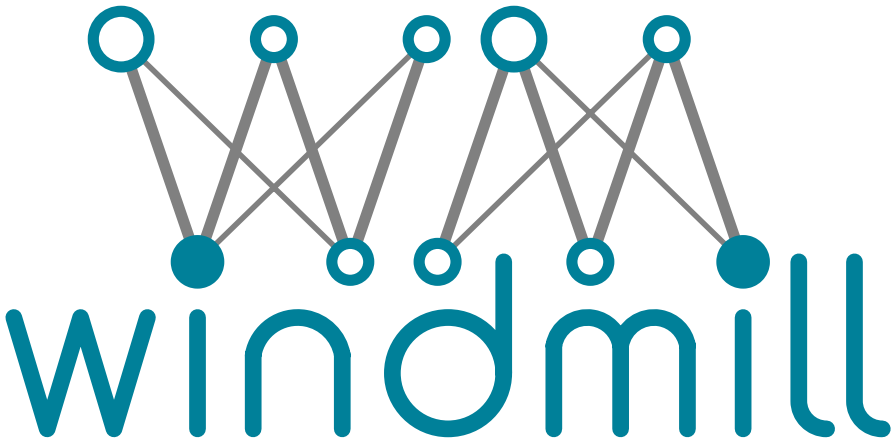
Marie Skłodowska Curie Action

WINDMILL

Machine Learning for Wireless Communications

H2020-MSCA-ITN-ETN

Grant Agreement Number: 813999



WP3–Advancing the field of ML for wireless communications

D3.2–Novel ML techniques for Wireless Network Optimization

Contractual Delivery Date:	August 31, 2021
Actual Delivery Date:	August 26, 2021
Responsible Beneficiary:	ETHZ
Contributing Beneficiaries:	ETHZ, Eurecom, CTTC
Dissemination Level:	Public
Version:	Final

PROPRIETARY RIGHTS STATEMENT
This project has received funding from the European UnionâĂŹs Horizon 2020 research and innovation programme under the Marie SkÅĆodowska-Curie grant agreement No 813999.

Document Information

Document ID:	WP3/D3.2
Version Date:	August 26, 2021
Total Number of Pages:	39
Abstract:	
Keywords:	Machine Learning, Kernel methods, Graph Neural Networks, MIMO Optmization

Authors

Full name	**Beneficiary/ Organisation**	**e-mail**	**Role**
Shirin Goshtasbpour	ETH Zurich	shirin.goshtasbpour@inf.ethz.ch	Contributor
Roberto Pereira	CTTC	rpereira@cttc.es	Contributor
Davit Gogolashvili	Eurecom	davit.gogolashvili@eurecom.fr	Contributor
Fernando Perez-Cruz	ETH Zurich	fernando.perezcruz@sdsc.ethz.ch	Coordinator

Table of Contents

List of Figures

1. Introduction to Novel ML techniques for Wireless Network Optimization

Machine Learning and Digital communications have fruitful history. When Neural Networks have their second coming in the late 1980s, there were several papers in which fully connected, wavelet neural networks, recurrent neural networks have been using for multiuser detection and channel equalization, for example [2–6], but not limited to. These papers show the interests of digital communication researchers in using the new technology to advance in relevant problem in digital communications. Another, even more relevant example, is the use of Belief Propagation for solving the all-relevant channel coding problem. David MacKay rediscovered Low-Density Partity-Check (LDPC) codes in mid-1990s [7], which were originally proposed by Gallager in his PhD thesis [8]. They were later shown that Irregular LDPC codes can achieve capacity for the Binary Erasure Channel (BSC) [9].

In this report, we have focused on two of the most promising techniques in machine learning nowadays that can have an impact in the developments of communications networks, and we have also worked on a relevant case study. First, Davit Gogolashvili has summarized recent advances in reinforcement learning and especially deep reinforcement learning. Reinforcement learning has been a well-known technique, but with the combination of deep NNs for designing the action-state models and the Q function it has opened many possibilities. Reinforcement learning can be applied to many different aspects of optimization in communication, from resource allocation in physical channels to network optimization.

In the second chapter, Shirin Goshtasbpour has introduced the new field of graph neural networks. NNs have had an impressive improvement over the years because their structure matched the type of data that they were trying to study. For example, convolutional filters are well-suited for natural images and transformer and attention networks have been discovered for texts data. Graph neural network generalizes those particular cases to any form of data. Wireless commutation networks can be better expressed using graphs, especially those deployed in the real world in which the hexagonal idealization is far from perfect. Graph neural networks are the future to understand wireless communications.

Finally, Roberto Pereira has dedicated time to explain how to use kernel methods to optimize massive MIMO systems. He has shown how complex kernel methods can be used to understand mMIMO and has applied hierarchical clustering to improve the communication networks.

2. Reinforcement Learning

2.1. Introduction

Reinforcement learning is an important area of machine learning with variety of practical applications including dynamic channel allocation algorithms [10] [11] robot control, board games such as backgammon [12],chess, go [13], elevator scheduling problems [14] and a number of other problems (see [15–18]).
The general scenario of reinforcement learning can be described as follows: an agent interacting with a dynamical environment have to learn behavior trough trial-and-error. Unlike the supervised learning scenario, the agent doesn't passively receive a labeled data set, he collects information through a course of actions by interacting with the environment. The objective of the agent is to determine the best course of actions to maximize long term objective. However, the information he receives from the environment is just immediate reward with no future or long term reward feedback. So the main dilemma agent faces is to exploit existing information about the environment or to explore unknown states and actions.
The remainder of this work is organized as follows. In section 2.2, we review basic facts about Markov Decision Process. Section 2.3. is devoted to the problem of finding optimal policy when the increment model is known. In section 2.4. we describe some model-free algorithms based on stochastic approximation methods. In section 2.5. we mention few challenges of modern reinforcement learning.

2.2. Markov Decision Process

Markov decision process is defined as a triplet $(\mathcal{S}, \mathcal{A}, \mathcal{P}')$ where $\mathcal{S}$ is a set of countable states, A is a set of countably many actions. For each pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ we assign *transition probability measure* $\mathcal{P}'$ over $\mathcal{S} \times \mathbb{R}$ which gives the probability of the next state and reward being in some set $C \in \mathcal{S} \times \mathbb{R}$ provided that the current state is s and the action taken is a. To emphasize dependency of the measure on (s, a) we use the notation $\mathcal{P}'(\cdot \mid s, a)$.
The transition probability measure gives rise to other important quantities related to the environment. Most important ones are *expected reward function* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ giving the expected immediate reward received when action a is chosen in state s:

$$r(x, a) = \mathbb{E}\left[R_{(x,a)}\right], \quad \left(Y_{(x,a)}, R_{(x,a)}\right) \sim \mathcal{P}_0(\cdot \mid x, a), \tag{2.1}$$

and *state-transition probabilities*

$$\mathcal{P}(s, a, s') = \mathcal{P}'(\{s'\} \times \mathbb{R} \mid s, a) \tag{2.2}$$

which gives the probability of moving from state s to some other state s' provided that action a was chosen in state s.
In the standard reinforcement learning model, MDP was introduced to model the environment and interaction with the environment as follows: On each step of interaction n, the agent observes the current random state $S_n \in \mathcal{S}$ and takes an action $A_n \in \mathcal{A}$. In particular $\mathbb{P}(S_{n+1} = s' \mid S_n = s, A_t = a) = \mathcal{P}(s, a, s')$ holds for any $s, s' \in \mathcal{S}, a \in \mathcal{A}$. Further,

$\mathbb{E}[R_{n+1} \mid S_n, A_n] = r(S_n, A_n)$ The action changes the state of the environment to $S_{n+1} \in \mathcal{S}$, and the value of this state transition is communicated to the agent through a scalar reinforcement signal, $R_{n+1} \in \mathbb{R}$. This transition to (S_{n+1}, r_{n+1}) from state S_t after action A_n happens with probability $\mathcal{P}'(\cdot \mid S_n, A_n)$. The agent then observes the next state S_{n+1} and reward r_{n+1}, chooses a new action $A_{n+1} \in \mathcal{A}$ and the process is repeated. The goal of the agent is to choose actions that tend to increase the long-run sum of values of the reinforcement signal. Mapping π which assigns probability distribution supported on the action space $\mathcal{A}$ to the state $s \in S$ is called *policy*. A policy π is *deterministic* if for each s, it assigns unique $a \in \mathcal{A}$ with probability one. In that case, we can identify π with a mapping from $\mathcal{S}$ to $\mathcal{A}$ and use $\pi(s)$ to denote that action.
The agent's objective is to find a policy that maximizes its expected reward. The return it receives following a deterministic policy π along a specific sequence of states $s_0, \ldots, s_T$ is defined as follows:

$$\sum_{t=0}^{T-1} \gamma^t r(s_t, \pi(s_t)) + \mathbb{I}(T < \infty)\gamma^T r(s_T, \pi(s_T)).$$

where $\gamma \in [0, 1)$ is a constant factor less than one used to discount future rewards.
The *value function*, $V^\pi : \mathcal{S} \to \mathbb{R}$, underlying π is defined as the expected reward returned when starting at s and following policy π

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s\right], \quad s \in \mathcal{S}.$$

A policy π^* is optimal if its value is maximal for every state $s \in S$, that is, for any policy π and any state $s \in S$, $V^{\pi^*}(s) \geq V^\pi(s)$. Underling value function $V^* : \mathcal{S} \to \mathbb{R}$ is called *optimal value function.*
The *state-action value function* Q associated to a policy π is defined for all $(s, a) \in S \times A$ as the expected return for taking action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$ and then following policy π :

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a\right] \tag{2.3}$$

$$= r(s, a) + \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a\right], \quad s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.4}$$

The optimal state-action value function $Q^*(s, a)$ at the state-action pair (s, a) is defined as the maximum of the expected return under the constraints that the process starts at state s, and the first action chosen is a. The underlying function $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is called the *optimal action-value function*.
The optimal value- and action-value functions are connected by the following equations:

$$V^*(s) = \sup_{a \in \mathcal{A}} Q^*(s, a), \qquad s \in \mathcal{S},$$

$$Q^*(s, a) = r(s, a) + \gamma \sum_{u \in \mathcal{S}} \mathcal{P}(s, a, y) V^*(y), \quad s \in \mathcal{S}, a \in \mathcal{A}.$$

Bellman's optimality condition: A policy π is optimal iff for any pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ with $\pi(s)(a) > 0$ the following holds:

$$a \in \operatorname*{argmax}_{a' \in \mathcal{A}} Q^\pi(s, a').$$

From Bellman's optimality condition if follows that

$$\pi^*(s) = \underset{a\in\mathcal{A}}{\operatorname{argmax}} Q^*(s,a), \quad \forall s \in S.$$

Thus, the knowledge of the state-action value function Q^* is sufficient for the agent to determine the optimal policy, without any direct knowledge of the reward or transition probabilities. Similarly, knowing V^*, r and $\mathcal{P}$ also suffices to act optimally.
Question we try to address is how to find V^* and Q^*.

2.3. Finding a Policy Given a Model

In this section, we assume that the environment model is known. In this section, we assume that the environment model is known. That is, the transition probability $\mathcal{P}$ and the expected reward r for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ are assumed to be given. *Dynamic programming techniques* used to determine optimal policy will serve as the foundation and inspiration for the learning algorithms to follow.
Let's first find optimal value function V^*. As we already know optimal value function is unique and can be defined as the solution of the nonlinear system of equations

$$V^*(s) = \sup_{a\in\mathcal{A}} \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') V^*(s') \right), \quad a \in \mathcal{A}. \tag{2.5}$$

Defining the Bellman optimality operator, $T^* : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$, by

$$(T^*V)(s) = \sup_{a\in\mathcal{A}} \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') V^*(s') \right), \quad a \in \mathcal{A},$$

we can rewrite equation () compactly as

$$T^*V^* = V^*.$$

Given the optimal value we can spacify the optimal policy as

$$\pi^*(s) = \underset{a\in\mathcal{A}}{\operatorname{argmax}} \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') V^*(s') \right).$$

2.3.1. Value iteration and Policy iteration

Value iteration seeks to determine the optimal policy by generating value functions iteratively

$$V_{n+1} = T^*V_n, \quad n \geq 0,$$

where V_0 is arbitrary. Banach's fixed point theorem [1] guarantees convergence to V^* at geometric rate [19, 20].

[1] To apply Banach's fixed point theorem one needs to show that the operator T^* is a contraction mapping from $\mathbb{R}^{|\mathcal{S}|}$ to $\mathbb{R}^{|\mathcal{S}|}$ in some norm, usually in $\|\cdot\|_\infty$-norm.

The *policy iteration* algorithm manipulates the policy directly using policy evaluation, which can be achieved by solving system of linear equations

$$V^{\pi}(s) = r(s,a) + \gamma \sum_{s' \in \mathcal{A}} \mathcal{P}(s,a,s') V^{\pi}(s'), \quad a \in \mathcal{A}.$$

Starting with an arbitrary action policy π_0, the algorithm repeatedly computes the value of the current policy π via that matrix inversion and greedily selects the new policy by maximizing $T^* V^{\pi}$.
It can be shown that the policy iteration converges in a smaller number of steps than the value iteration algorithm. But, value iteration is much faster per iteration since for the policy iteration algorithm requires solving a system of linear equations, which is more expensive to compute than an iteration of the value iteration algorithm.

2.4. Learning an Optimal Policy-Model-free Approach

Now let's consider more realistic in practice scenario, when the environment model of an MDP is unknown, that is we don't know transition probability $\mathcal{P}$ and expected reward function r. As was discussed earlier, finding the optimal value function is related to the fixed point of operator T^* which is not directly accessible. Algorithmic methods adopted for this situation are closely related to the concepts and techniques in *stochastic approximation*. Let's first introduce the following version of *law of large numbers*.
Theorem 1. Let $X_1, \ldots, X_n$ be i.i.d copies of random variable X, taking values in $[0,1]$. Than the sequence

$$\mu_{n+1} = (1 - \alpha_n)\mu_n + \alpha_n X_n$$

converges to the expected value of X almost surely under the assumption [2] that

$$\sum_{i=1}^{n} \alpha_i = \infty \quad \text{and} \quad \sum_{i=1}^{n} \alpha_n^2 < \infty.$$

2.4.1. TD(0) algorithm

The algorithm is based on the system of linear equations giving the value of a policy π

$$V^{\pi}(s) = \mathbb{E}\left[R_{n+1} + \gamma V^{\pi}(S_{n+1}) | S_n = s\right].$$

Motivated by the theorem 1 sequence we should construct in order to converge to the expectation above should be

$$V_{n+1}(s) = V_n(s) + \alpha_n(R_n + \gamma V_n(S_{n+1}) - V_n(s)).$$

The convergence of the algorithm can be proved using theorem similar to theorem 1 (ref).

[2] This assumption is known as *Robbins-Monro* condition.

2.4.2. Q-learning

The Q-learning algorithm is based on the equations giving optimal state-action value function Q^*:

$$Q^*(s,a) = \mathbb{E}\left[R_{n+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{n+1}, a') | S_n = s, A_n = a\right]$$

The Q-learning rule is the same as $TD(0)$ and is given by

$$Q_{n+1}(s,a) = Q_n(s,a) + \alpha_n(R_{n+1} + \gamma \max_{a' \in \mathcal{A}} Q_n(S_{n+1}, a') + Q_n(s,a))$$

The techniques used in TD-learning and Q-learning are closely related to those of stochastic approximation originated with the work of Robbins and Monro [21], followed by a series of results including Dvoretzky [22], Kiefer and Wolfowitz [23]. For a recent survey of stochastic approximation see [24] and the references therein. The connection with stochastic approximation was emphasized by [25, 26], who gave a proof of the convergence of Q-learning. For the convergence rate of Q-learning, consult [27]. For recent results on the convergence of the policy iteration algorithm, see [28].

2.4.3. Dealing with Large State Space

In practice the number of states can be very large (or infinite) and it is not feasible to keep a separate value for each state in the memory. Instead of storing the table of value function one can use approximate value function $V_\theta(s)$ with the parameters $|\theta|$ much smaller than the number of states. Most common representation of $V_\theta(s)$ is

$$V_\theta(s) = \sum_{i=1}^{d} \theta_i \phi_i(s)$$

and is discussed in [29]. The case when $\phi_i(s)$ forms Fourier basis was introduce in [30].
Another popular function for approximation is artificial neural network (ANN) first used in reinforcement learning is the work by Farley and Clark [31]. For the review of applications of ANN in reinforcement learning see [32].

2.5. Challenges

One critical challenge in RL is to *design a reward signal* [33] so that as an agent learns, its behavior approaches what the application's designer actually desires. The reward function is another hyperparameter that has to be set at the start.
Another important challenge is to *extend the capabilities* of basic RL systems. Some of the most dramatic results have been achieved by combining RL with other methods, such as deep neural networks and Monte Carlo tree search. Bayesian Reinforcement Learning which leverages methods from Bayesian inference extends the capabilities of basic RL systems in important directions. Other extensions are perhaps more correctly viewed as enhancements of the RL framework itself, as seen in the development of hierarchical RL, RL for partially observable MDPs, and ways of handling continuous state and action spaces [34].

3. Graph Neural Networks and Their Application in Wireless Communication

3.1. Introduction

In many applications, data can be naturally modeled in the form of a graph $G = (V, E)$ where V is the set of nodes or vertices and E is the set of edges with elements $e \in E$ consisting of tuples of graph vertices. Graphs are commonly used to model molecular connections in chemistry or biology [35–37] or interactions in particle systems in physics problems [38, 39]. In Natural Language Processing (NLP) it is prevalent to augment the text samples with dependency trees and semantic roles and relations [40–42]. Mined data is often used in form of knowledge graphs for few-shot or zero-shot learning [43–45]. User interactions in social networks [46, 47] and their interests in recommender systems are also inherently modeled with a set of nodes and edges [48]. It is easy to consider scenarios in wireless networks where end-to-end connection or pointwise links in legacy or Software Defined Networks (SDN) [49, 50], service input-output relations in datacenters or Network Function Virtualization (NFV) [51], network traffic routing [52, 53], as well as allocation of resources to services or users can be formulated as graph components [54–56].

In other cases where data is not inherently structured, it might be beneficial to extract structured data representation from the samples instead of latent representations that can only make sense in regular Euclidean spaces. This can be viewed as enforcing a domain-expert bias in the inference model to recover the most valuable information for a given task. For example, in computer vision (CV) visual reasoning and semantic segmentation are two such tasks where the state-of-the-art relies on extracting the relation of objects and pixels in the images, respectively.Similarly, in NLP graphs are used to formalize various syntactic or adjacency dependencies between words and phrases in given texts or for the purpose of reading comprehension and question answering.While, modeling wireless networks with graphs has been popular for over two decades, structured inference has received little attention in comparison. However, it is reasonable to believe that with the booming use of Machine Learning (ML) tools in wireless communication, these methods will also be adopted for compression or network tomography.

The main idea behind Graph Neural Networks (GNN) is building a state transition system on graphs by assigning state variables to nodes or edges and defining a state update rule for each graph entity. Then iteratively apply the update rule until convergence of the values. For instance, Interaction Networks (IN) encode each vertex or edge development in a dynamic system with DNNs [57] whereas CommNets only used the 1-hop neighborhood of each node to get the updated node states [58]. Visual IN (VIN) further combines IN architecture with Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) for better parameter sharing. The first two methods are not computationally efficient as the number of parameters grows linearly with the number of nodes and edges in the input graphs. In addition, it is not straightforward to extend these architectures to parse graphs with more complex structures[1] or have them process graphs with different number of vertices or of varying degrees.

[1] Graph complexity is detailed in Section 3.2

Although, RNNs and Feedforward Networks have higher expressivity, naive implementation of these architectures assume a partial ordering in the inputed nodes of the graph and are unable to engage all the information contained in a graphical structure efficiently. CNNs on the other hand add to the model's flexibility by learning multi-scale local spatial features and their compositions, are permutation equivariant and were proven to generalize well to undirected graphs and different graph scales.
Another application of parsing graph data is to learn low-dimensional euclidean vectors to represent, compare or cluster graph structures, nodes, edges or subgraphs. Graph embeddings were first introduced in [59] in DeepWalk algorithm by passing the state of random walks on sample graphs to the SkipGram model similar to the direct methods used for word embedding extraction in NLP. Graph Auto-Encoder architectures, on the other hand, unify the graph embedding and optimization task with higher flexibility [60].
The rest of this article is organized as follows: in Section 3.2 we describe different graph types with their additional complexities. In Section 3.3 we explain the details of computational approaches and modules used in GNNs to deal with these structural diversities. The common training objectives and optimization process of the GNNs are discussed in Section 3.4. Finally, in Section 3.5, we review GNN applications in wireless networks.

3.2. Graph types

Graphs are unique non-Euclidean structures which can express entities and their relations in a large number of systems in various areas. Based on the application, different graphs can be used to express the data structure inducing an implicit bias on our model such as partial ordering or relative importance or development progress of events in a system. Each of these forms have different complexity level which manifests in form of constraints in the optimization problem. The additional information in graph formalism can be used to achieve better classification, clustering, or prediction at node, edge, subgraph or graph level. The GNN model is designed accordingly to deal with the entailing complexities. Therefore, it is important to find the correct graph type to represent the data in a given problem. We aim to detail the differences of some of the more common graph types in this section which are currently used or thought suitable for wireless system modeling.

Undirected vs. directed graphs

We denote an undirected graph with $G = (V, E)$ where $E = \{(v_i, v_j) \in V^2 | 1 \leq i \leq j \leq |V|\}$. On the other hand, in a directed graph we have edges with orientation and $E = \{(v_i, v_j) \in V^2 | i, j \in [|V|]\}$ where the edge $e_{ij} = (v_i, v_j)$ is directed from v_i to v_j. Directed graphs are used to define partial ordering on the nodes and have a more elaborate definition for cycles, cliques and connected components which all depend on the direction of the edges.

Hypergraphs

A hypergraphs is a generalization of a graph where each edge $e_{\underline{i}} \in E$ connects $n \geq 2$ number of nodes together. The edges in a undirected hypergraph are denoted with unordered tuples $e_{\underline{i}} = (v_{i_1}, ..., v_{i_n})$ for $\underline{i} = (i_1, ..., i_n)$ and $1 \leq i_1 \leq ... \leq i_n \leq |V|$ and in a directed hypergraph

we denote the edges with ordered tuples. Hypergraphs can be used in formulation of groups of users or workloads requesting a similar service or resource in the wireless network.

Homogeneous and heterogeneous graphs

In homogeneous graphs we assume the nodes and edges have the same type, while in heterogeneous graphs we have nodes and edges of possibly different types. We denote a heterogeneous graph with $G = (V, E, \phi, \psi)$ where $\phi : V \to \mathbb{N}$ and is a mapping that associates each node $v_i \in V$ with type $\phi(v_i)$ and $\psi : E \to \mathbb{N}$ is a mapping $e_{ij} \mapsto \psi(e)$ that does the same for the edge $e_{ij} \in E$. In general, treating a heterogeneous graph as a number of homogeneous graphs results in the loss of information about dependencies between different edge and node types. Therefore, we need the GNN to account for the types and process the heterogeneous graph globally. This structure can efficiently be utilized to represent concurrent and different types of services and communication requirements in a wireless network.
A meta-path is a substructure of graph which is constructed by denoting the type of nodes and edges along a corresponding path in the graph. For example, a graph containing the path $v_1 \xrightarrow{e_{13}} v_3 \xrightarrow{e_{35}} v_5 \xrightarrow{e_{52}} v_2$ has a meta-path denoted by $\phi(v_1) \xrightarrow{\psi(e_{13})} \phi(v_3) \xrightarrow{\psi(e_{35})} \phi(v_5) \xrightarrow{\psi(e_{52})} \phi(v_2)$. Augmenting the graph data structure with its meta-paths is investigated in [61, 62].

Static vs. dynamic graphs

In general each of nodes or edges may have a weight, feature vector, or label assigned to them. If the graph topology or the assigned values vary with time, the graph is called a dynamic graph. Unless the studied system is memoryless, the temporal information in dynamic graphs should be regarded as the spatial information to allow the model to discern between graph sequences which are different permutations of the same set of graphs. This is particularly useful aspect of spatio-temporal GNNs which enables them to model, for instance, the time dependent characteristics of workloads wireless networks.
Dynamic graphs can be processed by first extracting the spatial information using GNNs and then passing the model output to sequence models like RNNs [63, 64]. Alternatively, we can perform dynamic state message propagation by extending the existing propagation modules to collect temporal messages [65, 66].

3.3. Computational approaches

As was mentioned before, the main idea behind GNNs is to put a state transition system on the graphs and iteratively update the states until convergence. GNNs are typically composed of four types of modules which we describe further in the following sections. In GNNs, these modules are typically stacked together to form a high level representation of the graphs. A general GNN architecture is depicted in Figure 3.1. Another method is to use Ordinary Differential Equations (ODE) to define the state update operators resulting in continuous-time GNN models. In ODE-GNNs we use numerical integrators instead of the backpropagation through a finite number of layers to find the optimal parameters [67].
We use $f^{(l)} \in \mathbb{R}^{|V| \times M}$ to denote the state of graph nodes with l indexing the layer.

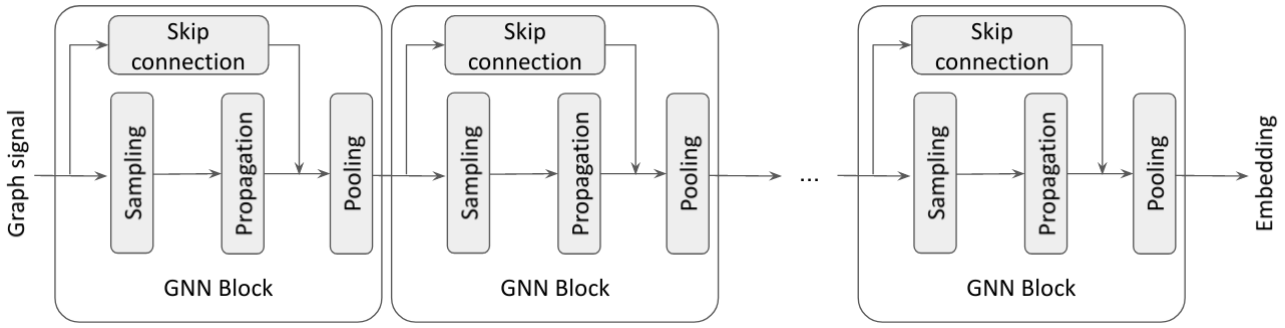


Figure 3.1: GNN architecture with propagation, sampling, pooling operators and skip connections. Propagation module parameters are shared between the blocks in RNN based GNNs and hidden state variables are passed to the next block along with the extracted feature. Some implementations omit sampling or pooling modules or skip connections. GNNs generate node, edge or higher level embeddings depending on the application which is then used classify or cluster the entities and train the parameters.

3.3.1. Convolutional propagation modules

Propagation modules are used to update the state information based on current state of the corresponding entity and its k-hop neighbors. Common propagation modules are recurrent operator, convolution operator and attention mechanism which are good in reusing model parameters and finding frequent patterns in the data.
GNNs use a convolution operator that is the generalization of Euclidean convolution filters to the graph domain.

Spectral convolution

Emerging from graph signal processing theory, many methods parameterize the spectral form of the convolution filter to control the information propagation in the graph using Graph Fourier Transformation (GFT) of the graph signal f (i.e. vector formed from mapping graph vertices to real numbers). Let $\mathcal{F}$ be the GFT for graph G with normalized Laplacian $L = I_{|V|} - D^{-1/2}AD^{-1/2}$ where D is the degree matrix and A is the adjacency matrix of the graph. Using the eigenvalue decomposition $L = U\Lambda U^T$ for the diagonal ordered eigenvalue matrix Λ and matrix of eigenvectors U, the Fourier transformation and its inverse applied on $\hat{f} \in \mathbb{R}^{|V|}$ is given by

$$\mathcal{F}(f) = U^T f \text{ and } \mathcal{F}^{-1}(\hat{f}) = U\hat{f}$$

And the convolution operator with filter $g : V \to \mathbb{R}$ is equivalent to

$$g * f = \mathcal{F}^{-1}(\mathcal{F}(g) \odot \mathcal{F}(f)) = U(U^T g \odot U^T f)$$

where $\odot$ denotes Hadamard product. Spectral ConvGNN define the convolution filters $\hat{g}_\theta = U^T g \in \mathbb{R}^{c_i \times c_o}$ in the spectral domain as a trainable set of parameters for c_i input c_o output channels [68]. The spectral convolution layer operation output for channel m is evaluated by

$$f_m^{(l+1)} = \sigma\left(\sum_{n=1}^{c_i} \left(g_\theta^{(l+1)}\right)_m * f_n^{(l)}\right) = \sigma\left(U \sum_{n=1}^{c_i} \left(\hat{g}_\theta^{(l+1)}\right)_m U^T f_n^{(l)}\right)$$

ChebNet avoids the expensive eigenvalue decomposition by approximating the convolution filter using its truncated polynomial expansion with K Chebyshev polynomials $T_k(x) = 2xT_{k-1}(x) - T_{i-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$ [69].

$$g * f \approx \sum_{k=0}^{K} \theta_k T_k \left(\frac{2L}{\lambda_{max}} - I_{|V|} \right) f \tag{3.1}$$

Coincidentally, ChebNet can extract K-hop localized features independent of the graph size since the filters defined by Chebyshev polynomials are spatially localized. Graph Convolution Network (GCN) introduces a first-order approximation of ChebNet with $K = 1$ and $\lambda_{max} = 2$ and considerably reduces the number of parameters simplifying (3.1) to

$$g * f \approx \theta \left(I_{|V|} + D^{-1/2} A D^{-1/2} \right) f$$

and proposes a renormalization trick to solve the exploding/vanishing gradient problem and stabilize training. Several other works employ spectral parameterization of convolutional filters. [70] extends ChebNet with complex Cayley polynomials. [71] uses wavelet transformations for fast computation of sparse localized information. In almost all of the spectral approaches, filters need to be retrained for graphs with different structures.

Spatial convolution

We can define convolutional filters in the spatial domain based on the topological properties of the entities. [35] propose Neural FP with different filters for nodes with different degrees $|\mathcal{N}_v|$ where $\mathcal{N}_v = \{u \in V | (u, v) \in E \text{ or } (v, u) \in E\}$ is the of node v.

$$f_v^{(l+1)} = \sigma \left(\left(g_\theta^{(l+1)} \right)_{|\mathcal{N}_v|} \left(f_v^{(l)} + \sum_{u \in \mathcal{N}_v} f_u^{(l)} \right) \right)$$

Diffusion Convolutional Neural Network (DCNN) uses the power series of degree normalized graph adjacency matrix to capture the effect of features in K-hop neighborhood [72]. PATCHY-SAN model extracts k neighbors for each node to propagate their information [73]. Learnable Graph Convolution Network (LGCN) uses max pooling on neighborhood features to spread the top-k features in each iteration [74]. GraphSAGE performs aggregation on fixed sized sampled features from the local neighborhood of nodes [75].

Self-attention mechanism

Attention based propagation assigns different weights for neighbors in the convolution operator. The Graph Attention Network (GAT) uses multi-head self-attention mechanism based on similarity of the node states at every given iteration [76].

$$\bar{f}(l+1)_v = \sigma \left(\sum_{u \in \mathcal{N}_v \cup v} \alpha_{uv}^{(l+1)} g_\theta^{(l+1)} f_u^{(l)} \right)$$

$$\alpha_{uv}^{(l)} = \text{softmax} \left(\text{NN}((g_\theta^{(l)} f_u^{(l)}; g_\theta^{(l)} f_v^{(l)})) \right)$$

Resulting state $\bar{f}_v^{(l+1)}$ is averaged or concatenated over different attention heads.

Unifying frameworks

Multiple generalization frameworks are proposed to unify the ConvGNN models. Mixture mOdel Network (MoNet) in [77] generalizes ConvGNNs by defining pseudo-coordinated on neighbor pairs and assigning model weights to these coordinates. Message Passing Neural Network (MPNN) defined messaging function $M_\theta^{(l)}$ and update function $U_\theta^{(l)}$ to propagate the states at iteration l and evaluates graph embedding using pooling operation R [78].

$$f_v^{(l+1)} = U_\theta^{(l+1)}(f_v^{(l)}, m_v^{(l+1)})$$
$$m_v^{(l+1)} = \sum_{u \in \mathcal{N}_v} M_\theta^{(l+1)}(f_v^{(l)}, f_u^{(l)}, w_{uv})$$

where w_{uv} is the weight assigned to undirected edge e_{uv}.
Non-Local Neural Network (NLNN) generalizes the local state update and self-attention mechanism with non-local separable operation

$$f_v^{(l+1)} = \frac{1}{Z^{(l)}} \sum_{u \in V} s_\theta(f_v^{(l)}, f_u^{(l)}) g_\theta(h_u^{(l)})$$

where $s_\theta(f_v^{(l)}, f_u^{(l)})$ is a scalar similarity measure between states of u and v and $g_\theta(f_u^{(l)})$ is the transformation of state of u [76, 79].
Graph Network (GN) is the generalizes majority of ConvGNNs including MPNN, NLNN, Interaction Networks, CommNet, GGNN and so on by applying Graph block operations with messaging, update, and state transformation steps.

3.3.2. Recurrent propagation modules

In recurrent GNNs state of the graph entities are updated by iterative application of the same parametric function. With slight abuse of notation we denote this function with g_θ. Also the observed information about the states t_v is gated using a parametric function o_θ. Note that here the state update functions are independent from iteration.
The model proposed in [80] uses the propagation operator

$$f_v^{(l+1)} = g_\theta(x_v, x_{E_v}, f_{\mathcal{N}_v}^{(l)}, x_{\mathcal{N}_v})$$
$$t_v^{(l)} = o_\theta(f_v^{(l)}, x_v)$$

where x_v, x_{E_v} and $x_{\mathcal{N}_v}$ are three feature vectors associated with node v, its edges and 1-hop neighbors. The matrix formulation of this update procedure results in a fixed point equation system at convergence (i.e. when updated states $(f_v^{(l+1)})_{v \in V} = (f_v^{(l)})_{v \in V}$) and by Banach's fixed point theorem has a unique solution $(f_v^{(\infty)})_{v \in V} = (\lim_{l \to \infty} f_v^{(l)} h)_{v \in V}$ for contrastive mapping g_θ which can be evaluated with the fixed point algorithm in [81]. Graph Echo State Network (GraphESN) is a special case of this model where fixed point solution is evaluated using reservoir dynamics [82]. Iterative fixed point computations can be avoided by formulating a constrained optimization problem as in Lagrange Propagation GNN (LP-GNN) [83].
Another extension to recurrent GNNs is to gate the information flow in each propagation step by employing well-known gating mechanisms. Gated Graph Neural Network (GGNN) uses GRU propagation units to aggregate messages from the neighbors [84] and GraphLSTM model uses LSTM units on specific graph structures without the contractive update operator limitation [85].

3.3.3. Skip connections

Skip connections are used in combination to propagation modules to prevent the over-smoothing and overfitting of the updated features in deep networks. As the GNN grows deeper most of the local information of the entities are washed out by the aggregated neighbor features, resulting in similar activations in the model output. Skip or gated connections ensure that local information about each entity sufficiently changes the output of the model. Highway GCN uses gating weights to apply the update function on state entity [86].

$$f^{(l+1)} = p^{(l+1)} \odot f_v^{(l)} + (1 - p^{(l+1)}) \odot g_\theta^{(l+1)}(f^{(l+1)})$$
$$p^{(l+1)} = \sigma(W_\theta^{(l+1)} f^{(l)} + b_\theta^{(l)})$$

Jump Knowledge Network (JKN) uses pooling and self-attention mechanisms to capture information from all the previous iteration states [87] and DeepGCN uses residual connections and dense connections so establish gradient flow in deep GNNs [88].

3.3.4. Sampling modules

Sampling modules are used to improve the generalization of the models, prevent overfitting and allow use of large scale graphs when the memory is not large enought to process them at once. In these cases, sampling module acts similar to Dropout connections, however operating on the inputs instead of the parameters.
Sampling can be performed in edge level, node level, of subgraph level. GraphSAGE performs edge sampling to fix the number of processed neighbors for each node [75]. While sampling may increase the generalization and allow training models over large structures, it might be necessary to employ variance reduction methods for more stable training.
In layer sampling we consider only a subset of V for each iteration. FastGCN [89] uses layer sampling with importance weights to propagate the information in the new graph structure. ClusterGCN [90] and GraphSAINT [91] implement examples of subgraph sampling modules where, sample distribution is determined by clustering and entity-dependent weights respectively.

3.3.5. Pooling modules

While propagation modules and sampling modules operate on a local level, pooling modules provide global information about the topology of subgraphs and graphs. One approach to combine entity information is to perform node-wise operations like max or weighted mean operations [92, 93]. Other approaches use clustering [69, 77] and graph downsampling [94, 95] in graph's spectral domain to extract the global features.

3.4. Training settings

Supervised settings involve training a model with pairs of graphs and target value and evaluating its performance on unobserved samples from the same distribution. Typical tasks in a supervised learning environment include classification or regression where the goal is to predict a discrete or real valued label for each unlabeled entity in a given graph and can be

applied on graph nodes or edges or entire graph samples. Common losses used in this case are categorical cross-entropy for discrete values and distance metrics for real values.
In unsupervised settings, the main objective is to find patterns in the graph distribution. For instance, clustering of nodes, edges, subgraphs or graphs or matching graphs, prediction of existence of edges or subgraphs with certain topology are tasks that we consider in unsupervised learning. Another approach in unsupervised learning is to model the generative process of graphs. Graph Auto-Encoders (GAE) assumes that each graph's adjacency matrix is generated from a low dimensional real latent variable with a simple DNN [60]. The posterior on the latent variable is approximated with GCN and the model is trained with variational inference where graph reconstruction quality is evaluated using matrix similarity metrics in euclidean space. In [96] an adversarial training method is used for a similar generation process. Alternative methods propose reconstruction of graph Laplacian [97] or pairwise node similarity [98]. Contrastive learning is another successful unsupervised method where the objective is to maximize the mutual information between graph representation and node features in [99] or more complex substructures.
In semi-supervised settings, labels are only available for a small partition of data samples and we intend to improve the prediction of the model using the structural information that we can get from large amount of unlabeled samples. The optimization objective in this settings is based on label prediction accuracy for labeled samples and reconstruction quality for unlabeled ones. [100] and [101] are two studies tackling this setting for graph domain.

3.5. Wireless applications

Many network characteristics and applications can inherently be represented in form of graph signals. Although, inference in large scale wireless networks with GNNs has not received any attention in the past years, optimization of various resource allocation problems within the network has benefited from this technology. In particular, one of the main attractions of GNNs in wireless networks is to find an approximate (locally optimized) resource allocator with amortized cost when finding the exact optima is otherwise intractable.
[102] proposes to utilize GNNs to optimize the allocated power in a wireless system. A K-user single access point environment with interference is modeled with a complete graph with channel information denoted as its edge and node features. Optimal power control scheme is approximated using the universality of GNNs and is learned by maximizing the weighted average of users' Signal to Inference + Noise Ratio (SINR) leveraging the permutation invariance and robustness against wrong Channel State Information (CSI) provided by the smooth GNN measures.
In [55], optimal power allocation problem with a set of transmitter and receiver pairs is solved using GNNs. Communication links with fading channels are modeled with stochastic edges in the graph and weighted rates are allocated to each pair by optimizing the dual problem of power allocation exploiting the scalability of GNNs. Radio resource allocation in large scale wireless network using MPNNs is also considered in [103] where it is proven to be effective in optimal power allocation and beamforming tasks.
Authors of [50] study the optimal link scheduling problem in device-to-device networks with imperfect CSI measurements. In their graph model, each connected pair is denoted with a graph node, while the interference channels are the edges. The graph embedding is

extracted based on only the device pairwise distances and the GNN model is trained along with a classifier that selects the optimal link at every given scenario. The model parameters are trained in supervised and unsupervised manner.
In [49], the authors propose RouteNet, an efficient model base on GNNs to estimate the end-to-end connection performance metrics such as delay and jitter given the underlying network topology, routing policy and instantaneous traffic. This model is used in Software Defined Networks (SDN) to manage the routing devices more efficiently in comparison to heuristic based methods and generalize the routing policies to unseen topologies. [52] extends the GNN architecture in RouteNet to cover more complex network characteristics induced by forwarding devices in the network and their queuing policies and service priorities.
Traffic optimization in datacenters is studies in the context of GNN models in [52]. Datacenter services and requests are modeled in terms of a graph and the optimal Flow Completion Time (FCT) scheme incorporating the centeralized information of the datacenter is parameterized with a GNN. Experiments show the effectiveness of GNNs in flow routing, scheduling and topology management in the datacenter in comparison to traditional heuristics.
[54] uses the decentralized optimization capability of GNNs to enhance the robustness of the network control and management policies in case of impairments in information exchange among neighbor base stations. A new retransmission mechanism is proposed to conform the message passing and propagation modules used in GNNs in wireless communication systems.

3.6. Discussion and Conclusion

Graph Neural Networks are efficient and scalable tools that allow modeling structured data present in many applications. Their success in embedding the topological information in graphs promoted their extensive use in wireless communication systems as resource allocation schemes and network performance predictors. While GNNs' utilization in wireless systems is spreading, it is important to take their challenges into consideration. GNNs as a family of parametric models are vulnerable to adversarial attacks. They are commonly deployed as black-box ML tools without an straightforward interpretability of their outputs. And, most importantly, training and optimization of GNNs requires large datasets and so far has been limited to simulation based datasets in wireless communication scenarios. Providing a short overview of the existing literature and methods to model and process graph signals we refer the curious readers to [104, 105] for more comprehensive reviews of GNNs and their applications.

4. Case Studies for Optimization in mMIMO

4.1. Introduction

Multi–antenna radio access technologies are widely employed as means to enhance wireless communication spectral efficiency and connectivity. Space-division multiple access (SDMA) has traditionally been used to enhance spectral efficiency in the uplink. In the downlink, dirty–paper coding (DPC) achieves the channel capacity region by encoding the data at the transmitter side in order to cancel interference at receiver side. Since, in practice, DPC is difficult to implement, there has been intensive research on suboptimal solutions which combine superposition coding (SC) and spatial processing. For instance, non-orthogonal multiple access (NOMA) [106] has recently become a key mechanism to significantly enhance communication rates by allowing multiple users to superimpose their signals in the time domain. The resulting interference is then processed at the receiver side using successive interference cancellation (SIC). Similarly, joint spatial division multiplexing (JSDM) [107] and rate splitting [108] use precoding to separate transmissions into clusters of users, and then apply the corresponding downlink processing to the resulting multiple–input single–output (MISO) channels. In that sense, an important issue to be solved is to decide which users should share the available resources to maximize the total performance. By partitioning different receivers into clusters, the system can take advantage of the spatial relationship among different signals and reduce multi-user interference, especially when the number of users is larger than the number of transmit antennas.

In the case where the number of transmit antennas is larger than the number of users to be served, the authors of [107] propose a two–stage precoding scheme (JSDM) such that interference among different groups of users is canceled out. In the first stage, a precoding matrix is designed according to the second-order statistics of each group. In the second stage, the instantaneous channels of each user is considered to minimize interference within each group. Later on, authors in [109] study two users clustering mechanisms. In both cases, the main assumption is that users spanning a similar subspace should be clustered together. Indeed, for a sufficient number of antennas, such assumption allows one to construct orthogonal precoding matrices based on the eigenvectors of the group channel matrix. In this sense, the authors mainly try to associate users to a cluster center, either estimating it from the data (k-means) or pre–fixing (fixed quantization) the groups' subspaces.

In the same line of work, [110] trains a neural network to solve the multi–dimensional user to cell assignment problem considering the users' geographical location. Despite providing large complexity gains when compared with traditional approaches, the method lacks on scalability due to the fixed input–output dimensions. The impacts and advantages of user pairing in NOMA downlink is investigated in [111]. As main results, the authors provide an analytical study of how to pair users either according to fixed power conditions or user fairness. As one would expect, in both cases, the channel quality of both users play a crucial role.

More recently, rate splitting (RS) [108, 112] has been recognized as possible leading mechanism to enhance communication in scenarios where we have imperfect channel state information (CSI). The idea consists on splitting the messages intended for users into a private $W^{(p)}$ and common part $W^{(c)}$. The common parts are encoded together into a common

stream $\tilde{s}^{(c)}$ while the private parts $s^{(p)}$ are encoded separately. Users decode the intended stream in a SIC fashion starting by $\tilde{s}^{(c)}$, the common part associated to all the users. In the generalized RS, $W^{(c)}$ might consist of several layers, which could lead to complex implementations. With that in mind, [108] also proposes a 2–layer hierarchical RS (HRS) mechanism. In this case, users are divided into G groups, each containing K_g users, resulting in the transmitted signal described by $\mathbf{x} = \tilde{\mathbf{p}}^{(c)}\tilde{s}^{(c)} + \sum_g^G \mathbf{p}_g^{(c)} s_g^{(c)} + \sum_g^G \sum_k^{K_g} \mathbf{p}_k^{(p)} s_k^{(p)}$. Notice that this is in contrast with conventional multi–user linear precoding (MU-LP) [113], which considers the residual multi–user interference as noise, and with NOMA, which would require users to completely decode other users' message. In order to compare these different approaches (NOMA, MU–LP and RS), the authors in [112] analyse their multiplexing gains in the downlink, considering both perfect and imperfect CSI. Results show that rate splitting performs equally or better than other classical approaches.

All the techniques reviewed above somehow benefit from clustering users into groups. Motivated by the need of an analytical study of clustering mechanisms for distinct wireless communication systems, we analyse two clustering algorithms. When there exist sufficient degrees-of-freedom, it is reasonable that groups are formed by users which are close in space, i.e., their AoA/AoD are close enough. However, since we are dealing with wireless channels where multipath is present, it is reasonable to measure users proximity based on how well aligned the subspaces spanned by their channel matrices are. Differently from previous works [107, 109, 110] we are primarily concerned with case where the number of antennas at the base station is smaller than the total number of receivers. Consequently, zero–forcing (ZF) and other linear precoder mechanisms are not possible. Additionally, the clustering algorithms discussed below allow for the comparison of subspaces of different dimensions. This is relevant, for instance, when we need to cluster users that have number of antennas.

We devote the following two sections to providing an account of the results that we have obtained in the context of the Windmill project regarding the use of clustering methods for the optimization of wireless networks. More specifically, in the next Section, we first consider the use of spectral clustering as means to classify channels into clusters according to how well aligned they are. In particular, we revisit kernel-based spectral clustering techniques based on dimensionality reduction according to the eigenstructure of kernel proximity matrices. Our contribution here has been on the study of the asymptotic spectrum of these matrix when the observations are complex valued. In Section 4.3, we describe an agglomerative hierarchical clustering of complex subspaces in which users are clustered according to the subspaced spanned by their channel matrices. Our contribution here has been on the derivation of a channel similarity measure, based on their subspace alignment, that allows to compare clusters of channels with different number of elements.

4.2. Spectral Clustering of Complex Valued Data Using Kernel Methods

Modern data science techniques use kernel methods as means to represent the data via high dimensional nonlinear mappings. The introduction of kernel methods has spurred the development of a number of non-linear extensions to classical machine learning linear algorithms, which have been usually referred to as kernel methods [114]. For instance, kernel methods enable us to perform clustering analysis in a higher dimensional feature space

without actually having to do the expensive operation of projecting the data onto this space. In the field of wireless communication, [115] shows that it is possible to tune spectral clustering techniques and do educated guesses on the choice of the kernel functions. To validate their results, the authors tackle the problem of pilot contamination in MIMO wireless networks by means of user clustering. One of the major issues in broadly applying machine learning techniques to telecommunications is the fact conventional ML methods traditionally deal with real–valued signals while telecommunication signals are fundamentally complex valued. In this sense, further investigation of kernel methods in large settings with complex observations is of high relevance and has a direct applications into wireless networks.

The spectral behavior of different kernels matrices (i.e. the behavior of its eigenvalues and eigenvectors) is of fundamental importance in order to determine the performance of the corresponding learning algorithms. To the best of our knowledge, all of the available studies focusing on the asymptotic behavior of kernel matrices are based on real valued observations. However, as we showed in [1], when the observations are drawn from a multivariate circularly symmetric complex distribution, the empirical eigenvalue distribution of a kernel matrix $\mathbf{K}$ converges almost surely to a limit that is different from the one obtained with real valued inputs [116,117]. In other words, it is not possible to directly apply the results obtained for real valued observations into the complex observations obtained in wireless systems.

To exemplify this idea, let's consider the case where the kernel matrix $\mathbf{K}$ of size $n \times n$ has entries

$$K_{ij} = \frac{1}{\sqrt{p}} k\left(\frac{\mathbf{x}_i^H \mathbf{x}_j}{\sqrt{p}}\right) \delta_{i \neq j}$$

where $\mathbf{x}_i$ is an independent circularly symmetric standard complex p–dimensional Gaussian random vector[1]. In spectral clustering, the largest eigenvectors of matrix $\mathbf{K}$ are often used as input to some vector quantization method (e.g., K-Means) in order to perform clustering. Moreover, $\mathbf{x}_i^H \mathbf{x}_j$ measures the alignment between observations $\mathbf{x}_i$, $\mathbf{x}_j$. Unfortunately, there exists no direct way of expressing this alignment in terms of the scalar product of the corresponding real and imaginary parts. Thus, complex kernel functions cannot be derived directly from its real–valued function counterparts.

In [1], we study the asymptotic eigenvalue distribution of the kernel matrix in the case where $\mathbf{x}$ are complex observations, with $p, n \to \infty$, $n/p \to \gamma$, $0 < \gamma < \infty$. We show that under some assumptions on the probability measures of the real and imaginary parts of $\mathbf{x}_i^H \mathbf{x}_j / \sqrt{p}$, with probability one, the empirical eigenvalue distribution of $\mathbf{K}$ converges weakly to a probability measure ξ, uniquely determined by its Stieltjes transform $m(z) = \int_{\mathbb{R}} \frac{d\xi}{t-z}$ for $z \in \mathbb{C}^+$ as the unique solution in $\mathbb{C}^+$ (the upper complex semi–plane) to the following quartic equation

$$\frac{-1}{m(z)} = z + \gamma m(z)\omega + \left(\frac{|\tilde{\alpha}|^2}{1 + \tilde{\alpha}\gamma m(z)} + \frac{|\alpha|^2}{1 + \alpha\gamma m(z)}\right)\gamma m(z) \tag{4.1}$$

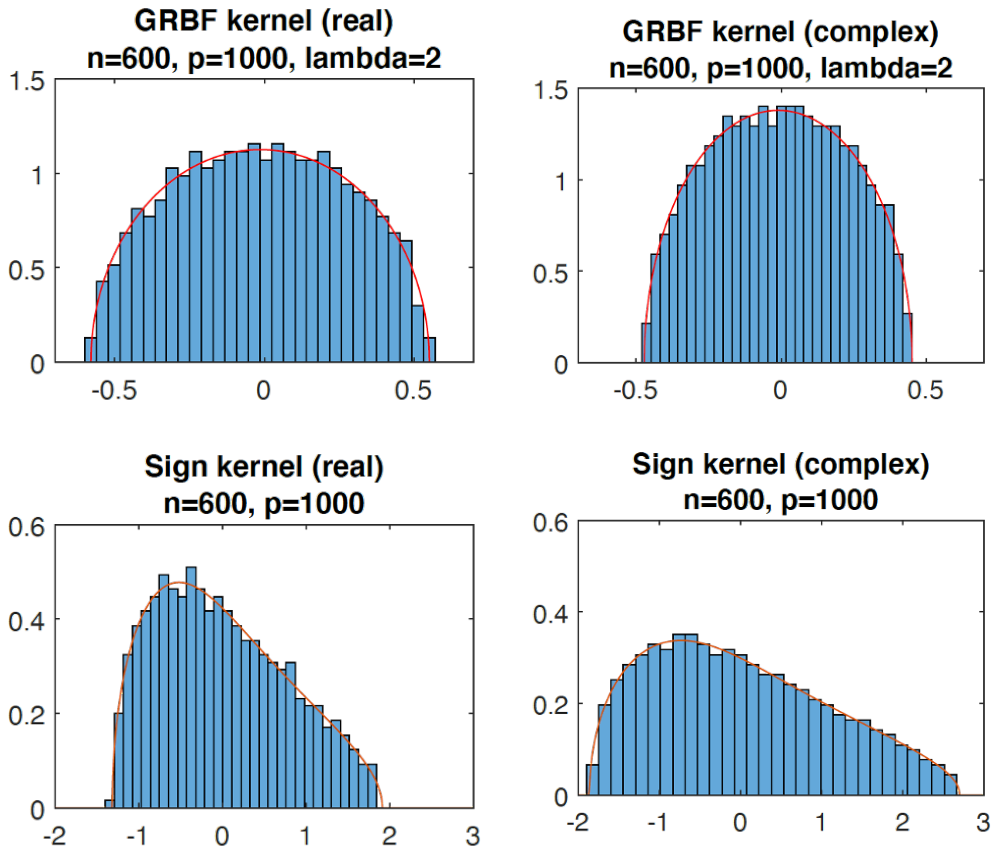
where

$$\omega = \sum_{q,r=0}^{\infty} |a_{q,r}|^2 - |a_{1,0}|^2 - |a_{0,1}|^2$$

and where we have introduced the two complex coefficients

$$\alpha = \frac{1}{\sqrt{2}}(a_{1,0} - \mathrm{i}a_{0,1}) \text{ and } \tilde{\alpha} = \frac{1}{\sqrt{2}}(a_{0,1} - \mathrm{i}a_{1,0}).$$

[1] Admittedly, this is an overly simplistic model, but provides a first step that can be used to study the behavior under more elaborate statistical models.



(a) (b)

Figure 4.1: Comparison between eigenvalue histograms and asymptotic eigenvalue distribution for different kernel functions built of real (a) and complex (b) observations. Extracted from [1].

Moreover, the coefficients $a_{k,l}$ are related to the complex expansion of the kernel function $k(z)$ with respect to the system of complex Hermite polynomials (see [116, Lemma 4.1]). A detailed explanation of this proof is can be found in [1].
In contrast, for the case where $\mathbf{K}$ is built of real valued observations, its Stieltjes transform $\tilde{m}(z)$ is uniquely defined by the cubic equation

$$\frac{-1}{\tilde{m}(z)} = z + \gamma \frac{a_1^2 \tilde{m}(z)}{1 + a_1 \gamma \tilde{m}(z)} + \gamma \tilde{m}(z) \sum_{k \geq 1} a_k^2. \tag{4.2}$$

Here the variables a_k are again described according to [116]. In that sense, the solution to (4.2) can be understood as a particularization of (4.1) – $\mathbf{K}$ built of complex observations. Furthermore, the particularisation of both solutions to different kernel functions shows the fact that the eigenvalue density behaves essentially differently in the real and complex valued cases. Figure 4.1 illustrates the histogram of the eigenvalues obtained for $n = 600$ and $p = 1000$ against the asymptotic eigenvalue density as established in (4.1) and (4.2) – solid red lines. Observe that, despite the distinct behavior of the real and complex cases, our method provides (in both cases) a very good approximation between the asymptotic eigenvalue distribution and the empirical one. Notice that here we are only considering

the case where the observations come from the same distribution, i.e., they all belong to the same class. As we further explain in [1], the study of the support of the eigenvalue distribution will be crucial to extend the current result to the case where the observations follow a multi-class Gaussian mixture model.
The results described in this section provide a primary characterization of the analytical behavior of complex kernel matrices. As previously discussed, this analytical study provides the foundations for future studies of a number of kernel–based learning methods [115–117]. In that sense, the results in [1] provide us with initial tools to analyse and perform clustering of network users based on their wireless channels.

4.3. Hierarchical Clustering in mMIMO Systems

In previous section we have revised spectral clustering of single antenna users using kernel methods. In this section we explore a more generic approach which allows us to perform clustering of multiple–input multiple–output (MIMO) channels. Consider the scenario where we have K user receivers, each equipped N_k antennas, and a base station (BS) transmitter equipped with M antennas. We assume a separable Rayleigh model for the MIMO fading channel (also referred to as Kronecker model), according to which the channel matrices are independent among users and can be decomposed as

$$\mathbf{H}_k = \mathbf{R}_k^{\frac{1}{2}} \mathbf{G}_k \mathbf{T}_k^{\frac{1}{2}} \tag{4.3}$$

where $\mathbf{R}_k \in \mathbb{C}^{M \times M}$ is the channel spatial covariance matrix at the BS and $\mathbf{T}_k \in \mathbb{C}^{N_k \times N_k}$ is the channel spatial covariance matrix at the kth UE. These correlations are inherently dependent on the scattering structure of the scenario, and in particular on the angles–of–arrival (AoA) and angles–of–departure (AoD) of the multiple transmission paths. Moreover, the entries of the $M \times N_k$ matrix $\mathbf{G}_k$ are assumed to be independent and identically distributed (i.i.d) complex circularly symmetric Gaussian random variables, with zero mean and unit variance. As mentioned before, in the line–of–sight case, it is possible to cluster users based only on their AoA but when reflections and obstructions take place, this is no longer trivial. With that in mind, we are interested in analysing the alignment of the subspace spanned by the users' channel matrices. The Grassmann manifold is a common tool which provides a topological structure to a set of subspaces [118, 119].
Let $\mathcal{H}_j, \mathcal{H}_k$ denote the subspaces spanned by the columns of $\mathbf{H}_j, \mathbf{H}_k$ respectively. Then, for $N = min(N_j, N_k)$, the principal angles between $\mathcal{H}_j, \mathcal{H}_k$ induce several distance metrics on the (complex) Grassmann manifold $\mathcal{G}(N, M)$, which provides a topological structure to the set of all N–dimensional subspaces in a (complex) M–dimensional space [118–120]. The Grassmannian $\mathcal{G}(N, M)$ can be seen as the manifold built of the symmetric projection matrices of size $M \times M$ and rank N [120]. In other words, we can represent the column space of a full rank channel $\mathbf{H}_k$ as a point $\hat{\mathbf{P}}_k \in \mathcal{G}(N_k, M)$, where $\hat{\mathbf{P}}_k$ is the projection matrix

$$\hat{\mathbf{P}}_k = \mathbf{H}_k (\mathbf{H}_k^{\mathrm{H}} \mathbf{H}_k)^{-1} \mathbf{H}_k^{\mathrm{H}}. \tag{4.4}$$

Particularly, we are interested in the squared projection-Frobenius distance, which is defined as

$$d_{\mathrm{PF}}^2(\mathbf{H}_k, \mathbf{H}_l) = \sum_{i=1}^{N} \sin^2(\alpha_{k,l}(i)) = \ = N - \sum_{i=1}^{N} \lambda_i^{(\hat{k},l)} = N - \mathrm{tr}(\hat{\mathbf{P}}_k \hat{\mathbf{P}}_l) \tag{4.5}$$

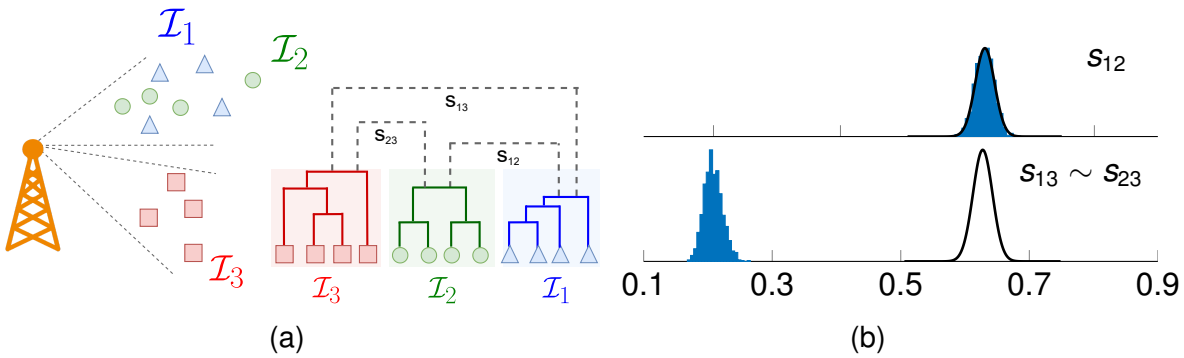


Figure 4.2: Merging point in agglomerative hierarchical clustering where groups have equal number of antennas, $N_{\mathcal{I}_1} = N_{\mathcal{I}_2} = N_{\mathcal{I}_3} = 4$. (a) Groups spread in the spatial domain and their respective dendrogram connectivity. (b) behavior of the similarity measures for different channel realisations of these groups.

where tr(·) denotes the trace of a matrix and $\alpha_{k,l}(i)$ the ith principal angle between the subspaces $\mathcal{H}_i, \mathcal{H}_j$. We refer the reader to [121] for a detail review in other metrics performed in Grassmann manifolds. The main advantage of the distance in (4.5) with respect to other metrics in this manifold is the fact that it can be computed without the need for eigendecompositions.

In summary, the above framework allows us to compare the column subspace spanned by two distinct wireless channels $\mathbf{H}_k, \mathbf{H}_j \in \mathbb{C}^{M\times N}$ according to the distance between their projection matrices $\hat{\mathbf{P}}_k, \hat{\mathbf{P}}_j$ as elements of $\mathcal{G}(N, M)$. For simplicity, from now on we will instead consider the similarity measure that follows from (4.5), namely

$$s_{k,j} = \frac{1}{M}\text{tr}(\hat{\mathbf{P}}_k\hat{\mathbf{P}}_j). \tag{4.6}$$

Note that this measure is sufficient to describe the alignment between the two different subspaces spanned by $\hat{\mathbf{P}}_i$ and $\hat{\mathbf{P}}_j$.

We now have introduced the main concepts necessary to formulate an agglomerative hierarchical clustering. In this bottom up approach, the objective is to combine clusters according to some similarity measure. This merging process continues until all clusters are merged together or until clusters are no longer similar. The output of the algorithm is usually visualized as a dendrogram. Often, an important design question is how to define if clusters are similar or not.

In the context of wireless communication, a group is represented by the different users in the network and the similarity measure between two clusters is defined as in (4.6). Initially, each group is formed by a single user. Moreover, merging different users results in horizontally concatenating their channel matrices. Let us exemplify this merging process in some specific scenario. Figure 4.2a illustrates a merging point in the hierarchical algorithm where all the clusters have the same number of antennas, i.e., $N_{\mathcal{I}_1} = N_{\mathcal{I}_2} = N_{\mathcal{I}_3} = 4$. We are interested in merging either $[\mathcal{I}_1, \mathcal{I}_2]$, $[\mathcal{I}_1, \mathcal{I}_3]$ or $[\mathcal{I}_2, \mathcal{I}_3]$. Moreover, we assume that $\mathcal{I}_1$ and $\mathcal{I}_2$ have the same spatial covariance matrix at the receiver side and, therefore, belong to the same cluster. Also, $\mathcal{I}_3$ has a different spatial covariance matrix. In other words, $\mathbf{R}_{\mathcal{I}_1} = \mathbf{R}_{\mathcal{I}_2}$ and $\mathbf{R}_{\mathcal{I}_2} \neq \mathbf{R}_{\mathcal{I}_3}$.

In this scenario, the similarity measure should be higher between $[\mathcal{I}_1, \mathcal{I}_2]$, so that we should in principle have $s_{1,2} > s_{1,3}$ and $s_{1,2} > s_{2,3}$. Indeed, by comparing different realisations of

users' channels (blue histograms in Figure 4.2b) we notice that $s_{1,2}$ is always higher than all the other comparisons. In Figure 4.2b, observe that because $N_{\mathcal{I}_2} = N_{\mathcal{I}_1}$ and $\mathbf{R}_1 = \mathbf{R}_2$ also $s_{12} \sim s_{23}$, i.e., they have the same distribution. Moreover, the solid black line represents the similarity measure's expected behavior[2] whenever ith and jth groups are drawn from the same distribution, i.e., it represents the probability density function of the statistic $s_{i,j}$ under the null hypothesis $\mathrm{H}_{\text{null}}(i,j) : \mathbf{R}_i = \mathbf{R}_j$. We emphasize that if the histogram is on the left–hand side of the expected distribution, then there exists certain evidence that the two groups should not be merged, e.g, in the case of $s_{2,3}$ and $s_{1,3}$.

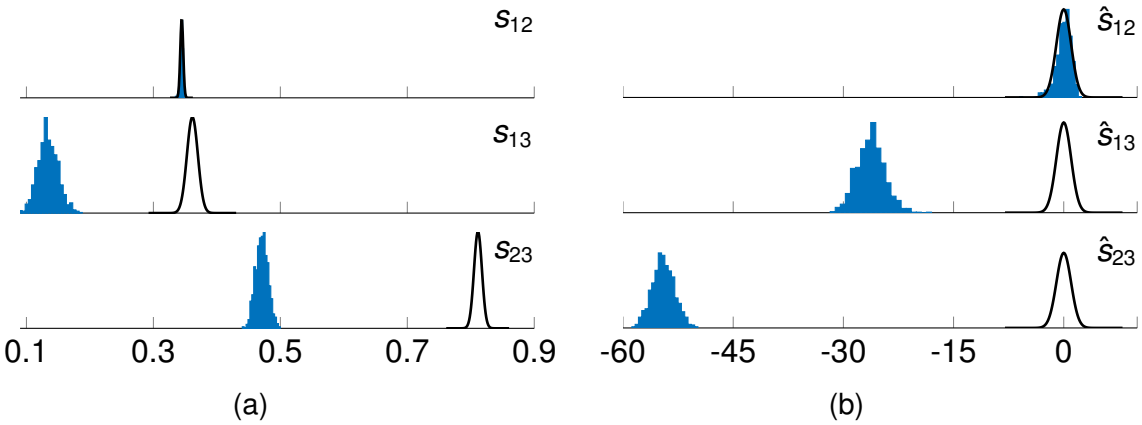


Figure 4.3: behavior of similarity measure for groups of different dimensions before (a) and after (b) normalization with respect to the null hypothesis.

Next, let's now consider the scenario where the total number of antennas are distinct among the different groups. In this case, we have that $\mathbf{P}_{\mathcal{I}_1} \in \mathcal{G}(N_{\mathcal{I}_1}, M)$, $\mathbf{P}_{\mathcal{I}_2} \in \mathcal{G}(N_{\mathcal{I}_2}, M)$, $\mathbf{P}_{\mathcal{I}_3} \in \mathcal{G}(N_{\mathcal{I}_3}, M)$ and

$$\mathcal{G}(N_{\mathcal{I}_1}, M) \neq \mathcal{G}(N_{\mathcal{I}_2}, M) \neq \mathcal{G}(N_{\mathcal{I}_3}, M). \tag{4.7}$$

Therefore, in this new scenario, the measures $s_{1,2}, s_{2,3}$ and $s_{1,3}$ are no longer comparable. Figure 4.3a represents the same results as in Figure 4.2b but for the case where $N_{\mathcal{I}_1} = 4, N_{\mathcal{I}_2} = 24$ and $N_{\mathcal{I}_3} = 32$. Based only on the similarity comparisons, one would wrongly conclude $(\mathcal{I}_1, \mathcal{I}_3)$ to be the merge of choice.
Notice that from the definition (4.5) that whenever $N_i \neq N_j$ then only the first $min(N_i, N_j)$ principal angles are considered. In this sense, comparisons between large clusters might lead to higher similarity than comparisons between a small and a large cluster. This might happen regardless of the true group assignment. In fact, in this new scenario, even tough $s_{1,2}$ follows the expected behavior $\mathrm{H}_{\text{null}}(1,2)$, the highest similarity happens between clusters $\mathcal{I}_2$ and $\mathcal{I}_3$.
There are a few existing proposals for estimating the distance between subspaces of different dimensions – containment gap [122], symmetric directional [123], distance and Schubert varieties [124], to name a few. We argue that one could also take a more statistical approach and standardize the similarity $s_{i,j}$ measure with respect to its expected behavior $\mathrm{H}_{\text{null}}(i,j)$ which can be established in the asymptotic regime of large antenna systems. As a result, we obtain can obtain a re-normalised metric $\hat{s}_{ij}$ which follows a Gaussian distribution with

[2]We have studied the asymptotic distribution and derived a close form expression for this distribution, but do not provide details here to keep the exposition as simple as possible.

zero mean and unit standard deviation whenever $\mathrm{H}_{\mathrm{null}}(i, j)$ holds. Figure 4.3b exemplifies this case when comparing $\mathcal{I}_1$ and $\mathcal{I}_2$ in $\hat{s}_{12}$. In the cases where two clusters have different spatial covariance matrices – e.g., $(\mathbf{R}_{\mathcal{I}_1}, \mathbf{R}_{\mathcal{I}_3})$ and $(\mathbf{R}_{\mathcal{I}_2}, \mathbf{R}_{\mathcal{I}_3})$ – their normalised similarities – $\hat{s}_{13}$ and $\hat{s}_{23}$, respectively – are moved far away from the standard normal distribution. As a conclusion, notice that in this normalised metric space, groups that have the same covariance matrix at the receiver side will always have higher similarity than groups with different covariance matrices. Therefore allowing the comparison of clusters with distinct number of users.

5. References

[1] M. Xavier, P. Roberto, and G. David, “Asymptotic Spectral Behavior of Kernel Matrix in Complex Valued Observations,” in *Accepted for publication on IEEE Data Science & Learning Workshop (DSLW)*, IEEE, 2021.

[2] C. P. Callender and C. F. N. Cowan, “Two novel non-linear approaches to channel equalisation for digital communications,” in *Proc. of the 2 nd Cost # 229 on Adaptive Algorithms in Communications*, pp. 247–254, Unk, 1992.

[3] G. Kechriotis, E. Zervas, and E. Manolakos, “Using recurrent neural networks for adaptive communication channel equalization,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 267–278, 1994.

[4] P. R. Chang and B. F. Yeh, “Non-linear communication channel equalization using wavelet neural networks,” in *Proc. of the IEEE Int. Conf on Neural Networks*, pp. 3605–3610, IEEE, 1994.

[5] S. Chen, G. Gibson, C. Cowan, and P. Grant, “Reconstruction of binary signals using an adaptive radial basis function equalizer,” *Signal Processing*, vol. 22, no. 2, pp. 77–93, 1991.

[6] B. Aazang, B. Paris, and G. Orsak, “Neural networks for multiuser detection in code-division multiple access communications,” *IIEEE Transactions on Communications*, vol. 40, no. 7, pp. 1212–1222, 1992.

[7] D. J. C. MacKay, *Information theory, inference, and learning algorithms*. Cambridge University Press, 2004.

[8] R. G. Gallager, *Low density parity check codes*. MIT Thesis, 1990.

[9] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[10] S. Singh and D. Bertsekas, “Reinforcement learning for dynamic channel allocation in cellular telephone systems,” 1997.

[11] S. Liu, X. Hu, and W. Wang, “Deep reinforcement learning based dynamic channel allocation algorithm in multibeam satellite systems,” *IEEE Access*, vol. 6, pp. 15733–15742, 2018.

[12] G. Tesauro, “Temporal difference learning and td-gammon,” *Commun. ACM*, vol. 38, p. 58âĂŞ68, Mar. 1995.

[13] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[14] R. Crites and A. Barto, "Improving elevator performance using reinforcement learning," in *Advances in Neural Information Processing Systems* (D. Touretzky, M. C. Mozer, and M. Hasselmo, eds.), vol. 8, MIT Press, 1996.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[16] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. USA: John Wiley & Sons, Inc., 1st ed., 1994.

[17] C. Szepesvari, *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[18] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012.

[19] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[20] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*. USA: Prentice-Hall, Inc., 1987.

[21] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951.

[22] J. H. Venter, "On Dvoretzky Stochastic Approximation Theorems," *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1534 – 1544, 1966.

[23] J. Kiefer and J. Wolfowitz, "Stochastic Estimation of the Maximum of a Regression Function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462 – 466, 1952.

[24] H. Kushner, "Stochastic approximation: a survey," *WIREs Computational Statistics*, vol. 2, no. 1, pp. 87–96, 2010.

[25] J. N. Tsitsiklis, "Asynchronous stochastic approximation and q-learning," *Mach. Learn.*, vol. 16, p. 185âĂŞ202, Sept. 1994.

[26] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," *Neural Computation*, vol. 6, no. 6, pp. 1185–1201, 1994.

[27] E. Even-Dar and Y. Mansour, "Learning rates for q-learning," *J. Mach. Learn. Res.*, vol. 5, p. 1âĂŞ25, Dec. 2004.

[28] Y. Ye, "The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate," *Mathematics of Operations Research*, vol. 36, no. 4, pp. 593–603, 2011.

[29] A. G. Barto, *Connectionist Learning for Control: An Overview*, p. 5âĂŞ58. Cambridge, MA, USA: MIT Press, 1990.

[30] G. Konidaris, S. Osentoski, and P. Thomas, "Value function approximation in reinforcement learning using the fourier basis," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, p. 380âĂŞ385, AAAI Press, 2011.

[31] B. Farley and W. Clark, "Simulation of self-organizing systems by digital computer," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 76–84, 1954.

[32] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[33] A. G. Barto, "Reinforcement learning: Connections, surprises, and challenge," *AI Magazine*, vol. 40, pp. 3–15, Mar. 2019.

[34] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*. Springer Publishing Company, Incorporated, 2014.

[35] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in Neural Information Processing Systems*, vol. 2015-January, pp. 2224–2232, 2015.

[36] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of computer-aided molecular design*, vol. 30, no. 8, pp. 595–608, 2016.

[37] K. Do, T. Tran, and S. Venkatesh, "Graph transformation policy network for chemical reaction prediction," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 750–760, 2019.

[38] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *International Conference on Machine Learning*, pp. 2688–2697, PMLR, 2018.

[39] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning*, pp. 4470–4479, PMLR, 2018.

[40] Y. Zhang, P. Qi, and C. D. Manning, "Graph convolution over pruned dependency trees improves relation extraction," *arXiv preprint arXiv:1809.10185*, 2018.

[41] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 7370–7377, 2019.

[42] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, "Large-scale hierarchical text classification with recursively regularized deep graph-cnn," in *Proceedings of the 2018 world wide web conference*, pp. 1063–1072, 2018.

[43] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto, "Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach," *arXiv preprint arXiv:1706.05674*, 2017.

[44] X. Wang, Y. Ye, and A. Gupta, "Zero-shot recognition via semantic embeddings and knowledge graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6857–6866, 2018.

[45] M. Kampffmeyer, Y. Chen, X. Liang, H. Wang, Y. Zhang, and E. P. Xing, "Rethinking knowledge graph propagation for zero-shot learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11487–11496, 2019.

[46] Y. Wu, D. Lian, Y. Xu, L. Wu, and E. Chen, "Graph convolutional networks with markov random field reasoning for social spammer detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 1054–1061, 2020.

[47] W. Lin, Z. Gao, and B. Li, "Guardian: Evaluating trust in online social networks with graph convolutional networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 914–923, IEEE, 2020.

[48] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, 2018.

[49] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," in *Proceedings of the 2019 ACM Symposium on SDN Research*, pp. 140–151, 2019.

[50] M. Lee, G. Yu, and G. Y. Li, "Graph embedding-based wireless link scheduling with few training samples," *IEEE Transactions on Wireless Communications*, vol. 20, no. 4, pp. 2282–2294, 2020.

[51] J. Li, P. Sun, and Y. Hu, "Traffic modeling and optimization in datacenters with graph neural network," *Computer Networks*, vol. 181, p. 107528, 2020.

[52] A. Badia-Sampera, J. Suárez-Varela, P. Almasan, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Towards more realistic network models based on graph neural networks," in *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*, pp. 14–16, 2019.

[53] Y. Kong, D. Petrov, V. Räisänen, and A. Ilin, "Path-link graph neural network for ip network performance prediction," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 170–177, IEEE, 2021.

[54] M. Lee, G. Yu, and H. Dai, "Decentralized inference with graph neural networks in wireless communication systems," *arXiv preprint arXiv:2104.09027*, 2021.

[55] M. Eisen and A. Ribeiro, "Large scale wireless power allocation with graph neural networks," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, IEEE, 2019.

[56] M. Eisen and A. Ribeiro, "Optimal wireless resource allocation with random edge graph neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2977–2991, 2020.

[57] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," *arXiv preprint arXiv:1612.00222*, 2016.

[58] S. Sukhbaatar, R. Fergus, *et al.*, "Learning multiagent communication with backpropagation," *Advances in neural information processing systems*, vol. 29, pp. 2244–2252, 2016.

[59] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

[60] T. N. Kipf and M. Welling, "Variational Graph Auto-Encoders," *arXiv*, 2016.

[61] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous Graph Attention Network," in *WWW '19: The World Wide Web Conference*, pp. 2022–2032, New York, NY, USA: Association for Computing Machinery, May 2019.

[62] X. Fu, J. Zhang, Z. Meng, and I. King, "MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding," *Proceedings of WWW*, 2020.

[63] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting," *ICLR*, 2018.

[64] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting," in *IJCAI'18: Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3634–3640, AAAI Press, Jul 2018.

[65] Y. Huang, H. Xu, Z. Duan, A. Ren, J. Feng, and X. Wang, "Modeling Complex Spatial Patterns with Temporal Features via Heterogenous Graph Embedding Networks," *arXiv preprint arXiv:2008.08617*, 2020.

[66] S. Yan, Y. Xiong, and D. Lin, "Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition," 2018.

[67] C. Zang and F. Wang, "Neural Dynamics on Complex Networks," in *KDD '20: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[68] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral Networks and Locally Connected Networks on Graphs," *ICLR*, Dec 2013.

[69] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," 2016.

[70] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph Convolutional Neural Networks With Complex Rational Spectral Filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, 2018.

[71] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph Wavelet Neural Network," *ICLR*, 2019.

[72] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 2001–2009, 2016.

[73] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning Convolutional Neural Networks for Graphs," in *International Conference on Machine Learning*, pp. 2014–2023, PMLR, 2016.

[74] H. Gao, Z. Wang, and S. Ji, "Large-Scale Learnable Graph Convolutional Networks," in *KDD '18: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1416–1424, Jul 2018.

[75] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

[76] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph Attention Networks," *ICLR*, 2018.

[77] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5425–5434, IEEE Computer Society, 2017.

[78] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for Quantum chemistry," *ICML'17: Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 2017.

[79] Y. Hoshen, "VAIN: Attentional Multi-agent Predictive Modeling," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[80] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[81] M. A. Khamsi and W. A. Kirk, *An Introduction to Metric Spaces and Fixed Point Theory*. Wiley, 2001.

[82] C. Gallicchio and A. Micheli, "Graph Echo State Networks," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2010.

[83] M. Tiezzi, G. Marra, S. Melacci, and M. Maggini, "Deep constraint-based propagation in graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[84] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated Graph Sequence Neural Networks," *ICLR*, 2015.

[85] N. Peng, H. Poon, C. Quirk, K. Toutanova, and W.-t. Yih, "Cross-Sentence N-ary Relation Extraction with Graph LSTMs," *Transactions of the Association for Computational Linguistics*, vol. 5, no. 0, 2017.

[86] A. Rahimi, T. Cohn, and T. Baldwin, "Semi-supervised User Geolocation via Graph Convolutional Networks," *ACL Anthology*, 2018.

[87] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 5449–5458, PMLR, 2018.

[88] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive Graph Convolutional Neural Networks," *AAAI*, vol. 32, Apr 2018.

[89] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling," *arXiv*, 2018.

[90] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks," in *KDD '19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Association for Computing Machinery, 2019.

[91] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph Sampling Based Inductive Learning Method," *ICLR*, 2020.

[92] O. Vinyals, S. Bengio, and M. Kudlur, "Order Matters: Sequence to sequence for sets," *arXiv*, 2015.

[93] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[94] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 29–38, IEEE Computer Society, 2017.

[95] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *International Conference on Machine Learning*, pp. 3734–3743, PMLR, 2019.

[96] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," in *IJCAI'18: Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 2609–2615, AAAI Press, 2018.

[97] J. Park, M. Lee, H. J. Chang, K. Lee, and J. Y. Choi, "Symmetric graph convolutional autoencoder for unsupervised graph representation learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6519–6528, 2019.

[98] G. Cui, J. Zhou, C. Yang, and Z. Liu, *Adaptive Graph Encoder for Attributed Graph Embedding*. Association for Computing Machinery.

[99] P. Velickovic, W. Fedus, W. L. Hamilton, P. Lio, Y. Bengio, and R. D. Hjelm, "Deep Graph Infomax," *ICLR*, 2019.

[100] H. Wang and J. Leskovec, "Unifying Graph Convolutional Neural Networks and Label Propagation," *arXiv*, 2020.

[101] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," *arXiv preprint arXiv:2004.11198*, 2020.

[102] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "A graph neural network approach for scalable wireless power control," in *2019 IEEE Globecom Workshops, Waikoloa, HI, USA, December 9-13, 2019*, pp. 1–6, IEEE, 2019.

[103] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "Graph Neural Networks for Scalable Radio Resource Management: Architecture Design and Theoretical Analysis," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 101–115, 2020.

[104] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[105] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[106] M. Saideh, Y. Alsaba, I. Dayoub, and M. Berbineau, "Joint Interference Cancellation for Multi-Carrier Modulation-Based Non-Orthogonal Multiple Access," *IEEE Communications Letters*, vol. 23, no. 11, pp. 2114–2117, 2019.

[107] A. Adhikary, J. Nam, J.-Y. Ahn, and G. Caire, "Joint spatial division and multiplexingâĂŤthe large-scale array regime," *IEEE transactions on information theory*, vol. 59, no. 10, pp. 6441–6463, 2013.

[108] Y. Mao, B. Clerckx, and V. O. Li, "Rate-Splitting Multiple Access for Downlink Communication Systems: Bridging, Generalizing, and Outperforming SDMA and NOMA," *EURASIP journal on wireless communications and networking*, vol. 2018, no. 1, pp. 1–54, 2018.

[109] J. Nam, A. Adhikary, J.-Y. Ahn, and G. Caire, "Joint Spatial Division and Multiplexing: Opportunistic Beamforming, User Grouping and Simplified Downlink Scheduling," *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 5, pp. 876–890, 2014.

[110] A. Zappone, L. Sanguinetti, and M. Debbah, "User Association and Load Balancing for Massive MIMO through Deep Learning," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pp. 1262–1266, IEEE, 2018.

[111] Z. Ding, P. Fan, and H. V. Poor, "Impact of User Pairing on 5G Nonorthogonal Multiple-Access Downlink Transmissions," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 8, pp. 6010–6023, 2016.

[112] B. Clerckx, Y. Mao, R. Schober, E. Jorswieck, D. J. Love, J. Yuan, L. Hanzo, G. Y. Li, E. G. Larsson, and G. Caire, "Is NOMA efficient in multi-antenna networks? A critical look at next generation multiple access techniques," *IEEE Open Journal of the Communications Society*, 2021.

[113] B. Clerckx and C. Oestges, *MIMO wireless networks: channels, techniques and standards for multi-antenna, multi-user and multi-cell systems*. Academic Press, 2013.

[114] B. Schölkopf, A. J. Smola, F. Bach, *et al.*, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT press, 2002.

[115] R. Couillet and A. Kammoun, "Random Matrix Improved Subspace Clustering," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, pp. 90–94, IEEE, 2016.

[116] X. Cheng and A. Singer, "The Spectrum of Random Inner-product Kernel Matrices," *Random Matrices: Theory and Applications*, vol. 2, no. 04, p. 1350010, 2013.

[117] R. Couillet and F. Benaych-Georges, "Kernel spectral clustering of large dimensional data," *Electronic journal of statistics*, vol. 10, no. 1, pp. 1393–1454, 2016.

[118] U. Helmke, K. Hüper, and J. Trumpf, "Newton's method on Grassmann manifolds," *arXiv preprint arXiv:0709.2205*, 2007.

[119] J. Boets, K. De Cock, M. Espinoza, and B. De Moor, "Clustering Time Series, Subspace Identification and Cepstral Distances," *Communications in Information & Systems*, vol. 5, no. 1, pp. 69–96, 2005.

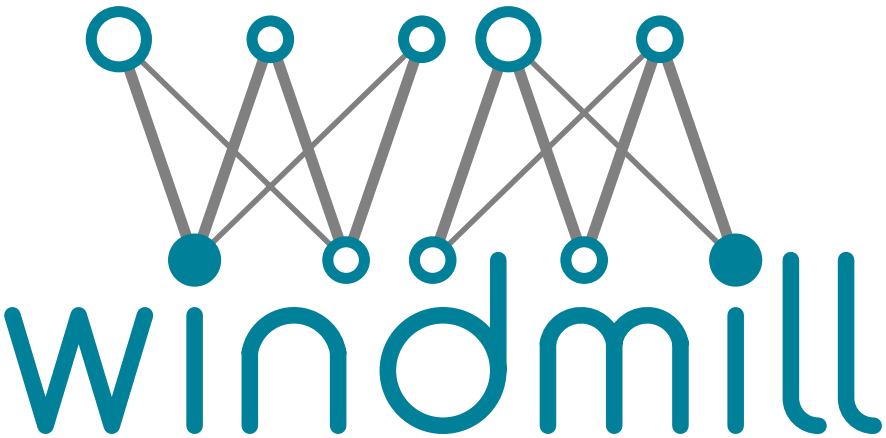
[120] L.-H. Lim, K. S.-W. Wong, and K. Ye, "The grassmannian of affine subspaces," *Foundations of Computational Mathematics*, vol. 21, pp. 537–574, 2021.

[121] A. Edelman, T. A. Arias, and S. T. Smith, "The Geometry of Algorithms with Orthogonality Constraints," *SIAM J. Matrix Anal. Appl.*, vol. 20, p. 303âĂŞ353, Apr. 1999.

[122] C. A. Beattie, M. Embree, and D. C. Sorensen, "Convergence of polynomial restart krylov methods for eigenvalue computations," *SIAM review*, vol. 47, no. 3, pp. 492–515, 2005.

[123] X. Sun, L. Wang, and J. Feng, "Further results on the subspace distance," *Pattern recognition*, vol. 40, no. 1, pp. 328–329, 2007.

[124] K. Ye and L.-H. Lim, "Schubert Varieties and Distances between Subspaces of Different Dimensions," *SIAM Journal on Matrix Analysis and Applications*, vol. 37, no. 3, pp. 1176–1197, 2016.



www.windmill-itn.eu