(3.2.0.20) for problem (3.2.0.21) results

$$(x^{k+1}, z^{k+1}) = \arg\min_{x,z} L_\rho(x, z, y^k) \tag{3.2.0.23}$$

$$y^{k+1} = y^k + \rho^k(Ax^{k+1} + Bz^{k+1} - c) \ . \tag{3.2.0.24}$$

In contrast, ADMM split the (x, z)-minimization step (3.2.0.23) into two sequential updates, namely x-minimization and z-minimization. Specifically, ADMM consist of the iteration

$$x^{k+1} = \arg\min_x L_\rho(x, z^k, y^k) \tag{3.2.0.25}$$

$$z^{k+1} = \arg\min_z L_\rho(x^{k+1}, z, y^k) \tag{3.2.0.26}$$

$$y^{k+1} = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \ . \tag{3.2.0.27}$$

where ρ is a positive scalar.

There are many convergence results for ADMM in the literature. Here, we do not go into explicit details and refer the reader for [3]. However, under the assumptions 1) f and g are closed, proper, and convex, 2) the augmented Lagrangian has a saddle point, we list 3 interesting convergence properties of ADMM algorithm:

$$L_0(x^*, z^*, y) < L_0(x^*, z^*, y^*) < L_0(x, z, y^*) \tag{3.2.0.28}$$

1. **Iterates approach feasibility**: $Ax^k + Bz^k - c \to 0$ as $k \to \infty$.
2. **Objective function of the iterates approaches optimal value** $f(x^k) + g(z^k) \to p^*$ as $k \to \infty$, where p^* is the optimal value .
3. **Dual variable convergence.** $y^k \to y^*$ as $k \to \infty$, where y^* is the dual optimal point .

$$r^{k+1} = A^{k+1} + B^{k+1} - c \tag{3.2.0.29}$$

Compared to interior point algorithms, which are based on the Newton's method, the convergence of ADMM algorithm is noticeably slow. However, the convergence is faster compared to the dual ascent method or classical dual decomposition techniques that rely on subgradient methods for solving the master problem.

3.3 Fast-Lispchitz optimization

An F-Lipschitz optimization problem is defined as:

$$\begin{array}{lll} \text{maximize} & f_0(x) & \\ \text{subject to} & x_i \leq f_i(x). & i = i, ..., l \\ & x_i = h_i(x), & i = l+1, \ldots, n \ , \\ & x \in D, & \end{array} \tag{3.3.0.30}$$

where $D \in R^n$ is a non empty, convex and compact set, $l \leq n$, with objective and constraints being continuous differentiable functions such that:

$$\begin{array}{ll} f_0(x) : D \to R^m, & m \geq 1 \\ f_i(x) : D \to R, & i = 1, ..., l \\ h_i(x) : D \to R, & i = l+1, ..., n \end{array} \tag{3.3.0.31}$$

And the following three properties are satisfied:

$$
\begin{array}{l}
\quad \nabla f_0(x) \geq 0 \qquad (f_0(x) \textit{is strictly increasing},) \\
\textit{and} \\
\quad \nabla_j f_i(x) \leq 0 \qquad \forall i \neq j, \quad \forall x \in D \\
\quad \nabla_j h_i(x) \leq 0 \qquad \forall i \neq j, \quad \forall x \in D \\
\textit{or} \\
\quad \nabla_i f_0(x) = \nabla_j f_0(x) \qquad \forall i \neq j, \quad \forall x \in D \\
\quad \nabla f_i(x) \geq 0 \qquad \forall i \neq j, \quad \forall x \in D \\
\quad \nabla h_i(x) \geq 0 \qquad \forall i \neq j, \quad \forall x \in D \\
\textit{and} \\
\quad |f_i(x) - f_i(y)| \leq \alpha_i ||x - y||, \quad i = 1, ..., l \qquad \forall x, y \in D \\
\quad |h_i(x) - h_i(y)| \leq \alpha_i ||x - y||, \quad i = l+1, ..., n \quad \forall x, y \in D \\
\qquad \textit{with} \quad \alpha_i \in [0, 1), \quad \forall i
\end{array}
\tag{3.3.0.32}
$$

All these properties are called, *qualifying properties* of an F-Lipschitz optimization problem. The objective function and the constraints are allowed to be linear or non linear functions, as for instance concave, convex, monomial, polynomial, etc. and also decomposable or not. For an F-Lipschitz problem is true the following theorem.

Theorem: Let an F-Lipschitz optimization problem be feasible. Then, the problem admits a unique Pareto optimum $x^* \in D$ given by the solutions of the following set of equations:

$$
\begin{array}{ll}
x_i^* = [f_i(x^*)]^D & i = 1, ...l \\
x_i^* = h_i(x^*) & i = l+1, ..n
\end{array}
$$

There is a unique optimal solution to F-Lipschitz optimization problems, which is achieved by solving the system of equations given by the projected constraints at the equality. If such a system of equations can be solved in a closed form, then we have the optimal solution in a closed form, otherwise we need numerical algorithms. The solution is obtained quickly by asynchronous algorithms of certified convergence. F-Lipschitz optimization can be applied to both centralized and distributed optimization. Compared to traditional Lagrangian methods, which often converge linearly, the convergence time of centralized F-Lipschitz algorithms is superlinear. It is proved that a class of convex problems, including geometric programming problems, can be cast

as F-Lipschitz problems, and thus they can be solved much more efficiently than interior point methods. Some typical optimization problems that occur on wireless sensor networks are shown to be F-Lipschitz.

Distributed computation
Let $x(0) \in D$ be the initial value of the optimal solution to a feasible F-Lipschitz problem. Let $x^i(k) = [x_1^i(\tau_1^i(k)), x_2^i(\tau_2^i(k)), ..., x_n^i(\tau_n^i(k))]$ the vector of decision variables available at node i at time $k \in N$, where $\tau_j^i(k)$ is the delay with which the decision variable of node j is communicated to node$i$. Then, the following iterative algorithm converges to the optimal solution:

$$x_i(k+1) = [f_i(x^i(k))]^D \qquad i = 1, ..., l \tag{3.3.33}$$

$$x_i(k+1) = h_i(x^i(k)) \qquad i = l+1, ..., n.$$

Every node i of the network collects asynchronously the decision variables at time$k$ and update its decision variable by the iteration (3.5.4). Notice that when $f_i(x)$ depends only on the decision variables of the neighboring nodes, the communications of these variables to node i can be very fast. In other situations, $f_i(x)$ can be given by an oracle locally at node i without any communication of decision variables from the other nodes. We refer the reader to [12] for details.

Chapter 4

Privacy preserving optimization

Privacy preserving problems are very common nowadays and the study of more efficient solutions is a topic more interesting. They occur in many areas, such as data publishing, data mining, secret voting, auctions, scientific and statistical computations. The main requirements are that the entities involved in the system are unwilling to reveal the date they hold or make them public and they may want to collaborate and find the solution of a bigger problem without revealing the private data. A possible solution is for the parties to agree on a commonly trusted entity T: then T could gather the private problem data, solve the optimization problem on behalf of the parties, and announce the result. Also, the parties can replace the trusted entity T with a cryptographic protocol while still preserving the same level of security that the trusted entity intuitively provides. These early protocols are however very far from being practical. There are some existing approaches to handle this kind of problem and a classification of them is as follows:

- **cryptographic methods**;
- **transformation-based methods**;
- **decomposition-based methods**.

4.1 Privacy preserving solution methods

In this section we describe with more details the aforementioned approaches.

Cryptographic methods use existing cryptographic techniques in order to implement a privacy-preserving version of the Simplex or Interior Point

methods. The cryptography techniques are vary, like the secret sharing (eg. Shamir sharing) or thereshold homomorphic public key cryptography (eg. Pailler encryption). Also the homomorphic encryption and blind-and permute procedures to permute the tableau at each iteration, like in [15]. In [16] the authors used a solution approach based on secret sharing and the protocols use a variant of the simplex algorithm with fixed-point rational number, while in [17] the authors used a secret-sharing with shared matrix indexes. These techniques offer better computational complexity give better performance over Simplex for very large problems [18], [19]. A privacy-preserving interior point method has been presented in [26, chapter 8].

The **transformation-based methods** use some cryptography sub-protocols in order to transform the original linear problem. The transformation of the problem is made using random matrices and then the resulting problem is solved in the transformed domain. The cryptography sub-protocols are often used only at the beginning of the algorithm. Then, one party solved the transformed problem and the solution is converted back to the original problem. The first approach that used such transformation techniques was prosposed by Du [23] and now there are several improvements, expecially in terms of the level of the knowledge of the data of the interested parties. Bednarz [22], proposed a method, a modified variants of the Du methods, where some of the complications of the original approach were removed. Bednarz considers a case where the ownership of the objective function and the constraints are distributed among two entities. Recently Mangasarian [4],[24] proposed the first transformation method for the multi-party environments, where the data are assumed to be vertically or horizontally partitioned. Here no cryptographic operations are required to apply the transformation, making these methods very efficient compared to the methods relying on cryptography.

In **decomposition-based methods** the private data can be partitioned between the agents and they can collaborate to solve the problem in a parallel and distributed fashion, without revealing their private variables. The concept of privacy is intrinsic. So each agent achieves its optimum using only local information. In this way the private data is not required to be exchanged among the agents to solve the problem and for that reason they are privacy preserved. This falls in the general area of distributed decision making with incomplete information.

The first two categories of methods are described in detail in [25]. Let us next compare the three main approaches mentioned above.

4.2 Comparison of privacy preserving solution methods

The first noticeable difference between cryptography and transformed-based methods is in their efficiency of the computations. In cryptographic approaches at each step of the algorithm the encryption/decryption of the data is applied. In transformed-based methods, instead, these operations of the private data are done only once. Therefore, transformed based approaches can outperform cryptographic approaches, in terms of the computational efficiency. The decomposition techniques, however, do not require any kind of transformation domain because the data are inherently encrypted. This method is therefore much more efficient than the others two. The second advantage is the freedom to use any LP solver, unlike the cryptographic method is tied to a particular LP solver. Another advantage of decomposition techniques compared to the others two is that the methods are scalable. Therefore, very large problems can be handled gracefully by using decomposition techniques. Moreover, the decomposition techniques are fairly general and can address many places where the transformation based methods fails. Most cryptographic methods, infact, is carried out over a finite field and thus, is constrained to integer data values. For example, some of the Simplex-based methods impose integer restrictions on the objective function coefficient and constraint coefficients. This means that only integer variants of Simplex can be used. Solving non trivial LP using these algorithms leads to computations with integers potentially involving thousands of bits. Therefore, cryptographic approaches are not good in the case of large problems. Transformation-based methods and decomposition based approaches, instead, are free to use floating-point arithmetics. Finally, the algorithms are conceptually simpler, good for large problems and don't require specialized optimization software.

Transformed-based methods and decomposition based approaches have, however, several disadvantages compared to the cryptographic methods. The most important one is that these methods only provide heuristic security guarantees, Indeed, without the cryptographic protocols at each step, security is more challenging to prove. Cryptographic approaches guarantee a robust security. Another limit of transformation-based methods and the decomposition based approaches is the dependence to the problem structure, in particular on the way the data is partitioned between agents. There is also no standard way to define the subset of LPs to which a transformation applies. The cryptographic methods, instead, are more generics and robust to problem structure and the types of contraints and variables. In the table 4.1

Cryptographic methods	**Transformation-based methods**	**Decomposition-based methods**
inefficient	efficient	efficient
restricted to LP problem	not restricted to LP problem	not restricted to LP problem
protocols often tied to a particular solver	free to use any solver	free to use any solver
operations are restricted to finite field	handle real/complex vecor spaces	handle real/complex vecor spaces
small scale problems	large scale problems	very large scale problems
not scalable	scalable	scalable
privacy via encrypted domain	privacy via transformed domain	privacy is intrinsic
robust to problem structure	very sensitive to problem structure	data partitioning dependence
robust security guarantees	heuristic security guarantees	heuristic security guarantees

Table 4.1: Privacy preserving classification based on the classical distributed optimization methods

we summarize main advantages and disadvantages that we discussed above.

4.3 Privacy preserving classification problems

In this section we categorize some existing methods to solve decentralized optimization problem in terms of privacy preserving. In particular we categorize these methods based on the different level of privacy preserving request in the problem. Also, there are several kind of optimization problems, such linear, convex, non conex and Fast-Lipschitz. So, depending on the mathematical nature of the problem, we try to find a specific algorithm that solve it in the better way. Distributed optimization is used in a wide area of problems, such as Cloud Computing, Data Mining, Vehicular Rounting, Online Learning, Belief Propagation, Destributed Estimation and so on. So, another classification is based just on the kind of the application.

4.3.1 Privacy preserving algorithms

A particular privacy preserving method applied to a **linear optimization problem** are described in [4]. This approach will allow us to solve a distributed optimization problem by a random linear transformation that will not reveal any of the privately held data but will give a publicly available exact minimum value to the origianl problem. Thus, we are able to solve a privacy preserving transformed linear program that generates an exact solution to the original linear program without revealing any of the privately data of which node. The m-by-n constraits matrix A of the bacis linear problem is divided into p blocks of columns, each block of which toghether with the corresponding block of cost vector, is owned by a distinct entity. Each entity not willing to share or make public its column group or cost coefficient vector. Component groups of the solution vector of the privacy preserving transformed linear program can be decoded only by the group owners and can be made public by these entities to give the exact solution vector of the original linear problem.

Considering the **problem of optimizing a convex function** subject to a collection of convex inequality constraints and set constraints, we could refer to the paper [5]. In which is assumed that each node has a set of a private optimization variables that partecipate in the global optimization problem but are unknown to other nodes of the graph. This distributed algorithm operates over any connected graph of processors and yields a solution that is arbitrarily close to a global optimal solution, where proximity to optimality is controlled by a parameter that affects a tradeoff in the required computation time. This new algorithm is inspired of Lyapunov drift theory.

In [11] is proposed a new distributed algorithm, named **D-ADMM**, based on the alternating direction method of multipliers (ADMM) for solving. In a separale optimization problem, the cost function, and the constraint set is the intersection of all the agents' private constraint sets. In this paper is required the private cost function and the constraint set of a node to be known by that node only, during and before the execution of the algorithm.

If we have an optimization **problem with a concave objective function** we could refer to [6]. The paper proposes an adaptation of Lagrangian method to solve distributed weighting method for both strictly concave and not strictly concave (e.g. linear) value functions for a maximization problem, maintaining the privacy of the participating parties.

In **Cloud Computing** the concept of privacy preserving has an important role. Indeed, costumer confidential data processed and generated during the computation need to be secret. The problem of securely outsourcing computation in cloud computing is formalized in [7]. It is based on on a problem transformation techniques that enable customers to secretly transform the original problem into some random one while protecting sensitive input/output information.

Another field in which privacy preserving plays an important role is **Online Learning**. In [8] is considered a general distributed outonomous online learning algorithm to learn from fully decentralized data sources. Learners need to exchange information between them, so a local learner updates its local parameter basing on the local subgradient and then propagates the parameter to other learners. The paper examines under which conditions a malicious node cannot recostruct all subgradients of other nodes based on the parameter vectors of its adjacent nodes. So, the algorithm has intrinsic privacy preserving properties if the network topologies respect some conditions.

Belief propagation, also known as Sum-product message passing, is a message passing algorithm for performing inference on graphical models, such as Bayesian networks and Markov random fields. It calculates the marginal distribution for each unobserved node, conditional on any observed nodes. Belief propagation is commonly used in artificial intelligence and information theory and has demonstrated empirical success in numerous applications including low-density parity-check codes, turbo codes, free energy approximation, and satisfiability. The paper [9] provides provably privacy preserving versions of belief propagation and other local message passing algorithms on large distributed networks. Each party learns their conditional probability of exposure to the didease and absolutely nothing else. A party can efficiently compute after having partecipated in the protocol, they could have efficiently computed alone given only the value of their conditional propability. Thus, the protocol leaked no additional information beyond its desired outputs. The paper shows how to blend tools from cryptography with local message passing algorithms in a way that preserves the original computations, but in which all messages appear to be randomly distributed from the viewpoint of any individual.

The **Vehicle Routing Problem (VRP)** is a combinatorial optimization and integer programming problem seeking to service a number of customers with a fleet of vehicles. In its multiple depot variant, the routes of vehicles

Distributed solver method	**Privacy preserving decision variables**	**Privacy preserving utility function**	**Privacy preserving constraints**
Primal Decomposition	√	|	|
Dual Decomposition	√	|	|
ADMM	|	√	|
Fast-Lipschitz	|	√	√

Table 4.2: Privacy preserving classification based on the classical distributed optimization methods

located at various depots must be optimized to serve a number of costomers. The paper [10] investigates how to protect the privacy of delivery companies, when each depot is owned by a different company with a limited view of the overall problem.

4.3.2 Classification based on the classical distributed optimization methods

In the first chapter we discussed the most important methods to solve a distributed optimization problem. Now, we categorize these methods by the viewpoint of privacy preserving. So, in the table 4.2 we explain, for each of them, if it is privacy preserving in terms of utility function, decision variables or contraints.

Mathematical nature	Privacy preserving decision variables	Privacy preserving utility function	Privacy preserving constraints
Linear	[4]	[4]	[4]
Convex	|	[5],[11]	[5],[11]
Non Convex	|	[6]	|
Fast-Lipschitz	|	[12]	[12]

Table 4.3: Privacy preserving classification based on the mathematical nature of the optimization problem

4.3.3 Classification based on the mathematical nature of the optimization problem

Another kind of classification is based on the mathematical nature of the optimization problem. As view in the previous chapter, there are some papers which propose a method to solve each of them. In particular we can make a classification to linear, convex, non convex and F-Lipshitz problem. For each entry of the table we indicate which paper solve the problem, always making a distinction about the level of privacy preserving they approach (see table 4.3).

4.3.4 Classification based on the application

In the previous chapter we talked about various application based on distributed optimization that require privacy preserving. So, in the table (4.4) we make a classification of them.

Application	Privacy preserving decision variables	Privacy preserving utility function	Privacy preserving constraints
Online Learning	[8]	[8]	|
Cloud Computing	[7]	[7]	[7]
Belief Propagation	[9]	[9]	[9]
Vehicle Routing	[10]	|	|

Table 4.4: Privacy preserving classification based on the application

Chapter 5

Car-parking problem

One of the most relevant problem in urban transportation is the traffic congestion and parking difficulties, which are also a major cause for losing time. These two problems are interrelated since looking for a parking space creates additional delays and impairs local circulations. Given the importance of efficient car parking strategies, in this chapter, we place a greater emphasis on car parking problem, together with privacy preserving properties. Our goal is to reduce the distance from the parking slot to the intended destination of the car, thus helping the driver to easily find a parking closer to their places of interest.

Our proposed solution method is privacy preserving, in the following sense: vehicles do not want to reveal their destinations during the algorithm iterations. In addition, the proposed solution method is distributed among users (vehicles) with a little central coordination. Moreover, the method is fair, roughly speaking, it finds a allocation such that the maximum distance to from the parking slot to the destination of cars is minimized. Thus, our solution method is privacy preserving distributed, and fair.

In particular, we consider the car park illustrated in the figure 5.1 to mathematically model the problem. Parking slot assignment is time slotted, with slot period T. At the beginning of any time slot t, the number of free slots in the park should be known. Moreover, their details (e.g., location information) should be informed to the *selected* set of cars which are scheduled for parking at the beginning of the time slot t. Each car then knows the distances from free slots to their intended destination. In particular, this information is simply extracted from a table (see table 5.2), which contains all the distances from every free slot to every shop. At the beginning of time slot t, it is required to decide the slot-vehicle assignment, which is indeed the *binary* decision variables. This is shown in table 5.3. For example, if the j-th slot is occupied by the i-th vehicle, we indicate this assignment by using 1 in

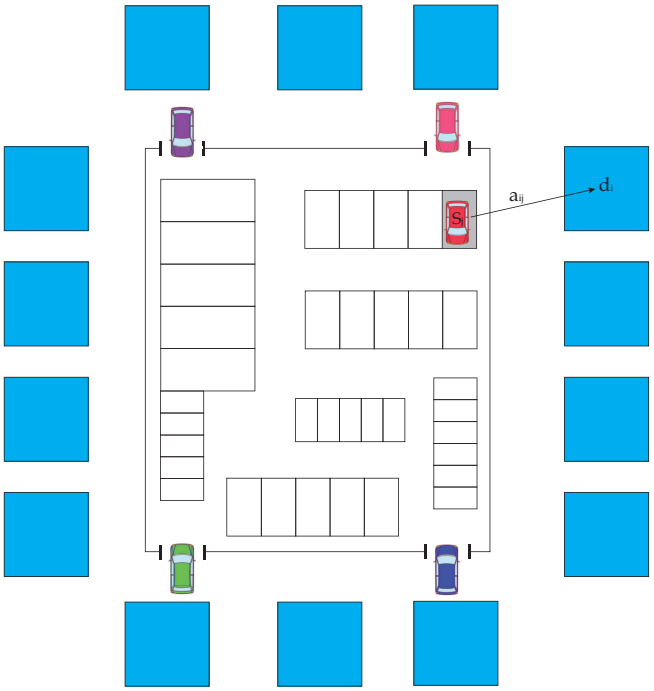

Figure 5.1: Car-Parking model

the (ij)th position, otherwise 0. The formulation can model parking assignments for different kind of vehicles, with different sizes and dimensions, e.g., cars, vans, motorbikes, and scooters.

In such a formulation, vehicles should maintain data such as the dimension of the slots. Roughly speaking, the following should be considered in the problem formulation.

- The distances from free slots to the destinations or shops. This information is required for constructing the objective function of the problem.
- The availability of the slots and their assignment: this is required for expressing constraints to enforce a correct assignment. For example, the assignment shown in table 5.3 is correct. But the assignment shown in table 5.4 is incorrect.
- The dimensions of the vehicles over the slots: this information is again required for expressing constraints.

Shops	**Slots**				
	S1	S2	S3	S4	S5
shop1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
shop2	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
shop3	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
shop4	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}

Table 5.1: Table of distances: for each shop (vehicle destination) is indicated the distance to each slot of the parking.

Shops	**Slots**				
	S1	S2	S3	S4	S5
shop1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
shop2	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
shop3	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
shop4	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}

Table 5.2: Table of distances: for each shop (vehicle destination) is indicated the distance to each slot of the parking.

Vehicles	**Slots**				
	S1	S2	S3	S4	S5
vehicle1	0	0	1	0	0
vehicle2	0	0	0	0	1
vehicle3	1	0	0	0	0
vehicle4	0	0	0	1	0

Table 5.3: Table of slots' availability: for eache slot is indicated if it's assigned to some vehicles or not.

Vehicles	**Slots**				
	S1	S2	S3	S4	S5
vehicle1	0	0	1	0	0
vehicle2	0	0	0	0	1
vehicle3	1	0	0	0	0
vehicle4	1	0	0	1	0

Table 5.4: Incorrect assignment of parking

5.1 Notations

Now we introduce essential notations to formulate the problem:

- $j = 1, ..., M$: the indexes of the parking slots
- W_j: the width of jth slot
- L_j: the length of jth slot
- $i = 1, ..., N$: the indexes of the scheduled vehicles for parking at the beginning of the time slot
- w_i: the width of ith vehicle
- l_i: the length of ith vehicle
- $d_i = (d_i^x, d_i^y)$: the destination coordinates of the ith vehicle
- $a_{ij}(d_i)$: the distance from the jth slot to the destination of ith vehicle. In particular, let (s_j^x, s_j^y) denote the coordinates of the jth slot. Then, $a_{ij}(d_i)$ is given by

$$a_{ij}(d_i) = \alpha\sqrt{(d_i^y - s_j^y)^2 + (d_i^x - s_j^x)^2} \ ,$$

  where α is a parameter known to all the scheduled vehicles only and is used to perform a scalar transformation to the distance. This scalar transformation further increases the privacy of the destinations of the vehicles to a third party.

- x_{ij}: the variable to indicate the vehicle-slot assignment. In particular,

$$x_{ij} = \begin{cases} 1 \text{ if the } j\text{th slot is assigned to the } i\text{th vehicle} \\ 0 \text{ otherwise} \end{cases} \tag{5.1.1}$$

5.2 Problem formulation

The car-parking problem explained in the previous chapter can be written in a mathematical form as follow:

$$\text{minimize } \max_{i \in N} \sum_{j=1}^{M} x_{ij} a_{ij}(d_i)$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{i=1}^{N} x_{ij} \leq 1, && \forall j && (5.2.0.1)\\
& \sum_{j=1}^{M} x_{ij} = 1, && \forall i && (5.2.0.2)\\
& x_{ij} \in \{0,1\} && && (5.2.0.3)\\
& W_j \geq w_i x_{ij}, && \forall i,j && (5.2.0.4)\\
& L_j \geq l_i x_{ij}, && \forall i,j && (5.2.0.5)
\end{aligned}$$

where the variables are x_{ij}, $i = 1, \ldots, N$, $j = 1, \ldots, M$. The constraint (5.2.0.1) requires that each slot is assigned at most to one vehicle. Such an assignment is shown in table 5.3, where parking slots S2 has been assigned no vehicles and all others have been assign one vehicle each. Table 5.4 shows a constraint violation, where two vehicles (vehicle3 and vehicle4) have been assigned the same slot (i.e., the parking slot S1). Constraint (5.2.0.2) imposes the condition that each vehicle must be assigned to a slot. In this thesis we assume that the number of scheduled vehicles is smaller than the free parking slots, i.e., $N \leq M$. Otherwise the problem is clearly infeasible. Thus, a schedular should take into account these issues, which are extraneous to the main focus of the thesis. See table 5.3 and table 5.4 for examples of correct and incorrect assignments, respectively. Constraint (5.2.0.3) requires that the values of x_{ij} to be 0 or 1.

Constraints (5.2.0.4-5.2.0.5) are clearly associated with dimensions of slots and vehicles (see figure 5.2 and 5.3). For notational simplicity, let

$$\beta_{ij} = \frac{W_j}{w_i}$$

and

$$\gamma_{ij} = \frac{L_j}{l_i} \ .$$

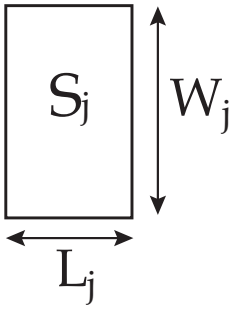

Figure 5.2: Slot's dimension

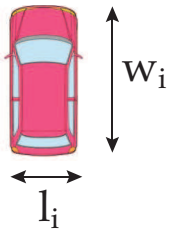

Figure 5.3: Vehicle's dimension

Then, the optimization problem becomes:

$$\text{minimize} \ \max_{i \in N} \sum_{j=1}^{M} x_{ij} a_{ij}(d_i)$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{i=1}^{N} x_{ij} \leq 1, \qquad \forall j && (5.2.0.6)\\
& \sum_{j=1}^{M} x_{ij} = 1, \qquad \forall i && (5.2.0.7)\\
& x_{ij} \in \{0,1\} && (5.2.0.8)\\
& \beta_{ij} x_{ij} \leq 1, \qquad \forall i,j && (5.2.0.9)\\
& \gamma_{ij} x_{ij} \leq 1, \qquad \forall i,j && (5.2.0.10)
\end{aligned}$$

where the variables are x_{ij}, $i = 1, \ldots, N$, $j = 1, \ldots, M$. Note that the problem above is combinatorial and we have to rely on global optimal methods

such as exhaustive search and branch and bound methods to solve it. The main disadvantage of global methods is the prohibitively expensive computational complexity, even in the case of very small problems. Such methods are not scalable and impractical. In the sequel, we provide a method based on duality. Even though the optimality cannot be guaranteed, the proposed method is efficient, fast, and allows distributed implementation with a little coordination from a central controller. Therefore, the method is favorable for practical implementations.

5.3 Finding the dual problem

We start by equivalently formulating problem (5.2.0.6-5.2.0.10) in its epigraph form. The equivalent problem is given by

$$\begin{aligned}
\text{minimize} \quad & t \\
\text{subject to} \quad & \sum_{j=1}^{M} x_{ij} a_{ij}(d_i) \leq t, \qquad \forall i && (5.3.0.1) \\
& \sum_{i=1}^{N} x_{ij} \leq 1, \qquad \forall j && (5.3.0.2) \\
& \sum_{j=1}^{M} x_{ij} = 1, \qquad \forall i && (5.3.0.3) \\
& x_{ij} \in \{0,1\} && (5.3.0.4)
\end{aligned}$$

where the variables are t and $x = (x_{ij})_{i=1,\ldots,N, j=1,\ldots,M}$. Note that we have removed the constraints (5.2.0.9-5.2.0.10) of the original problem for simplicity. But the solution method to be given fully extends to include the constraints (5.2.0.9-5.2.0.10) as well. Now we want to apply duality theory to the epigraph form. It is important to note that we want also to decouple the problem among the vehicles. We can clearly see that constraints (5.3.0.1-5.3.0.2) are the coupling constraints of the problem. The constraints (5.3.0.3-5.3.0.4) are already decoupled among the vehicles and we can treat them as implicit constraints. Next, we introduce Lagrange multiplies and form the partial Lagrangian by dualizing the coupling constraints (5.3.0.1-5.3.0.2). So, we have to introduce the Lagrangian multipliers, in particular $\lambda = (\lambda_i)_{i \in \{1,\ldots,N\}}$ for the first set of inequality constraints and multipliers $\mu = (\mu_j)_{j \in \{1,\ldots,M\}}$

for the second set of inequality constraints. The Lagrangian associated with problem (5.3.0.1-5.3.0.4) is:

$$\begin{aligned} L(t,x,\lambda,\mu) &= t + \sum_{i=1}^{N} \lambda_i (\sum_{j=1}^{M} x_{ij} a_{ij}(d_i) - t) + \sum_{j=1}^{M} \mu_j (\sum_{i=1}^{N} x_{ij} - 1) \\ &= t + \sum_{i=1}^{N}\sum_{j=1}^{M} \lambda_i x_{ij} a_{ij}(d_i) - \sum_{i=1}^{N} \lambda_i t + \sum_{j=1}^{M}\sum_{i=1}^{N} \mu_j x_{ij} - \sum_{j=1}^{M} \mu_j \\ &= t(1 - \sum_{i=1}^{N} \lambda_i) + \sum_{i=1}^{N}\sum_{j=1}^{M} (\lambda_i a_{ij}(d_i) + \mu_j) x_{ij} - \sum_{j=1}^{M} \mu_j \end{aligned} \tag{5.3.0.5}$$

Now we need to find the dual function $g(\lambda, \mu)$. To do this, we minimize the Lagrangian with respect to primal variables t and x, i.e.,

$$g(\lambda,\mu) = \inf_{\substack{t \in R, \\ \sum_{j=1}^{M} x_{ij} = 1, \forall i, \\ x_{ij} \in \{0,1\}, \forall i,j}} L(t,x,\lambda,\mu) \tag{5.3.0.6}$$

$$= \begin{cases} \inf_{\substack{\sum_{j=1}^{M} x_{ij} = 1, \forall i, \\ x_{ij} \in \{0,1\}, \forall i,j}} \sum_{i=1}^{N}\sum_{j=1}^{M} (\lambda_i a_{ij}(d_i) + \mu_j) x_{ij} - \sum_{j=1}^{M} \mu_j & \sum_{i=1}^{N} \lambda_i = 1 \\ -\infty \qquad \text{otherwise} \end{cases} \tag{5.3.0.7}$$

$$= \begin{cases} \sum_{i=1}^{N} (\inf_{\substack{\sum_{j=1}^{M} x_{ij} = 1, \forall i, \\ x_{ij} \in \{0,1\}, \forall i,j}} \sum_{j=1}^{M} (\lambda_i a_{ij}(d_i) + \mu_j) x_{ij}) - \sum_{j=1}^{M} \mu_j & \sum_{i=1}^{N} \lambda_i = 1 \\ -\infty \qquad \text{otherwise} \end{cases} \tag{5.3.0.8}$$

$$= \begin{cases} \sum_{i=1}^{N} g_i(\lambda,\mu) - \sum_{j=1}^{M} \mu_j & \sum_{i=1}^{N} \lambda_i = 1 \\ -\infty \qquad \text{otherwise} \end{cases} \tag{5.3.0.9}$$

In 5.3.0.7, we have removed the linear term

$$t(1 - \sum_{i=1}^{N} \lambda_i) \ ,$$

because it is bounded below only when $\sum_{i=1}^{N} \lambda_i = 1$. The constraints of inf operator in 5.3.0.7 are separable among the vehicles $i \in \{1, \ldots, N\}$. Therefore, we can move the inf operator inside the summation $\sum_{i=1}^{N}$, see (5.3.0.8). The function $g_i(\lambda, \mu)$ denotes the optimal value of the following problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{M} (\lambda_i a_{ij}(d_i) + \mu_j) x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{M} x_{ij} = 1, \\ & x_{ij} \in \{0, 1\} \, , \forall j, \end{aligned} \tag{5.3.10}$$

with the variable $(x_{ij})_{j \in \{1,\ldots,M\}}$. Each vehicle has to solve the problem (5.3.10). Note that the problem (5.3.10) is combinatorial, but it has a closed form solution given by:

$$x_{ij}^* = \begin{cases} 1 & j = \arg\min_{n \in \{1,\ldots,M\}} (\lambda_i a_{in}(d_i) + \mu_n) \\ \\ 0 & \text{otherwise} \end{cases} \tag{5.3.11}$$

The dual problem is given by:

$$\begin{aligned} \text{maximize} \quad & g(\lambda, \mu) = \sum_{i=1}^{N} g_i(\lambda, \mu) - \sum_{j=1}^{M} \mu_j \\ \text{subject to} \quad & \sum_{i=1}^{N} \lambda_i = 1, \\ & \lambda_i \geq 0, \forall i, \\ & \mu_j \geq 0, \forall j \ . \end{aligned} \tag{5.3.12}$$

where variables are $(\lambda_i)_{i \in \{1,\ldots,N\}}$ and $(\mu_j)_{j \in \{1,\ldots,M\}}$.

5.4 Solving the dual problem

To solve the dual problem 5.3.12 we use the projected subgradient method, which is often applied to large-scale problems with decomposition structures. Note that $g(\lambda, \mu)$ is a concave function, therefore, we need to find the subgradient of $-g$ at a feasible (λ, μ). We denote by s the subgradient and for clarity we separate s into two vectors as follows:

$$s = (u, v), \tag{5.4.0.1}$$

where $u = (u_i)_{i\in\{1,\dots,N\}}$ is the part that corresponds to λ and $v = (v_j)_{j\in\{1,\dots,M\}}$ the part that corresponds to μ. The (negative of) dual function $-g(\lambda, \mu)$ is given by

$$\begin{aligned}
-g(\lambda,\mu) &= \sum_{j=1}^{M}\mu_j - \sum_{i=1}^{N} g_i(\lambda,\mu) \\
&= \sum_{j=1}^{M}\mu_j - \sum_{i=1}^{N}\sum_{j=1}^{M}(\lambda_i a_{ij}(d_i) + \mu_j)x_{ij}^* \\
&= \sum_{j=1}^{M}\mu_j - \sum_{j=1}^{M}\mu_j\sum_{i=1}^{N}x_{ij}^* - \sum_{i=1}^{N}\lambda_i\sum_{j=1}^{M}a_{ij}(d_i)x_{ij}^*.
\end{aligned}$$

So we obtain, for all $i \in N$:

$$u_i = -\sum_{j=1}^{M} a_{ij}(d_i)x_{ij}^* \qquad \text{and} \qquad v_j = 1 - \sum_{i=1}^{N} x_{ij}^*, \tag{5.4.0.2}$$

where x_{ij}^* given in (5.3.11). The projected subgradient method is given by

$$(\lambda, \mu)^{(k+1)} = P((\lambda, \mu)^{(k)} - \alpha_k(u, v)^{(k)}), \tag{5.4.0.3}$$

where k is the current iteration index of the subgradient method, $P(z)$ denotes the Euclidean projection of z onto the feasible set of the dual problem (5.3.12), and $\alpha_k > 0$ is the kth step size, chosen to guarantee the asymptotic convergence of the subgradient method, e.g., $\alpha_k = 1/k$. Since the feasible set of dual problem is separable in λ and μ, the projection $P(\cdot)$ can be performed independently. Therefore, the iteration (5.4.0.3) is equivalently performed as follows:

$$\lambda^{(k+1)} = P_s(\lambda^{(k)} - \alpha_k u^{(k)}) \tag{5.4.0.4}$$

$$\mu^{(k+1)} = [\mu^{(k)} - \alpha_k v^{(k)}]^+, \tag{5.4.0.5}$$

where $P_s(\cdot)$ is the Euclidean projection onto the probability simplex

$$\left\{\lambda \,\middle|\, \textstyle\sum_{i=1}^{N} \lambda_i = 1, \lambda_i \geq 0\right\}$$

and $[\,\cdot\,]^+$ is the Euclidean projection onto the nonnegative orthant.

5.5 Distributed implementation

Let us now present the distributed solution methods for the car parking problem. Here, we capitalize on the ability to construct the subgradient (u, v) in a distributed fashion via the coordination of scheduled vehicles. As we have already mentioned, there should be a little involvement of a central controller (e.g., owner of the car park) for realizing the overall algorithm. This involvement is mainly for dual variable updating and broadcasting new dual variables to the scheduled vehicles until the algorithm stops.

Algorithm : **Distributed algorithm for car-parking**

1. Central controller sets $k = 1$ and broadcasts the initial (feasible) $\lambda_i^{(k)}$ and $(\mu_j^{(k)})_{j=\in\{1,\ldots,M\}}$ to vehicle i, $i \in \{1, \ldots, N\}$.

2. Vehicle i sets $\lambda_i = \lambda_i^{(k)}$ and $\mu = \mu^{(k)}$ and locally solves the problem (5.3.10), to yield the solution $(x_{ij}^*)_{j=1,\ldots,M}$, which is given by (5.3.11).

3. Vehicle i computes *scalar* u_i from (5.4.0.2) and transmits this to the central controller. For each j, scheduled vehicles communicate (binary variables x_{ij}^*) and construct the *scalar* v_j and transmits this to the central controller.

4. Subgradient iteration:

   - Central controller forms $u^{(k)}$ and performs (5.4.0.4)
   - Central controller forms $v^{(k)}$ and performs (5.4.0.5).

5. Stopping criterion: if the stopping criterion is satisfied, then STOP. Otherwise, set $k = k + 1$, and central controller broadcasts the new $\lambda_i^{(k)}$, $(\mu_j^{(k)})_{j\in\{1,\ldots,M\}}$ to vehicle i, $i = 1, \ldots, N$, and go to step 2.

5.5.1 Algorithm description

In step 1, the algorithm starts by choosing initial feasible values for $\lambda_i^{(k)},\ i = 1, \ldots, N$ and $\mu_j^{(k)},\ j = 1, \ldots, M$. After receiving these values from the central controller, each vehicle computes both $\{x_{ij}\}_{j\in\{1,\ldots,M\}}$ in step 2 in a decentralized fashion. Step 3 is used for communication and coordination. In particular, each vehicle i constructs *scalar* parameter u_i and sends this to the central controller. Moreover, scheduled vehicles communicate *binary* parameters to construct v_j and transmits this to the central controller. We see that the solution method is privacy preserving, because no one (vehicles and the central controller) can guess vehicle i's destination data (d_i^x, d_i^y). Note that, step 3 does *not* reveal the private destinations of vehicle i, i.e., (d_i^x, d_i^y) to the the central controller. This is mainly because, (d_i^x, d_i^y) is hidden inside u_i. However, for larger iteration index k, the central controller can guess that the algorithm has a feasible solution and as a result, $-u_i = a_{ij}(d_i)$ for some parking slot j. But, still $a_{ij}(d_i)$ is a α-scaled version of the true distance from vehicle i's parking slot to its intended destination. Therefore, without knowing α, the central controller finds it difficult to compute vehicle i's true destination coordinates (d_i^x, d_i^y). Moreover, in step 3, the scheduled vehicles coordinate their solutions $(x_{ij})_{j\in\{1,\ldots,M\}}$ with each other. This communication also privacy preserving, because, no scheduled vehicle can know the vehicle i's problem data $(a_{ij}(d_i))_{j\in\{1,\ldots,M\}}$. Then, the algorithm perform step 4, the subgradient iterations (5.4.0.4-5.4.0.5). In this way it generates a sequence of $\lambda_i^{(k)}$ and $\mu_j^{(k)}$, $k = 1, \ldots$. The price update or the Lagrange multiplier update mechanism attempt to achieve primal feasibility of the original problem (5.3.0.1-5.3.0.4). However, because the problem is nonconvex, primal feasibility is often not guaranteed. Therefore, it usually required to call a subroutine to construct a feasible solution at the end of the algorithm.

Finally, step 5 is the stopping criteria. If it is satisfied, then the algorithm stops, otherwise central controller broadcasts $\lambda_i^{(k)}$ and $\mu^{(k)}$ to all vehicles $i \in \{1, ..., N\}$ and the algorithm is repeated.

5.5.2 Recovering the primal feasible point

By using the algorithm above, we find the optimal dual variables. But the main requirement is to find an optimal solution for the primal problem. However, this is often impossible. First, note that the problem is nonconvex and as a result there is no guarantee that dual approach that we followed gives a mechanism for finding the optimal primal solution or even a primal feasible point. Therefore, as pointed in the description of step 4, we have to rely on a (heuristic) subroutine to construct a primal feasible point after the

algorithm terminates. In this thesis, we do not discuss such issues.

Chapter 6

Numerical results

In this chapter, we show simulation results to see the behavior of our proposed algorithm. Simulations were run with the MATLAB environment and carried out on a 3,4 GHz, Intel Core i7-2600 personal computer.

At the beginning of every time slot $t \in \{1, \ldots, T\}$, the total number of scheduled vehicles is assumed to be fixed, which is denoted by N and the total number of free parking slots is assumed to be fixed, which is denoted by M. Of course, the algorithm is not restricted to fixed scenarios. Fixing N and M as mentioned above is useful to see the key behaviors of the proposed method. At the beginning of every time slot t, the α-scaled distances $a_{ij}(d_i)$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ were randomly generated. Note that α here is an arbitrary scalar known to the scheduled vehicles only. Without loss of generality, we let $\alpha = 1$ in all the simulations. In each time slot, the proposed algorithm is carried out for $K = 300$ subgradient iterations. Moreover, we average over $T = 1000$ time slots to obtain average performances of the algorithm. Finally and most importantly, for every $t \in \{1, \ldots, T\}$, we keep track of the *best* point found so far by the algorithm during its subgradient iterations $k = 1, \ldots, K$. We denote by $\mathbf{x}(t, k)$ the best point and by $p(t, k)$ the corresponding overall objective value (i.e., the primal objective value of problem (5.3.0.1-5.3.0.4)) in time slot t and in subgradient iteration k. Recall from section 5.5.2 that, we do not use any subroutines to construct a primal feasible point at the end of the proposed algorithm. As a result, of course, the best point $\mathbf{x}(t, k)$ found so far can be infeasible to the original primal problem (5.3.0.1-5.3.0.4). Therefore, if a primal feasible point is not attained by the algorithm in time slot t and in subgradient iteration k, then we set the corresponding $p(t, k) = \infty$.

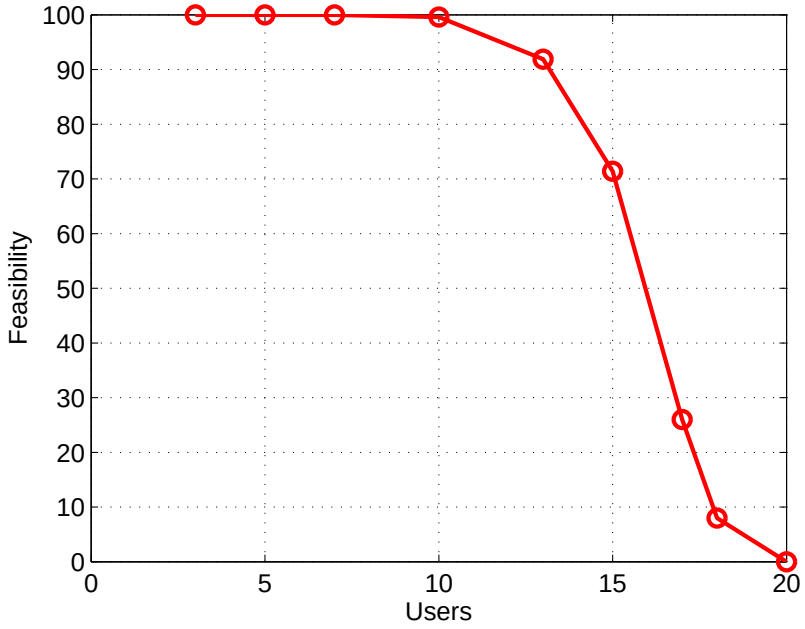

Figure 6.1: Feasibility versus vehicles or users; fixed number of parking slots, i.e., $M = 20$

6.1 Feasibility of the proposed method

First we show results relating to the solution's feasibility. In particular, we define a percentage measure of feasibility as follows:

$$\text{Feasibility} = \frac{\sum_{t=1}^{T} I(p(t,K) < \infty)}{T} \times 100\ \% \ , \tag{6.1.0.1}$$

where $I(E)$ is the indicator function of event E and recall that $K = 300$ is the total subgradient iterations and $T = 1000$ is the total time slots considered.

Figure 6.1 illustrates the change in the feasibility for fixed number of parking slots (i.e., $M = 20$) as the number of users N is increased from 3 to 20. The behavior is intuitively expected, because we have not implemented any subroutine to construct a feasible point at the end of the algorithm. However, results show that for all $N \leq 10$, the algorithm often finds a feasible point, even without such subroutines.

Figure 6.2 shows the change in feasibility for fixed number of users (i.e., $N = 3$) as the number of parking slots M is increased from 3 to 20. Results agree with out intuition: as the number of free slots increases, the feasibility increases.

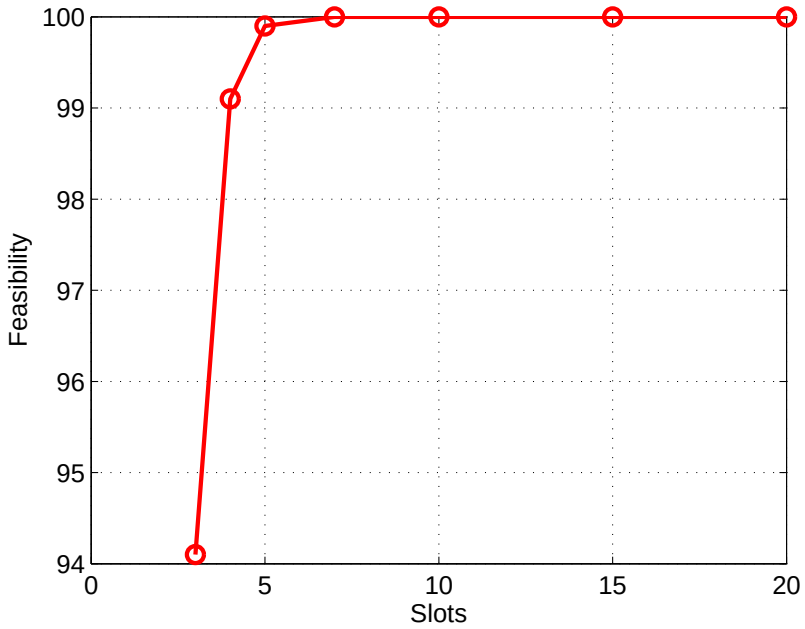

Figure 6.2: Feasibility versus parking slots; fixed number of vehicles or users, i.e., $N = 3$

Roughly speaking, the results in figure 6.1 and figure 6.2 indicate that as long as $N \leq (M/2)$, our proposed algorithm very likely returns a feasible point. Therefore, to further investigate the behavior of the proposed algorithm, in the following experiments, we stick to N and M values as mentioned above. As we already mentioned, when the resulting point is infeasible, we must rely on a subroutine to construct a primal feasible point. During this thesis those issues are not considered and are left for future research.

6.2 Comparison with other benchmarks

For evaluating the performance of our algorithm we compare it with the following benchmarks:

- **Random method**: This method employs a random parking slot assignment to each vehicle.
- **Greedy method**: This method employs a greedy parking slot assignment as follows. First, the vehicles are ordered. The first vehicle in the top of the order chooses its optimal parking, i.e., the parking slot

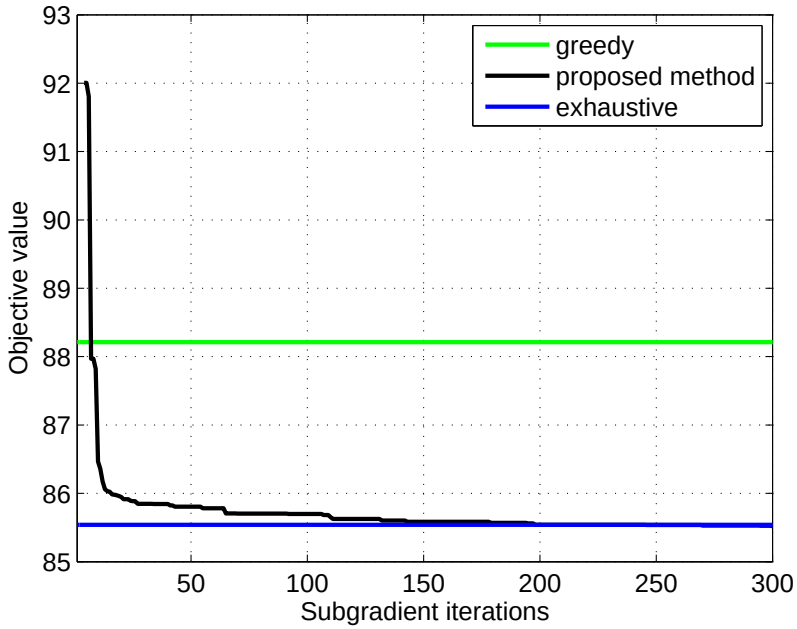

Figure 6.3: Average objective value $p(k)$ versus subgradient iterations k; $N = 3$ and $M = 20$.

closest to its destination. Then the next vehicle in the order looks in to the remaining free parking slots and selects the best one for him. The method continues until all scheduled vehicles have there own parking slots.

- **Exhaustive method**: This method computes all the combinations and finds the optimal parking slot assignment.

For comparison, we first consider the following performance metric:

$$p(k) = \sum_{t=1}^{T} p(t,k), \qquad k = 1, \ldots, K = 300 \ . \tag{6.2.0.1}$$

The metric $p(k)$ is a measure of the average objective value at subgradient iteration k.

Figure 6.3 shows the average objective value $p(k)$ of our proposed method versus the subgradient iteration k. Results are plotted for $N = 3$ and $M = 20$. For comparison, we have also plotted the average objective values obtained by benchmark algorithms. Note that the benchmark plots are straight lines, because they do not have subgradient iterations as our proposed method. Results show that the performance gap between our proposed method and the optimal exhaustive method become almost zero for larger

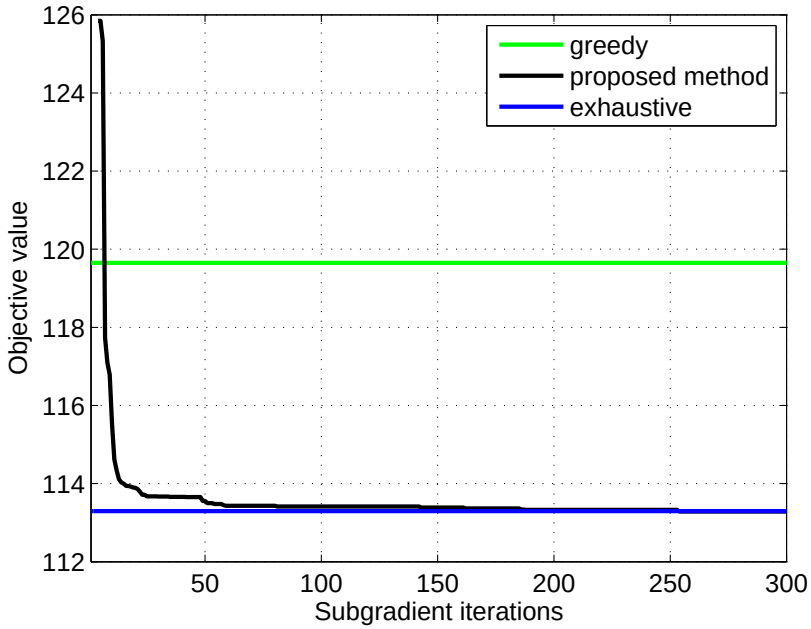

Figure 6.4: Average objective value $p(k)$ versus subgradient iterations k; $N = 3$ and $M = 15$.

subgradient iterations k. For example, see $p(k)$ for $k \geq 200$. Results further show that our method outperforms both the greedy and the random method [1]

Figure 6.4, 6.5, and 6.6 show again the average objective value $p(k)$ of our proposed method versus the subgradient iteration k for cases 1) $N = 3$ and $M = 15$, 2) $N = 3$ and $M = 10$, and 3) $N = 3$ and $M = 5$, respectively. Benchmark curves are plotted for comparisons. Note that there are some missing parts of the curves corresponds to our algorithm, see figure 6.5 and 6.6. For example, in the case of figure 6.5, curve that corresponds to our method starts at $k = 50$. This behavior is due to the primal infeasibility: i.e., for all $k \leq 50$, the point $\mathbf{x}(t, k)$ is infeasible for some time slot $t \in \{1, \ldots, T = 1000\}$. As a result, $p(k) = \infty$ for all $k \leq 50$ and not plotted in the figure. The performances in all considered cases are very similar to those discussed in the case of Figure 6.3. Our algorithm achieves almost optimal value for large values of k. Moreover, after some subgradient iterations, our proposed method outperforms both the greedy and the random methods.

[1] Curve that corresponds to the random method is excluded from the figure for clarity beacause the average objective value given by the random method is too high.

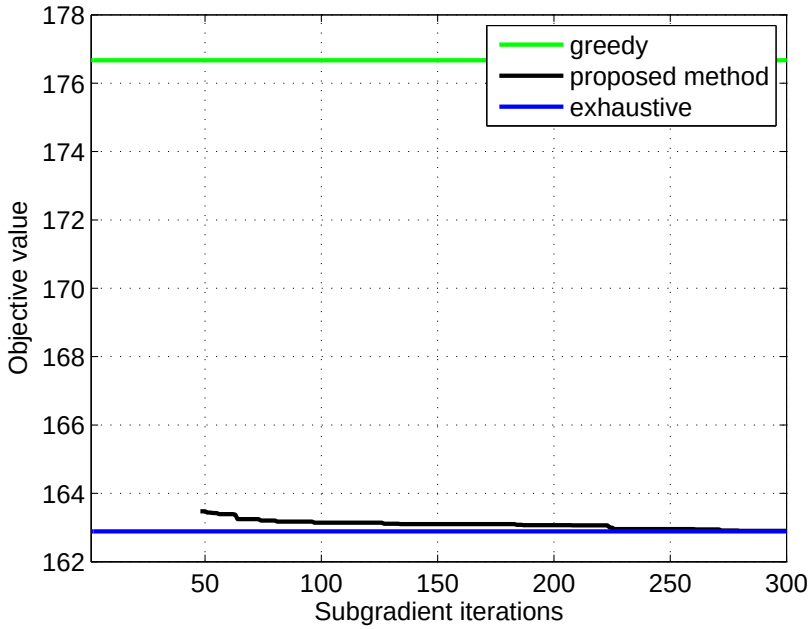

Figure 6.5: Average objective value $p(k)$ versus subgradient iterations k; $N = 3$ and $M = 10$.

Next, we define the following performance metric:

$$\text{Deviation with respect to optimal method } (D_{\text{opt}}) = \frac{X - p^*}{p^*} \times 100 \ \% \ , \tag{6.2.0.2}$$

where p^* is the optimal objective value obtained from the exhaustive method and X can be either the average objective value obtained from the random method, the greedy method, or our proposed method. In particular, we let $X = p(K)$ for our method.

In tables 6.1, 6.2 and 6.3 we illustrate the exact values of the objective function in random, greedy and proposed methods and also the deviation value from the D_{opt}.

In particular, table 6.1 shows the performance comparisons of different algorithms for the case $N = 3$ and $M = 20$. We have tabulated the the metric D_{opt} defined above together with the associated suboptimal objective values of random, greedy, and proposed methods. We can see that the objective value of the random method is too high, with a deviation D_{opt} of $777,14\%$. The greedy method, instead, has an acceptable objective value, with a deviation D_{opt} of $3,14\%$. But proposed method outperforms both methods above, infact its deviation from the optimal is zero.

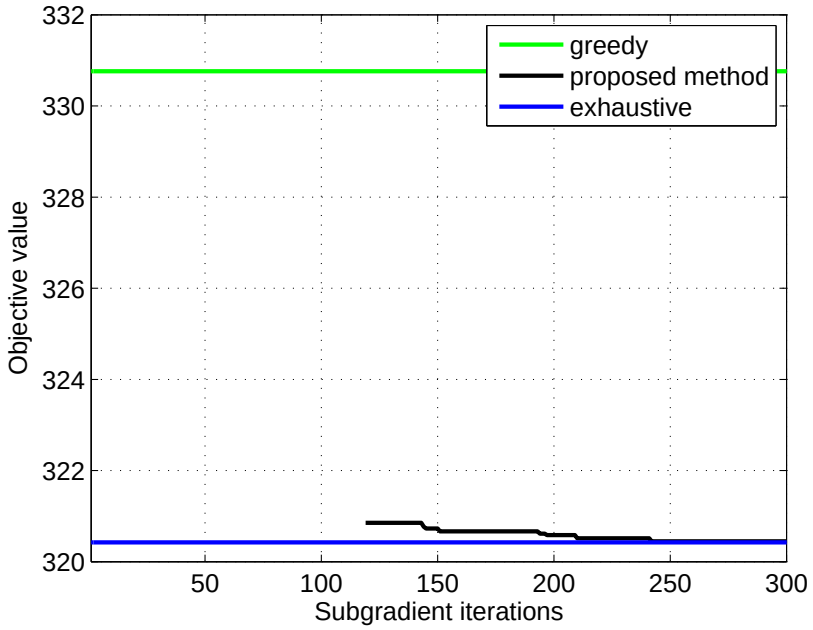

Figure 6.6: Average objective $p(k)$ versus subgradient iterations k; $N = 3$ and $M = 5$.

Method	**AVG of Objective function**	**Deviation from optimal (%)**
Random	$750,5491$	$777,14\%$
Greedy	$88,2119$	$3,14\%$
Proposed	$85,5397$	$\cong 0\%$

Table 6.1: Achieved objective value and deviation D_{opt}; $N = 3$ and $M = 20$

Method	**AVG of Objective function**	**Deviation from optimal (%)**
Random	$838,2$	$682,6\%$
Greedy	$117,2$	$9,5\%$
Proposed	$108,1$	$0,9\%$

Table 6.2: Achieved objective value and deviation D_{opt}; $N = 5$ and $M = 20$

Table 6.2 shows again the objective value and performance metric D_{opt} comparisons for the case $N = 5$ and $M = 20$. The results are very similar to those observed in table 6.1. We can note that the deviation of the random method is still unacceptable. The deviation of the greedy method has also been increased substantially from 3.14% to 9.5%. However, interestingly, the change in deviation of our proposed method is very small.

Now we consider a larger dimensional problems in which the exhaustive method is not applicable. So we consider a new deviation metric, $D_{\text{non-opt}}$, defined as:

$$\text{Deviation with respect to proposed method } (D_{\text{non-opt}}) = \frac{X - p(k)}{p(k)} \times 100\,\%\,, \tag{6.2.0.3}$$

where $p(K)$ is the average objective value from our proposed method after K=300 subgradient iterations and X can be either the average objective value obtained from the random method or the greedy method.

Table 6.3 shows the objective value and performance metric $D_{\text{non-opt}}$ comparisons for a larger dimensional problem. In particular, we consider a case with $N = 10$ and $M = 20$. The objective value of the random method is again unacceptable, with a deviation of $539,3\%$. Also, we can see that the deviation of the greedy method increases substantially, $26,32\%$.

6.3 CPU time comparison

Now we compare the CPU time performances of proposed method compared to the exhaustive method. Note that proposed method uses K = 300

Method	**AVG of Objective function**	**Deviation from proposed method's optimum (%)**
Random	$908,5$	$539,3\%$
Greedy	$179,5$	$26,32\%$

Table 6.3: Achieved objective value and deviation $D_{\text{non-opt}}$; $N = 10$ and $M = 20$

subgradient iterations before the algorithm terminates. First we consider a smaller scale problem, where N=3 users and M=5,10,15, and 20 free parking slots. Table 6.4 shows the exact value of the CPU time and also the feasibility of our method. When the number of available parking slots are increased, the CPU time of the exhaustive method also increases. However, the CPU time of the proposed method almost remains constant. We can see the behaviour of the CPU time also in figure 6.7.

Number of available slots	**CPU time of exhaustive method**	**CPU time of our method**	**Feasibility of our method**
5	$7.9561e^{-0.04}$	$2.9432e^{-0.04}$	99.9%
10	0.0042	$2.7619e^{-0.04}$	100%
15	0.0121	$2.9340e^{-0.04}$	100%
20	0.0263	$3.1651e^{-0.04}$	100%

Table 6.4: CPU time of exhaustive and proposed method with N=3

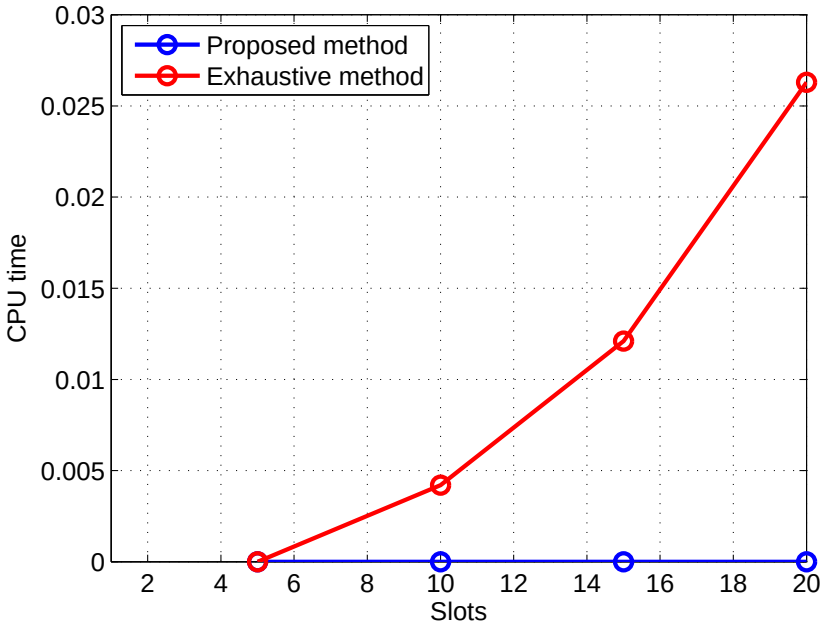

Figure 6.7: CPU time of exhaustive and proposed methods with N=3

Finally to highlight the computational complexity of exhaustive method with a large number of users, we consider a problem with N=5 users and M=5,10,15, and 20 available parking slots. In table 6.5 we can see that the required CPU time of the exhaustive method becames very high when the number of available slots increases. It is 0,281 seconds when M=5 and it becomes 40,4354 seconds when M=20. In particular there is an exponential growth in the CPU time required by the exhaustive method. However there is no such growth in the case of our proposed algorithm. Moreover, the time required by our proposed method is very small and shows a linear growth instead of an exponential growth. Thus the proposed method is scalable. The proposed method has a CPU time always less than 0 seconds and also, the feasibility increases when the number of slots increases. Figure 6.8 shows the above considerations more clearly.

Number of available slots	CPU time of exhaustive method	CPU time of our method	Feasibility of our method
5	$0,0281$	$0,1010$	$66,1\%$
10	$1,2469$	$0,1436$	$99,6\%$
15	$9,9211$	$0,1889$	100%
20	$40,4354$	$0,2428$	100%

Table 6.5: CPU time of exhaustive and proposed method with N=5

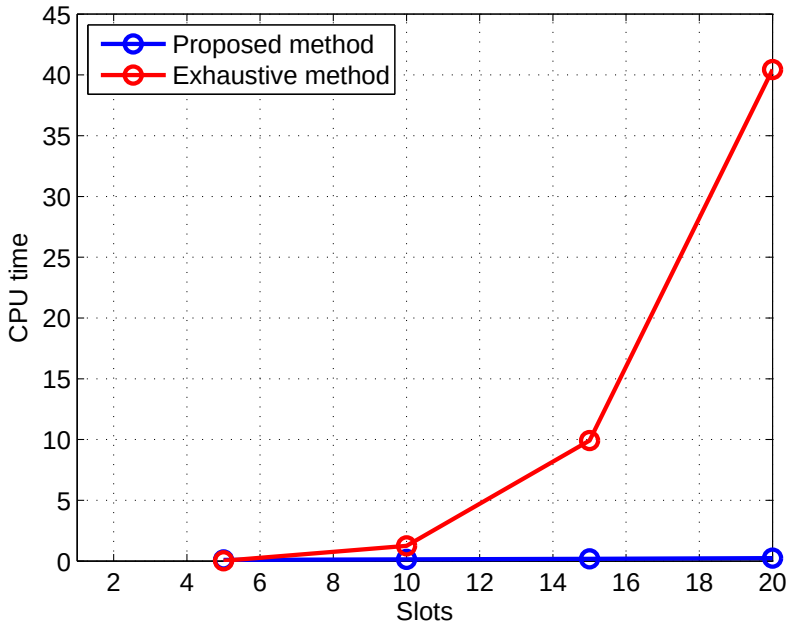

Figure 6.8: CPU time of exhaustive and proposed methods with N=5

Chapter 7

Conclusions

In this thesis we first discussed different approaches that are used in the privacy preserving optimization. In particular, we categorized these approaches into three main areas, cryptographic methods, transformation-based methods, and decomposition methods. The advantages and disadvantages of these approaches are identified. The solution approaches based on transformation based methods and decomposition based methods are very efficient compared to cryptographic variants. However the degrees of security and privacy guarantees offered by cryptographic solution methods are very high. Moreover, decomposition based approaches provide superior scalability properties compared to the other two categories. We also provided several other classifications based on the mathematical nature of the problem, the type of application, etc. Then we focused to a particular interesting problem: car parking slot assignment problem, where the objective was to minimize the maximum distance from the parking slots to the intended destinations of the cars, but without revealing anyone the destinations of the cars. We addressed this nonconvex problem by using an approach based on decomposition methods. We compared our algorithm with the optimal, random, and a greedy method and numerically evaluated the performance of the proposed algorithm, in terms of optimality and as well as the computational speed. Despite the reduced computational complexity of our proposed method, it provided close-to-optimal performance.

7.1 Limitations and future work

A limitation of our work is that the solution can be infeasible. With the proposed algorithm infact we find the optimal dual variable. The optimal solution for the dual problem is achieved, but we need to find the optimal for

the primal problem. Since we have a non convex problem, the duality gap between dual and primal solutions is not zero. So, is not guaranteed to obtain the primal optimal We need to construct the primal solution heuristically.

Therefore, unlike convex problems, there is no guarantee that from the dual optimal value and solution, we can construct the primal optimal value, primal optimal solution, or at least a primal feasible point. Thus, a potential future direction is to explore good heuristics to construct primal feasible points. Moreover, it would be interesting to explore the conditions under which zero duality holds for the original assignment problem or any other analytical performance guarantee which can describe the proposed algorithm's performance. It is also interesting to investigate the methods that we can use to quantify the security and privacy levels of decomposition based approaches.

Bibliography

[1] Stephen Boyd and Lieven Vandenberghe, *Convex Optimization*, Cambridge, 2004.

[2] Angelia Nedic and Asuman Ozdaglar, *Convex Optimization in Signal Processing and Communications*, Cambridge, 2009.

[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato and Jonathan Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*, Foundations and Trends in Machine Learning Vol. 3, No. 1 (2010) 1–122.

[4] O.L. Mangasarian, *Privacy Preserving linear programming*, Optimization Letters 5(1):165–172, 2011.

[5] Michael J. Neely, *Distributed and Secure Computation of Convex Programs over a Network of Connected Processors*, University of Southern California, Conference Guelph, Ontario, Canada, July 2005.

[6] Satish K. Sehgal and Asim K. Pal, *Privacy Preserving Decentralized Method for Computing a Pareto-Optimal Solution*, 7th International Workshop, Kharagpur, India, December 27-30, 2005.

[7] Cong Wang, Kui Ren and Jia Wang, *Secure Optimization Computation Outsourcing in Cloud Computing: A Case Study of Linear Programming*, Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, 2011.

[8] Feng Yan, Shreyas Sundaram, S.V. N. Vishwanathan and Yuan Qi, *Distributed Autonomous Online Learning: Regrets and Intrinsic Privacy-Preserving Properties*, arXiv:1006.4039v3 [cs.LG], 4 Feb 2011.

[9] Michael Kearns, Jinsong Tan and Jennifer Wortmani, *Privacy-Preserving Belief Propagation and Sampling*, Department of Computer and Information Science University of Pennsylvania, Philadelphia, PA 19104.

[10] Thomas Leaute, Brammert Ottens and Boi Faltings, *Ensuring Privacy through Distributed Computation in Multiple-Depot Vehicle Routing Problems*, Artificial Intelligence Laboratory (LIA), Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland.

[11] João F. C. Mota, João M. F. Xavier, Pedro M. Q. Aguiar and Markus Püschel, *D-ADMM: A Communication-Efficient Distributed Algorithm For Separable Optimization*, Fundação para a Ciência e Tecnologia (FCT): CMU-PT/SIA/0026/2009, PTDC/EEAACR/ 73749/2006, PEst-OE/EEI/LA0009/2011, and SFRH/BD/33520/2008.

[12] Carlo Fischione, *F-Lipschitz Optimization*, IEEE Transactions on Automatic Control, 2011.

[13] Stephen Boyd, Lin Xiao, Almir Mutapcic, and Jacob Mattingley *Notes on Decomposition Methods*, Stanford University, Notes for EE364B.

[14] Stephen Boyd, Lin Xiao, Almir Mutapcic, and Jacob Mattingley *Subgradient Methods*, Notes for EE392o, Stanford University, Autumn 2003.

[15] J. Li and M.J. Atallah *Secure and private collaborative linear programming*, In Proceedings of the International Conference on Collaborative Computing: Netowrking, Applications and Worksharing, pages 1–8, 2006.

[16] O. Catrina and S. de Hoogh *Secure multiparty linear programming using fixedpoint arithmetic*, In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, Computer Security – ESORICS 2010. LNCS, volume 6345, pages 134–150, Heidelberg, 2010. Springer-Verlag.

[17] T. Toft *Solving linear programs using multiparty computation*, In Roger Dingledine and Philippe Golle, editors, Financial Cryptography and Data Security. LNCS, volume 5628, pages 90–107. Springer, 2009.

[18] A. Arbel *Exploring interior-point linear programming: Algorithms and software*, MIT Press, 1993.

[19] R. J. Vanderbei *Linear programming: Foundations and extensions*, Third Edition. Springer, 2008.

[20] N. Karmarkar *A new polynomial-time algorithm for linear programming*, Combinatorica, 4(4):373–395, 1984.

[21] M. Asghar Bhatti *Practical optimization methods: with mathematica applications*, Springer-Verlag, 2000.

[22] A. Bednarz, N. Bean, and M. Roughan *Hiccups on the road to privacy-preserving linear programming*, In Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society, pages 117-120, 2009.

[23] W. Du *A study of several specific secure two party computation problems*, PhD thesis, Purdue University, West Lafayette, Indiana, 2001. 011.

[24] O.L. Mangasarian *Privacy-preserving horizontally-partitioned linear programs*, Technical report, Data Mining Institute Technical Report 10-02, April 2010.

[25] Alice Bednarz *Methods for Two-Party Privacy-Preserving Linear Programming*, PhD thesis in Applied Mathematics at The University of Adelaide, January 22, 2012.