Deep Learning, Sensing-based IRSA (DS-IRSA): Learning a Sensing Protocol with Deep Reinforcement Learning

Iman Hmedoush, Cédric Adjih, Paul Mühlethaler

Deep Learning, Sensing-based IRSA (DS-IRSA): Learning a Sensing Protocol with Deep Reinforcement Learning

Iman Hmedoush Cédric Adjih , Paul Mühlethaler

RESEARCH REPORT
N° 9479
June 2022
Project-Teams AIO and TRiBE

ISSN 0249-6399 ISRN INRIA/RR--9479--FR+ENG

Deep Learning, Sensing-based IRSA (DS-IRSA): Learning a Sensing Protocol with Deep Reinforcement Learning

Iman Hmedoush *Cédric Adjih †, Paul Mühlethaler ‡

Project-Teams AIO and TRiBE

Research Report n° 9479 — June 2022 — 42 pages

* Inria Saclay
† Inria Saclay
‡ Inria Paris

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Abstract: Irregular Repetition Slotted Aloha (IRSA) is one candidate member of a family of random access-based protocols to solve massive connectivity problem for Internet of Things (IoT) networks. The key features of this protocol is to allow users to repeat their packets multiple times in the same frame and use Successive Interference Cancellation (SIC) to decode collided packets at the receiver. Although, the plain IRSA scheme can *asympotically* reach the optimal 1 [packet/slot]. But there are still many obstacles to achieve this performance, specially when considering short frame length. In this report, we study two new variants of IRSA with short frame length, and we optimize their performance using a Deep Reinforcement Learning approach. In our first variant, Random Codeword Selection-IRSA (RC-IRSA), we consider an IRSA approach with random codeword selection, where each codeword represents the transmission strategy of a user on the slots. We apply a Deep Reinforcement Learning to optimize RC-IRSA: we train a Deep Neural Network model that choses the slots on which the user sends its packets. Our DRL approach for RC-IRSA is a new optimization method for IRSA using a DRL approach and it works as a base for our second proposed IRSA variant DS-IRSA.
Our second variant is a sensing protocol based on IRSA and trained with machine learning to synchronize the nodes during the transmission and avoid collisions. For that aim, we proposed DS-IRSA, Deep Learning Sensing-based IRSA protocol which is composed of two phases: a sensing phase, where the nodes can sense the channel and send short jamming signals, followed by a classical IRSA transmission phase. We use a DRL algorithm to optimize its performance. Our proposed protocol has shown an excellent performance to achieve an optimal performance of almost 1 [*decoded user/slot*] for small frame sizes (≤ 5) slots and with enough sensing duration.

Key-words: Random Access, mMTC, IoT, Coded Slotted Aloha (CSA), Irregular Repetition Slotted Aloha (IRSA), Successive Interference Cancellation (SIC), Density Evolution (DE), Machine Learning (ML).

Une approche avec l'apprentissage par renforcement profond pour le protocole IRSA

Résumé : Irregular Repetition Slotted Aloha (IRSA) est un candidat d'une famille de protocoles d'accès aléatoire pour résoudre le problème de la connectivité massive pour les réseaux de l'internet des objets (IoT). Les principales caractéristiques de ce protocole sont de permettre aux utilisateurs de répéter leurs paquets plusieurs fois dans la même trame et d'utiliser l'annulation successive d'interférences (SIC) au niveau du récepteur pour décoder les paquets en collision. Le protocole IRSA simple peut *asympotiquement* atteindre le 1 [paquet/slot] optimal. Mais il existe encore de nombreux obstacles pour atteindre ces performances, en particulier lorsque l'on considère une longueur courte de trame. Dans ce rapport, nous étudions deux nouvelles variantes d'IRSA avec une longueur courte de trame, et nous optimisons leurs performances en utilisant une approche d'apprentissage par renforcement profond (DRL). Dans notre première variante, Random Codeword Selection-IRSA (RC-IRSA), nous considérons une approche IRSA avec sélection aléatoire de mots de code, où chaque mot de code représente la stratégie de transmission d'un utilisateur sur les slots. Nous appliquons un DRL pour optimiser le RC-IRSA : nous utilisons un modèle avec des réseaux de neurones multi-couches qui choisit les slots sur lesquels l'utilisateur envoie ses paquets. Notre approche DRL pour RC-IRSA est une nouvelle méthode d'optimisation pour IRSA utilisant une approche DRL et elle fonctionne comme une base pour notre proposition de deuxième variante d'IRSA, DS-IRSA.

Cette deuxième variante est un protocole d'écoute basé sur IRSA et entraîné avec un model d'apprentissage automatique pour synchroniser les nœuds pendant la transmission et éviter les collisions. Ainsi, nous proposons DS-IRSA, le protocole IRSA basé sur le DRL qui est composé de deux phases : une phase de détection, où les nœuds peuvent détecter le canal et envoyer de courts signaux de d'occupation du canal, suivie d'une phase de transmission IRSA classique. Nous utilisons un algorithme DRL pour optimiser ses performances. Notre protocole proposé a montré d'excellentes performances et atteint une performance optimale de près de 1 [utilisateur décodé/slot] pour les petites tailles de trame (≤ 5) slots et avec une durée de détection suffisante.

Mots-clés : Accès Aléatoire, mMTC, IoT, Coded Slotted Aloha (CSA), Slotted Aloha à Répétition Irrégulière (IRSA), Annulation Successive des Interférences, Évolution de la Densité, Apprentissage Automatique.

Contents

1 Introduction

The Internet of Things (IoT) is a system of interrelated devices connected to the Internet to transfer data among each other and with servers. It applies to more than just sensors or devices: it focuses on entire use cases. Smart buildings, connected cars, and industrial automata are examples of applications, where things need to "talk to each other", through complex interactions.

Indeed, driven by the necessity and the prominence of IoT applications in everyday life, massive Machine Type Communication (mMTC) has emerged as a fundamental part of future communications. mMTC is a fundamental use case of IoT, where a massive number of devices transmit short packets in a sporadic way to the base station (BS). The sporadic traffic pattern and the small size of transmitted packets are the new features of mMTC that make the existing connection protocols and technologies insufficient to support the massive connectivity. The inevitable need for new connection technologies for mMTC has triggered over the recent years a lot of interest, leading to countless new research directions to design efficient new protocols for mMTC applications. In turn, the need for such mMTC -supported protocols has led to the development of very sophisticated PHY and MAC layer technologies that can better exploit the wireless channel.

ALOHA-based solutions have arisen as a promising candidate for mMTC applications. These solutions provide a competitive performance compared to that of their coordinated counterparts in terms of throughput and reliability, leading the way for the application of modern RA protocols to many IoT scenarios of 5G and beyond [1]. The main idea of these protocols is to allow transmitters to send multiple copies of their packets towards the base station (BS). At the receiver, Successive Interference Cancellation (SIC) is applied to resolve collisions. It is possible to combine SIC with other advanced physical layer techniques to recover the packets.

Despite that the research behind the applications of modern random access protocols for mMTC has shown a remarkable gain in terms of supported network load and achieved throughput, further study is necessary to understand their true potential in IoT networks and beyond 5G. In complex telecommunication systems such as IoT, where millions of devices compete for network resources, recent Machine Learning (ML) techniques have shown the capability of creating effective transmission strategies that no human could discover. ML techniques help the system to learn how to adapt to a fluctuating environment and to automatically tune the protocol parameters.

In this work, we focus on one protocol of modern random access family, Irregular Repetition Slotted Aloha (IRSA) [2]. In IRSA, users are allowed to repeat their packets multiple times in the same frame. The number of packet repetitions is determined by a probability distribution, as in [2]. The objective is to use advanced machine learning techniques to optimize IRSA. Although, as indicated, the plain IRSA scheme can *asympotically* reach the optimal 1 [packet/slot]. But yet the problem is not solved, and the ultimate challenges are (a) to ensure correct decoding just below the load $G = 1$, (b) to ensure correct decoding in the *non-asymptotic* case, e.g. small frame sizes, for which the performance can be much lower than 1 [packet/slot] (see [3, Fig 4.]), (c) to avoid the dramatic performance decrease right around the threshold load.

In the aim to enhance the achieved throughput of IRSA with short frame length, we study two different variants of IRSA, where we apply a DRL approach to each one of them. This report is divided in two parts:

In the first part, we present RC-IRSA, an IRSA approach with random codeword selection, where each codeword represents the transmission strategy of a user on the slots (i.e., defining precisely on which slots the user will transmit, instead of just deciding how many replicas will be transmitted). Then, we apply a DRL approach to RC-IRSA by training a neural network to choose users' codewords, thus we obtain the RC-IRSA variant denoted Deep-RC-IRSA.

In the second part, we present DS-IRSA, a another new variant of the IRSA protocol that is based on sensing and that is also optimized with Deep Reinforcement Learning: it is a derivative of Deep-RC-IRSA with the learning the sensing protocol performed entirely through Deep Reinformancement Learning. With sensing, our transmission scheme is composed of two phases: a sensing phase itself, where active nodes send and attempt to sense short jamming signals with the intent of interacting with other nodes and potentially performing some (weak) form of coordination. Sensing is a generalization of carrier sense [4]. The second phase is as classical IRSA, but each node will choose an adapted transmission strategy to send its replicas, partly based on the information of the sensing phase.

In both approaches, Deep-RC-IRSA and DS-IRSA, we use a Deep Reinforcement Learning approach based on the DRL method Proximal Policy Optimization (PPO) [5]. Our obtained results through learning show an excellent performance of DS-IRSA for short frame length, compared to classical IRSA [6], Deep-IRSA [7] and Deep-RC-IRSA, but our results are still for short and limited frame lengths. [1]

Part I
An IRSA approach with Random Codeword Selection (RC-IRSA)

2 Introduction

In this part, we present RC-IRSA, an IRSA approach with random codeword selection. The codeword represents the transmission strategy of a user on the slots. We mathematically formalize an optimization problem for the performance of RC-IRSA for short frame size. We are able to optimize it heuristically for very short frame sizes by using differential evolution. Then, we detail the main contribution of the part, which is an application of Deep Reinforcement Learning to optimize RC-IRSA: it results in the training of a Deep Neural Network model that defines exactly the slots on which the user will transmit. The main goal is not to improve the performance of IRSA, but to illustrate that the fine slot selection can be achieved through DRL (and to study its limits), and to use it as a framework for the next variant, DS-IRSA. We compare the performance of our learning approach with the obtained performance of RC-IRSA through differential evolution and show that our proposed DRL approach indeed achieves very good performance for short frame size and approaches the optimal throughput values that we found by differential evolution. Our DRL approach works as a base for our proposed IRSA variant in the second part of this work.

3 Related Work

Many research studies have addressed the problem of designing adaptive Medium Access Control (MAC) solutions for IoT networks. One research direction to optimize the users' transmission strategy is to deterministically select the slots on which the users will transmit. This can be represented essentially by a vector of 0 and 1 with one value for each slot. When considering all the users, this implies to design a *codebook* (a *dictionary*, or simply a *code*) of *access codes*

[1] This report is derived from part of a thesis on wireless telecommunications, it is a slightly modified version of the eighth and the ninth chapters of this thesis [8].

(also called *sequences*, *protocol sequences* or *codewords*). The goal is to maximize the network capacity and avoid collisions. But alternately, codebooks can be used to simply identify active users.

As an introduction to the topic, we start with the classic work [9], which studies two types of superimposed codes: Zero-False Drop codes (ZFD) and Uniquely Decipherable codes (UD), and their applications in data communication. The paper shows several properties and construction methods over a wide range of parameter values. It also shows how a new class of codes, nonrandom binary superimposed codes, are constructed.

Their channel model is as follows: each node transmits a fixed zero-one sequence, and what is observed is another binary sequence of the same size, which is the *logical OR* of all the simultaneously transmitted sequences (the terminology "*sum*" is used in [9] as they are referring to a *Boolean algebra*, but we will use the term "superposition").

For a given small positive integer m, a codebook whose codewords satisfy the following condition will be said to be uniquely decipherable of order m, abbreviated UD_m: every superposition of up to m different codewords is distinct from every other superposition of m or fewer codewords. A simple example of a list of 7-bits codewords from [9]

$$s_0 = \begin{pmatrix}1 & 1 & 0 & 0 & 0 & 0 & 0\end{pmatrix} \qquad s_1 = \begin{pmatrix}1 & 0 & 1 & 0 & 0 & 0 & 0\end{pmatrix}$$
$$s_2 = \begin{pmatrix}0 & 1 & 0 & 0 & 1 & 0 & 0\end{pmatrix} \qquad s_3 = \begin{pmatrix}0 & 0 & 1 & 1 & 0 & 0 & 0\end{pmatrix}$$
$$s_4 = \begin{pmatrix}0 & 0 & 0 & 1 & 1 & 0 & 0\end{pmatrix} \qquad s_5 = \begin{pmatrix}0 & 0 & 0 & 0 & 1 & 0 & 1\end{pmatrix}$$
$$s_6 = \begin{pmatrix}0 & 0 & 0 & 0 & 0 & 1 & 1\end{pmatrix} \qquad s_7 = \begin{pmatrix}0 & 0 & 1 & 0 & 0 & 1 & 0\end{pmatrix}$$

contains no duplicated codewords. In addition to that, when augmented with all $\binom{8}{2} = 28$ pairwise superposition of codewords, it still contains no duplicates. Thus, this set of eight codewords constitutes a UD_2 code. Observe for instance that the superposition $\begin{pmatrix}0 & 1 & 0 & 1 & 1 & 0 & 0\end{pmatrix}$ can uniquely be obtained from $s_2 \vee s_4$.

One usage of such code in communications is the following: such a UD_m codebook can be used, with a distinct codeword for each node. We can imagine a slotted frame (different from the IRSA frame), where nodes can transmit jamming signals (instead of packets) when there is a 1 in their sequence. At the base station (or any receiver), an energy detector can be used to identify in which slots jamming signals have been transmitted. If less than m nodes have been active and transmitted their code, then the receiver can uniquely retrieve the identity of the nodes.

A related line of work in [10] presents a grant-free random access scheme for short-packet communication on a collision channel without feedback, where user identities are conveyed through their activity patterns, i.e. their codewords. SIC is not used. Assuming that the population size is N and at most d devices can be active at any given time, the study shows that group testing codes can be used to design random access protocol sequences with minimal length. They consider the frame-synchronous and asynchronous cases, where each user has a corresponding codeword of length t. The receiver is assumed to know the user's codewords. Their approach leads to codes of length $t = \Theta(d \log N)$ for the asynchronous case. They minimize the complexity by minimizing the length of the codeword needed to decode d active devices at a time. Because SIC is not used, the scheme can also be used as a user identification scheme, e.g. if users transmit one bit, or if they use jamming signals in slots.

The previous codebooks were designed to perform user identification, an area where also *compressive sensing* has been proposed [11]. In the context of IRSA, it has also been shown that one can design novel efficient multi-detection algorithms that also take into account the physical layer [12], although in this report we will stay focused on an idealized transmission model. But codebooks have also been used for packet transmission: the concept of using codebooks has

been famously introduced in early work, along with the *collision channel without feedback* (and without frames) in [13]. Codebooks can be naturally used to specify on which slot users should transmit packets as in [13], where in addition, there is no frame boundary, hence each user starts transmitting at an arbitrary slot. [13, Eq. (15)] gives an example for two users who use the codebook:

$$s_0 = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix} \qquad s_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \end{pmatrix}.$$

SIC is not used. But as one can see, no matter on which slots the users will start transmitting, both packets will be recovered.

In the case of IRSA, it has been already proposed to construct a large codebook with one codeword for each user with *graph-defined* IRSA (G-IRSA) [14], based on the design of an LDPC code. This approach is not fully scalable when the proportion of active users decreases [15, Sec. IV.A]. This area is also linked to Code-Domain NOMA [16].

Recent work goes further and constructs capacity-achieving codebooks [17] that assume SIC. One of their example is for three users [17, SectionV.C]:

$$s_0 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad s_1 = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \quad s_2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Here, with SIC, all three users will get recovered, no matter in which slot they start. Additionally, instead of simply repeating the initial packet at the positions of the 1, some coding can be applied, such as MEBC-coding [13], and the same codebook can be used to transmit encoded packets, giving a global data rate of 1 packet per slot.

Back to the framed version of random access methods: one can define an M-Interference Cancelling code (or codebook), denoted M-IC code, of length n as a set of codewords such that: each active user has at least one successful transmission during every n consecutive time slots provided that the number of simultaneously active users is less than or equal to M after iterative decoding with SIC.

In this spirit, the authors of [15] have introduced the design of deterministic random access codes for the ultra-reliability region, targeting a packet loss rate less than 10^{-5}. They considered simple deterministic variations of CRDSA (D-CRDSA), where the number of repetitions is fixed (differing from IRSA, where the degree is drawn from a distribution), and compared them to the original random versions.

The study shows that larger codewords with more repetitions might maximize the user population and minimize the packet loss rate in their numerical experiments and at low load. Notice that they use M-IC codes with more than M users, but show that they are still performing better than selecting random slots (up to a certain point). The work studies codes based on Steiner systems and discuss why prior codes based on LDPC designs [14] have drawbacks for URLLC scenarios. Designing M-IC codes that support a large user population is a hard problem. The paper also points out the limitations of any approach that is based on superimposed codes, for instance, the codes which satisfied the 3-IC condition could only support relatively small user populations as shown by the provided lower bounds on the supportable user population for 3-IC codes.

Finding M-IC codes, as constructing any kind of code in general, is not easy. Currently, there are some examples of constructions, but there exists no mathematical constructions nor automated ways to construct *all* such codes. To avoid the complexity of finding IC codes with the current tools or search algorithms, some research work used machine learning techniques. In [18], The authors apply a deep reinforcement learning (DRL) based algorithm to search for IC codes, using specific metrics and reward functions according to the underlying mathematical constraints. They model the construction process of an M-IC code with N codewords each of

length n as an episodic symbol-filling game based on Markov Decision Processes (MDP). Each episode represents the construction of a codebook, C and it begins with the state where the codebook is empty, i.e. is constituted of empty codewords. Each step of the episode selects an action sequentially, to add ℓ bits to one codeword which is still shorter than the frame length. Each episode ends when the codebook is full, i.e. every codeword covers the frame length. At the end of each episode, C is evaluated by one metric $m(C)$ that counts the number of subsets of $m \leq M$ codes that cannot be fully decoded, and a reward $r(C)$ for this episode is linearly derived from $m(C)$. In particular, C is an M-IC code if and only if $m(C) = 0$. This MDP formulation is then used to search for codebook in a tree manner, with a Monte-Carlo Tree Search (MCTS), and a Deep Neural Network model is classically used to guide the search. The search results have indicated that the algorithm can efficiently discover IC codes, while the simulation results have shown that the discovered IC codes can produce significantly lower failure probability than random slot selection under the same latency requirements (again, even for several active users greater than M), and thus are more suitable for URLLC. The introduced codes in [18] are not necessarily optimal. A future research direction is to design a general construction for close-to-optimal IC-codes.

They also give an example of 4-IC code in [18, Section III.C]:

$$
\begin{array}{lll}
s_0 = \begin{pmatrix}0&0&0&0&1&0&0\end{pmatrix}, & s_1 = \begin{pmatrix}0&0&0&0&0&1&0\end{pmatrix}, & s_2 = \begin{pmatrix}1&0&1&0&0&0&0\end{pmatrix}, \\
s_3 = \begin{pmatrix}0&0&0&1&0&1&0\end{pmatrix}, & s_4 = \begin{pmatrix}1&1&0&1&1&0&0\end{pmatrix}, & s_5 = \begin{pmatrix}0&0&1&1&0&0&0\end{pmatrix}, \\
s_6 = \begin{pmatrix}0&0&0&0&1&0&1\end{pmatrix}, & s_7 = \begin{pmatrix}1&0&1&0&0&1&1\end{pmatrix}, & s_8 = \begin{pmatrix}0&1&0&0&1&0&1\end{pmatrix}, \\
s_9 = \begin{pmatrix}0&1&1&0&0&0&0\end{pmatrix}, & s_{10} = \begin{pmatrix}1&0&0&0&0&0&0\end{pmatrix}. &
\end{array}
$$

Many other works have proposed codes for random access. One example is Learn2MAC [19] where the users have their own individual codebooks, and optimize the transmissions (maximize their individual utility) with online learning. More precisely, codewords are randomized for each user, but also a probability of selection is associated with each codeword: the codeword probability distribution is updated by the online learning algorithm. We also became recently aware of quite recent and interesting work combining random access and machine learning [20].

In the next section, we shed the light on related studies that consider IRSA with short frame length, and we introduce our prior work on applying a deep learning approach on short frame length IRSA.

3.1 Prior Work on Short-Frame Length IRSA

3.1.1 Short-Frame Length IRSA

In the literature, there is a lack of precise results for optimizing IRSA for smaller frame sizes. The main challenge to optimize IRSA with short frame size, which is the realistic scenario, is that there is not a simple mathematical approach to track the decoding process and compute the expected number of decoded users as with density evolution for asymptotically large frame lengths.

One of the most accurate performance estimates for IRSA with short frame length is possibly in [3], where a short-frame (SF) approximation for the packet loss rate of IRSA is particularly suitable for very short frames, up to 50 slots, has been introduced. The idea of this work is to write the packet loss rate as a function of all possible stopping sets in order to compute the exact value of the packet loss rate. It is then possible to write and solve numerically an optimization problem, as with density evolution. Even for small frames, the number of stopping sets would rapidly become intractable, yielding unmanageable complexity. To limit it, the number of considered stopping set in [3], has been limited to only the stopping sets that have e_m edges,

where the parameter e_m has been introduced as an upper bound on the number of edges in the set.

One illustrative result is that short-frame length IRSA can be far from reaching a throughput near 1 [packet/slot] with near 0% PLR. The results in [3, Fig. 3 & Fig. 4] show unfavorable throughput/packet loss trade-off, despite that the simulations were done for a fixed number of users $M = 5$ in [3, Fig. 3] (PLR is 5% for $G = 0.5$) and for a constant load $G = 0.4$ in their [3, Fig. 4] (PLR is 4 %).

3.1.2 Our Prior Work: Deep-IRSA

One of the motivations for the work presented in this section is to improve IRSA performance, in particular when the size of the frames is finite, hence the assumption of "asymptotically infinite frame length" is far from being verified. To address this, our prior work in [7] has proposed Deep-IRSA, which uses a deep learning framework to optimize the performance of IRSA. We focused on the short frame length scenarios for different network parameters, with also the ability to enable two options: one which allows re-transmissions and another which has user priority classes, where one class has a higher priority than the other class. We applied Deep Reinforcement Learning techniques to solve this problem.

We first present how we applied DRL to IRSA with Deep-IRSA, as a predecessor to the more advanced variants, which will be presented in the next part. A Markov Decision Process (MDP) model is adopted: the action of one user is the selection of a degree (after that, slots are still randomly selected), the reward is a global reward that simply counts the number of decoded users. In Deep-IRSA, additionally, there is some observed state. The main idea is that users can receive feedback from the BS to inform them about the number of collisions in the past frame and if their own transmissions succeeded. Then, each user adds an internal state to the received feedback: its class of service and whether it is a re-transmitter or not when any of these options is enabled. The feedback, together with the internal state, both constitute the observed state of

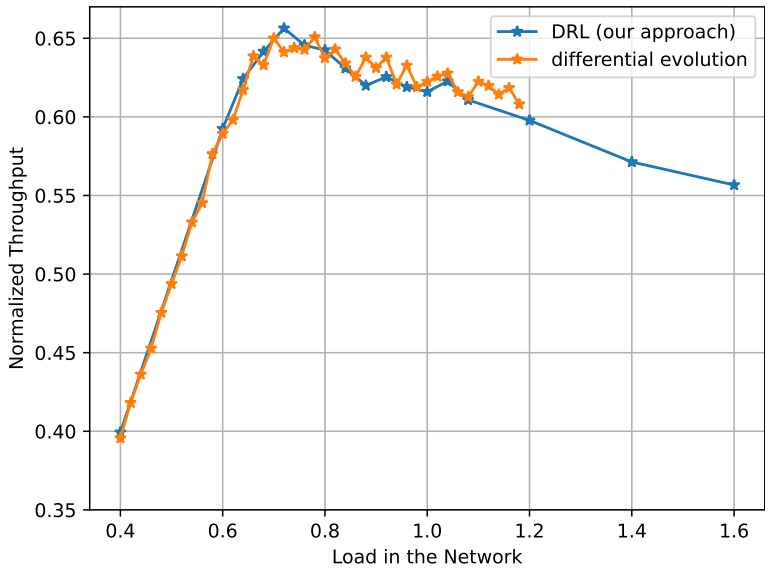


Figure 1: Throughput for (Relatively) Short Frame-Length, for Deep-IRSA, and for Differential Evolution with Simulations

one user. It is expected to influence the choice of user actions. Deep Reinforcement Learning with the Proximal Policy Optimization (PPO) [5] is used for training: a Deep Neural Network (DNN) model takes the state as input and outputs an action corresponding to a degree. PPO is a policy-based method, and the DNN model outputs a stochastic policy, e.g. the probability to select each degree. Without options, the DNN essentially outputs a degree distribution that is adjusted depending on the load (and its behavior becomes more complex when there are options). Note that the application of PPO to IRSA will be extensively explained in the next part of this report.

We are interested in the obtained maximum performance in case of small frame size and a few numbers of users, $M = 50$. We varied the load and used Deep-IRSA as an optimization method. For reference, we independently and numerically optimized distributions by the use of simulations. This becomes an obvious stochastic optimization problem: using the *Sample Average Approximation* (SAA) [21], we could just define an approximated objective function as the average of 100 simulations, that we can optimize with any method, and we opted for differential evolution. The results of the throughput obtained with both methods are reproduced in Fig. 1. The throughput we obtained for DRL shows a comparable performance against the differential evolution results. The Fig. 1 shows that IRSA with a short frame length is suffering from low throughput, e.g. a maximum of 0.65, whether it is optimized by differential evolution or DRL approach.

One way to enhance this work is to incorporate slot (or a group of slots) selection, which enables the DNN model to intelligently select slots instead of using random degree-then-slot selection. In effect, this would be equivalent to generating codewords.

4 RC-IRSA: IRSA with Random Codeword Selection

In this variant study, we are interested in the use of codebooks for the transmission phase of IRSA, as done for instance in G-IRSA [14] or [15]. However, unlike them, we do not intend to allocate deterministically a codeword to each user. We assume that the users are completely undifferentiated. In that case, it might be natural to have a codebook shared by all users (or by a subset of users), and to randomly select a codeword according to some distribution (as done in Learn2MAC [19] for instance). We denote this version of IRSA as *Random Codeword selection IRSA* (RC-IRSA). Notice that this version is not expected to be more efficient than individual codeword assignment, but it will serve as a basis for the improved method in the next part.

In this section, we provide a general definition of RC-IRSA and the related optimization problem formalization to find optimal IRSA codebooks and codeword probability distributions. An IRSA codebook is a set of codewords, where each codeword represents the positions of the slots where the user sends their replicas. Let $\mathcal{S}$ denote a binary $\{0, 1\}$ scheduling code for deterministic access, which consists of ω codewords of length w, i.e., $\mathcal{S} \triangleq \{s_0, s_1, ..., s_{\omega-1}\}$. To transmit a packet, the user will use a codeword $s_i \triangleq (s_i[0], s_i[1], ..., s_i[w-1])$, for $i = 0, 1, 2, ..., \omega - 1$.

If a user uses the codeword s_i to transmit its packet in the frame, it transmits at the slot t if and only if $s_i[t] = 1$, and thus it repeats the same packet $\phi_i \triangleq \sum_{j=0}^{N-1} s_i[j]$ times in the frame, where N is the number of slots. Each codeword $s \in \mathcal{S}$ is associated with a probability of selection π_s. The codeword probability distribution $(\pi_s)_{s\in\mathcal{S}}$ replaces the degree probability distribution $(\Lambda_i)_{i=0...L}$ of IRSA.

In line with the classical IRSA system, the aim is to find the best codebook probability distribution that maximizes the system throughput at a given network load G.

Given a codebook $\mathcal{S} = \{s_0, s_1, ..., s_{\omega-1}\}$, with $s_i = (s_i[0], s_i[1], \ldots, s_i[w-1])$ for all $i \in$

$\{0, 1, \ldots, \omega - 1\}$ and such that for all $t \in [0, w-1]$, $s_i[t] \in \{0, 1\}$. The optimization problem is:

$$\begin{aligned} &\underset{(\pi_{s_i})}{\text{maximize}} && T(\mathcal{S}, \pi_{s_0}, \ldots, \pi_{s_{\omega-1}}) && (\mathcal{P}_7) \\ &\text{subject to} && 0 \leq \pi_{s_i} \leq 1, \quad \forall i \in \{0, 1, ..., \omega - 1\} \\ & && \sum_{i=0}^{\omega-1} \pi_{s_i} = 1 \end{aligned}$$

where T is the throughput. Note that, in this work, we express the throughput as "the average number of decoded users". ω is the total number of codewords. Note that the throughput computation should be adjusted to take into account codewords probabilities.

We extend the optimal formulation of the optimization problem in $(\mathcal{P}_7)$, to a system that has a different but finite number of classes $K = (C_0, ..., C_{n-1})$, and a fixed global load G. For each class $c \in K$, there is a unique codebook. We can rewrite the optimization problem:

$$\begin{aligned} &\underset{(\pi_{s_{c,i}})}{\text{maximize}} && T_c(\mathcal{S}_c, \pi_{s_{c,0}}, \ldots, \pi_{s_{c,\omega-1}}) && (\mathcal{P}_8) \\ &\text{subject to} && 0 \leq \pi_{s_{c,i}} \leq 1, \quad \forall i \in \{0, 1, ..., \omega - 1\} \\ & && \sum_{i=0}^{\omega-1} \pi_{s_{c,i}} = 1 \end{aligned}$$

where $s_{c,i}$ is the codeword i of the codebook $\mathcal{S}_c$ and c is the class index and $c \in K$.

4.1 A Mathematical Analysis of RC-IRSA with 2 users and 2 slots

In this section, we introduce a scenario of RC-IRSA protocol in the simplest case: 2 users are competing for 2 slots. We present a mathematical analysis of this simple classical IRSA scenario. We suppose an IRSA scenario of 2 slots and 2 competing users A and B. Both users use the IRSA protocol with a maximum repetition degree of 2 and with a minimum repetition degree of 0. The set of possible codewords[2] is $\mathcal{S} = \{00, 01, 10, 11\}$, with the codeword probability distribution $(\pi_{00}, \pi_{01}, \pi_{10}, \pi_{11})$. The 1 in a codeword means that the user sends a packet on the slot, and the 0 means that the user does not send a packet on the slot. As a consequence, the number of all possible combinations of the codeword choices of both users is $2^4 = 16$. The six combinations that allow decoding both users are shown in Fig. 2. The probability that both users are decoded after sending on two slots is given as:

$$P_s = 2\pi_{01}\pi_{10} + 2\pi_{01}\pi_{11} + 2\pi_{10}\pi_{11} \tag{1}$$

In the case of 2 users, the number of decoded users is simply $2P_s$, hence we can equally maximize P_s. By using the method of Lagrange multipliers, we can find the probability to use

[2]For the ease of presentation, here, we index the codewords with their binary representation (e.g. "01" for $(0, 1)$) instead of indexing them with integers.

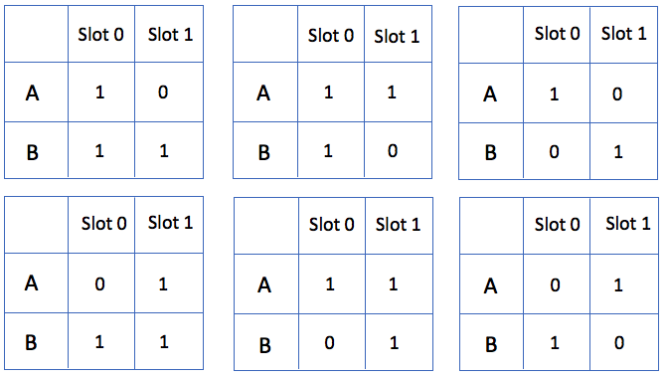


Figure 2: The transmission patterns that lead to the decoding of two users A and B sending on 2 slots

each codeword such that we maximize the success probability:

$$\begin{aligned}
&\mathcal{L} = 2\pi_{01}\pi_{10} + 2\pi_{01}\pi_{11} + 2\pi_{10}\pi_{11} - \lambda\left(\pi_{00} + \pi_{01} + \pi_{10} + \pi_{11} - 1\right) \\
&\frac{\partial \mathcal{L}}{\partial \pi_{00}} = -\lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \pi_{01}} = 2\pi_{10} + 2\pi_{11} - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \pi_{10}} = 2\pi_{01} + 2\pi_{11} - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \pi_{11}} = 2\pi_{01} + 2\pi_{10} - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \lambda} = \pi_{00} + \pi_{01} + \pi_{10} + \pi_{11} - 1 = 0
\end{aligned} \tag{2}$$

Solving the set of equations in Eq.(2), and also checking the boundary conditions, gives the optimal probability to use each codeword as: $\{\pi_{01} = \pi_{10} = \pi_{11} = p, \pi_{00} = 1 - 3p\}$. Choosing $p = \frac{1}{3}$ will maximize the success probability:

$$P_s = 2 \cdot 3 \cdot \left(\frac{1}{3}\right)^2 = \frac{2}{3} \approx 0.667\ldots \tag{3}$$

and the throughput is given by:

$$T = 2 \cdot P_s = 2 \cdot \frac{2}{3} \approx 1.334 \quad \text{decoded users/frame}$$

4.2 A Mathematical Analysis of RC-IRSA With M Users and N Slots

Let us consider a system of M users, competing to send their packets on N slots using IRSA protocol. Each possible transmission of each of the M users on the N slots is represented by a vector of ones and zeros, considered as a possible codeword of length N. The ones refer to the transmission positions on the slots, and the zeros represent the slots where the user has not transmitted. Thus, the possible codewords that could be sent by a user, form a codebook

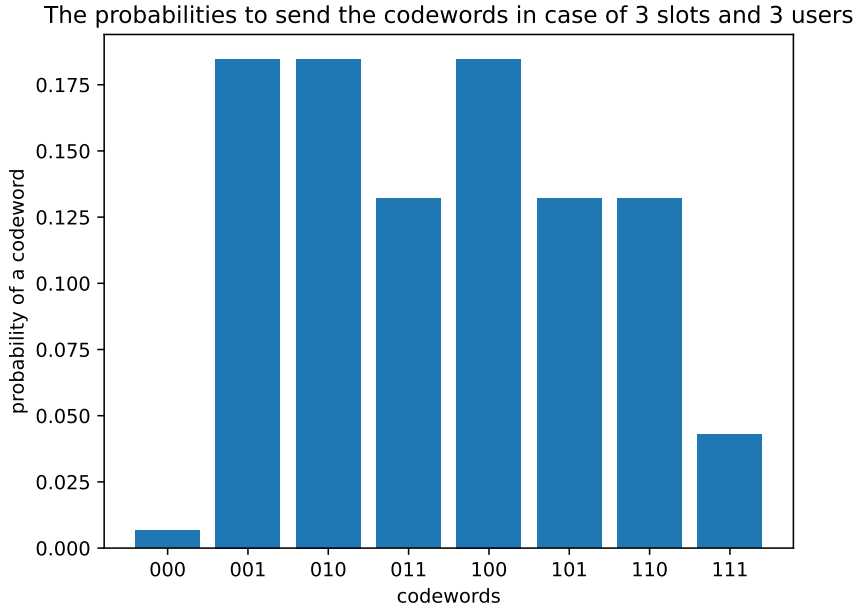


Figure 3: The probability to send each possible codeword in a system of 3 slots, 3 users

$\mathcal{S} = \{s_0, s_1, ..., s_{\omega-1}\}$, which has: $\omega = 2^N$ codewords. The total number of possible codeword combinations that could be sent by the M users on the N slots is calculated as: $N_{com} = \omega^M$.

We define D as an M-dimensional array. Each element $d_{i_1,i_2,..,i_M} \in D$ represents the number of decoded users after using the codewords $s_{i_1}, s_{i_2}, ..., s_{i_M}$. In other words, each element of D is the number of decoded users after using one combination of the selected codewords by the M users in the system. Each codeword $s_i \in \mathcal{S}$ has a length N. The probability of decoding all users in the frame (the probability of success) is calculated as follows:

$$P_s = \sum_{i_1=0}^{\omega-1} \sum_{i_2=0}^{\omega-1} \sum_{i_3=0}^{\omega-1} ... \sum_{i_M=0}^{\omega-1} \pi_{i_1} \times \pi_{i_2} \times \pi_{i_3} \times ... \times \pi_{i_M} \times d_{i_1,i_2,i_3,..,i_M} \quad (4)$$

where π_i is the probability to send the codeword $s_i \in \mathcal{S}$.

Note that the number of possible codewords ω grows exponentially with the number of users M. Therefore, it becomes very difficult to mathematically evaluate the probability of success (described in Eq. 4) as the number of slots $N \geq 3$. Alternatively, the probability to use each codeword can be computed numerically using differential evolution.

A small illustrative example could be for $M = 3$ users sending on $N = 3$ slots. The number of possible codewords for one user is: $\omega = 2^3 = 8$ codewords. The total number of possible combinations is $N_{com} = 8^3 = 512$. Computing manually 512 values of the D array, and optimizing manually Eq. (4) is a complicated task. Thus, we use differential evolution to compute the probability to use each codeword such that the throughput is optimized.

Fig. 3 shows the probability to use each possible codeword after solving the optimization problem $\mathcal{P}_7$, We note from the figure that the probabilities to send one replica in one of the three slots are identical: $\pi_{100} = \pi_{010} = \pi_{001} = 0.1847$. The probabilities to send two replicas on two of the three slots are also identical: $\pi_{110} = \pi_{101} = \pi_{011} = 0.132$. Thus, an optimized codebook

users \ slots	2	3	4	5	6
2	1.333332	1.714285	1.866634	1.935483	1.968247
3	0.969525	1.673334	2.288013	2.615522	2.791376
4	0.899124	1.440759	2.082369	2.789802	3.278239
5	0.864823	1.367171	1.922436	2.546374	–

Table 1: The obtained throughput of RC-IRSA via differential evolution for different number of users and slots

of eight codewords has been generated, with the probability to use each codeword. Note that we can extract the coefficients of the optimal degree probability distribution from the generated codebook as: $\Lambda_0 = \pi_{000}, \Lambda_1 = \pi_{100} + \pi_{010} + \pi_{001}$, $\Lambda_2 = \pi_{110} + \pi_{101} + \pi_{011}, \Lambda_3 = \pi_{111}$, but the inverse is not possible in general. In other words, we cannot extract an optimal codebook from an optimal degree distribution. Note that the codeword degree distribution happens to be the same as a degree distribution for this example, but this is not expected to be true in other variants of RC-IRSA. The number of decoded users for this example is ≈ 1.673 decoded users/frame. Table. 1 shows the obtained throughput by using differential evolution for different scenarios.

In this context, two questions arise: first, can we apply a DRL approach to find an optimal codebook that maximizes the throughput? The second question is whether extra knowledge about the users can help to synchronize the nodes, such that adding a sensing phase before the IRSA transmission may lead to optimizing the obtained codebook and maximizing the throughput. In the next section, we answer the first question, while we explore the other question in the next part.

4.3 Deep RC-IRSA: A Deep Learning analysis of IRSA with 2 users and 2 slots

In prior work, [7], we optimized IRSA using Deep Reinforcement Learning to obtain Deep-IRSA, where we learned the degree distribution. In this section, we use a deep learning approach to find the optimal codebook that maximizes the throughput in case of IRSA scenario of 2 users and 2 slots. We consider the same methodology to learn optimal codewords' probability distributions for IRSA instead of degree distributions. The application of DRL is done according to the following principles:

- IRSA is recast in the DRL framework, it becomes a multiagent system, where each node is an agent.
- The action of each agent, is essentially the codeword selection $s_i \in \mathcal{S}$.
- At each episode, we generate a state, where we add a source of randomness which acts as a *source of entropy* to help the model to generate different codewords for the same actual state. Indeed, keeping the same state for both users in all the episodes will lead to choosing statically the same actions, as the agents always observe the same value, and the neural network model is deterministic. On the contrary, changing the state helps to make the actions change dynamically to attain the optimal values.

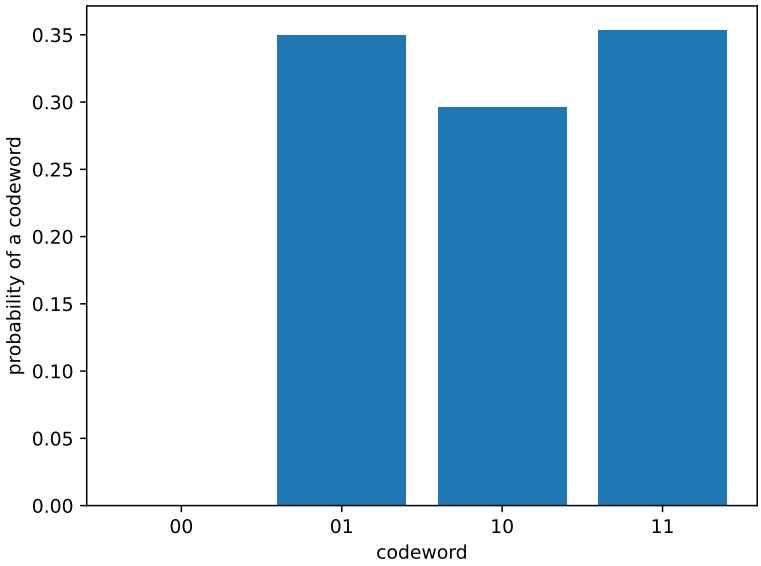


Figure 4: The probability to use each codeword in case of 2 users-2 slots system

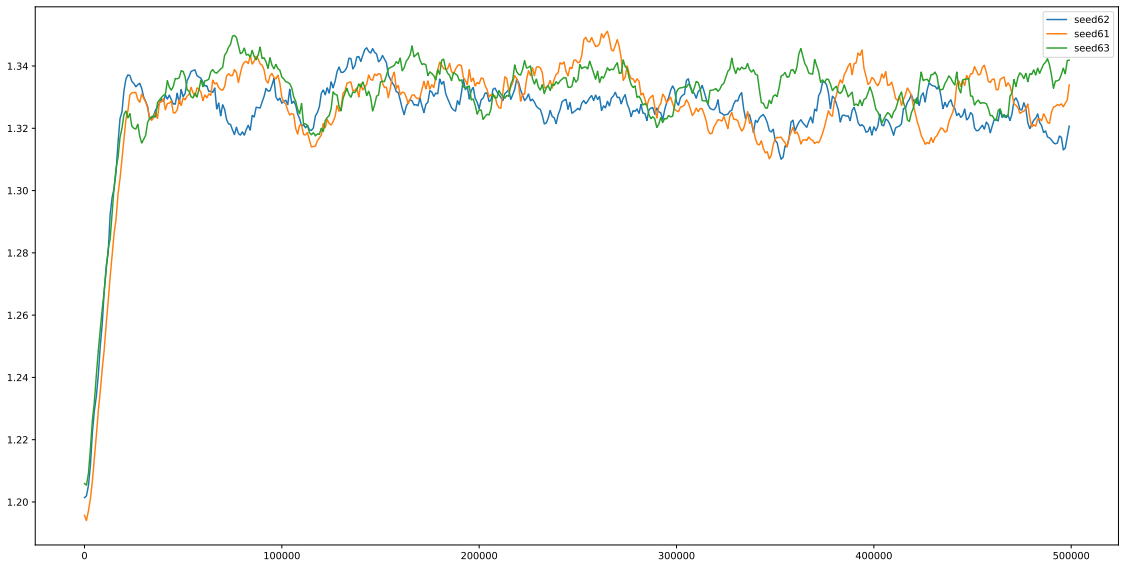


Figure 5: The convergence of IRSA throughput in case of 2 users-2 slots system

users \ slots	2	3	4	5	6
2	1.333117	1.701885	1.853383	1.927235	1.959373
3	0.967106	1.658842	2.271073	2.581683	2.754746
4	0.897425	1.436308	2.065276	2.724494	3.208501
5	0.857731	1.361922	1.908074	2.536552	3.160870

Table 2: The obtained throughput of RC-IRSA via the deep learning approach for different number of users and slots

4.4 Deep RC-IRSA with M Users and N Slots

Fig. 4 shows the probability to use each codeword. Note that we are computing *empirical probabilities* observed from episodes after the convergence of our DRL approach (the last episodes). The DRL model does not give actions probabilities as an output, but it gives the selected slots. Each empirical probability to use an action (a codeword) has been obtained by counting how many times the same action was used after the model has converged (e.g. in the last episodes).

The resulting observed probabilities from the DL approach are:

$$\{\pi_{01} = 0.34, \pi_{10} = 0.3, \pi_{11} = 0.36, \pi_{00} = 0\},$$

while we obtained $\left\{\pi_{01} = \pi_{10} = \pi_{11} = \frac{1}{3}, \pi_{01} = 0\right\}$ in the theoretical approach.

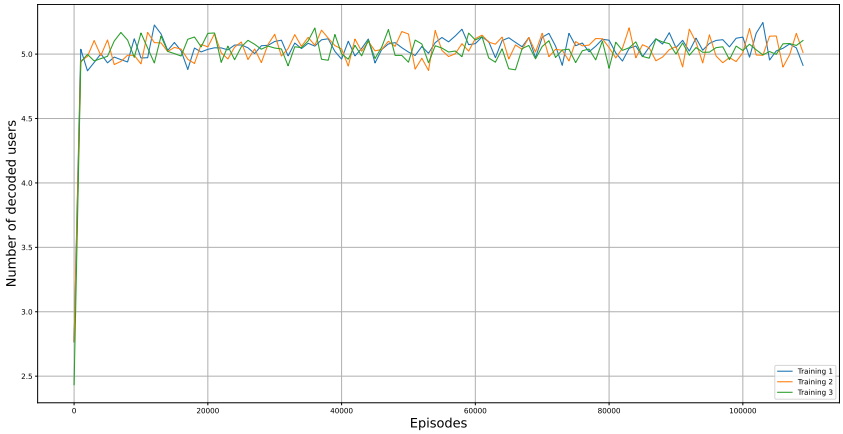
Fig. 5 shows the convergence of the DRL approach towards the optimal throughput $T \approx 1.334$. Note that we used a smoothing filter that takes the arithmetic average of each value with its neighbor.[3] The size of the smoothing is $20,000$, which means that the average is taken on each successive $20,000$ values. The figure shows that our DRL approach converges after approximately $30,000$ learning episodes towards the optimal value that we found in the Section. 4.1.

Fig. 4.1 compares three different simulations for the same scenario, but with different seeds. The figure shows that the DRL with different seeds attains the optimal throughput.

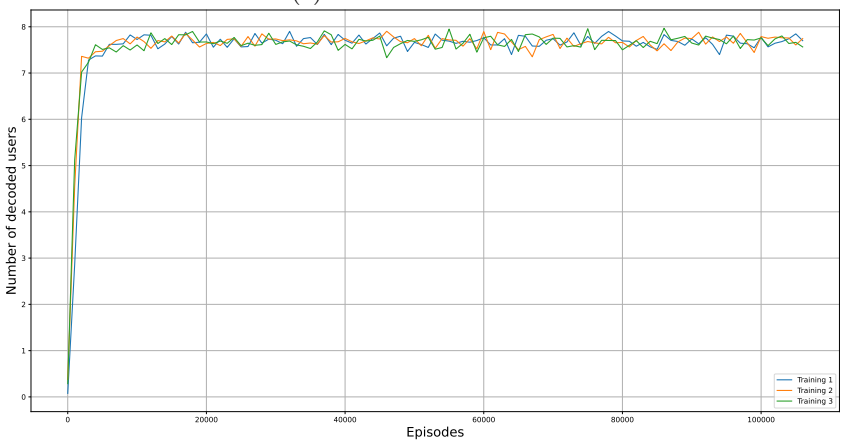
In this section, we show more results of using Deep RC-IRSA to optimize the throughput in the case of the IRSA scenario of ≥ 2 users and ≥ 2 slots. Table. 2 shows the IRSA throughput when the DRL approach is used to optimize the performance. Comparing these values with the throughput values in Table. 1 that were obtained through differential evolution, confirms that our DRL approach works perfectly as it achieves the same optimal values obtained by differential evolution.

Fig. 6 shows the training of our DRL model for a different number of users and slots. We did 3 different trainings with three different seeds. In each case, we note that our DRL approach converges fast towards a value of the throughput of around 50%. In other words, the number of decoded users that we obtain with our DRL approach is always around half of the competing users. For instance, the obtained throughput in Fig. 6a, for a scenario of 10 users and 10 slots, is around 5. Note that, the optimal values of throughput obtained through differential evolution, for the scenarios where the number of users is equal to the number of slots, were always approximately 0.5 (see Table. 1). It becomes very complicated to mathematically compute the optimal throughput when the number of slots is ≥ 2. In addition, solving the described optimization problem in $(\mathcal{P}_7)$ through differential evolution for more than ≥ 10 users becomes an extremely complicated task as the number of codewords combinations grows exponentially with the number of users.

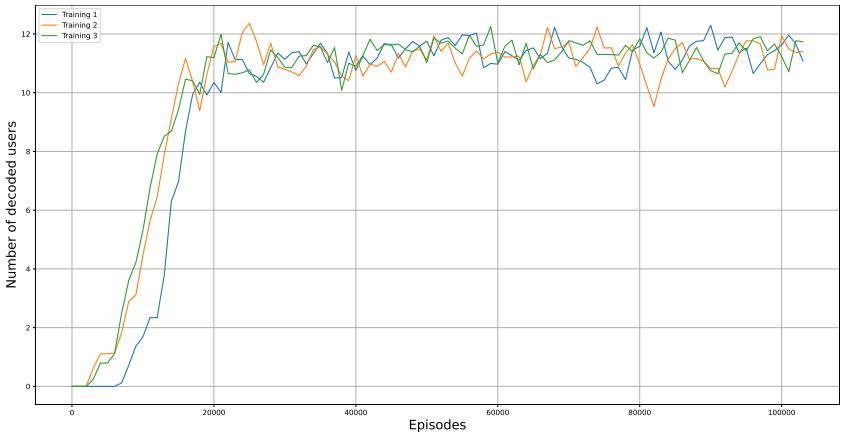
[3]For more information about the smoothing function, see: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.uniform_filter1d.html



(a) 10 users and 10 slots



(b) 15 users and 15 slots



(c) 25 users and 25 slots

Figure 6: The convergence of IRSA throughput for different number of users and slots

Our DRL approach has proven to achieve a throughput around 50% for a scenario of $\leq$ 25 users competing for $\leq$ 25 slots. For Deep RC-IRSA with more than 25 users, our DRL approach does not seem to work correctly for two reasons: First, adding more slots will increase, exponentially, the action space of the neural network. The second reason is related to large stopping sets. As we do not have any restrictions on the taken actions after initialization and after the beginning of the training, the neural network could try some actions that include many 1 and a few zeros. This will lead to blocking many users in large stopping sets and ultimately, they will have almost always a zero throughput and leads to a sparse network problem. One solution to resolve such a problem is to force a penalty (negative reward), when the taken action is not favorable, and also increase the number of learning episodes.

Part II
Deep Sensing IRSA (DS-IRSA)

5 Introduction

One of the main roles of MAC protocols is to decide the transmission strategy of the connected nodes. Given the limited network resources and the variable environmental conditions, the design of efficient MAC protocols is one of the main challenges for current networks. The primary goal of ML techniques applied to random access is to exploit communication resources such that they provide the best possible connectivity for all demands. By definition of *random access*, randomness will inevitably, sometimes, provide situations where decoding is not possible. But we observe that if nodes had just a small amount of knowledge on how the other nodes intend to transmit, the performance could be improved. In this part, our main idea is to introduce some form of sensing between the nodes: active nodes not only transmit to the base station but can also sense the channel in order to get information about the presence and the activity of other transmitting nodes. This is in line with very common and popular methods from classical random access, such as Carrier Sense Multiple Access (CSMA) [22, Section 4.4], and its numerous generalizations [4]. A major question is now: how to design or adapt sensing to IRSA?

Motivated by a very recent work that exploits learning not just to learn the parameters, but to learn the entire methods of interactions [23] or even entire programs, our initial goal is to *learn to interact*, in the sense of *learning a sensing protocol* entirely through Deep Learning. Because of the high complexity of this task, in this part, we mostly detail initial steps towards these goals and provide some interesting results.

For that aim, we introduce Deep-Learning and Sensing-based IRSA (DS-IRSA), a new variant of IRSA protocol which is based on sensing and that with optimizing through Deep Learning, with a derivative of Deep-RC-IRSA. With sensing, our transmission scheme is composed of two phases: a sensing phase, where active nodes send and attempt to sense short jamming signals with the intent of interacting with other nodes and potentially performing some (weak) form of coordination. Sensing is a generalization of carrier sense [4]. The second phase is as classical IRSA, but each node will choose an adapted transmission strategy to send its replicas, partly based on the information of the sensing phase. We use a Deep Reinforcement Learning approach based on the DRL method Proximal Policy Optimization (PPO) [5]. Our obtained results through learning show an excellent performance for short frame IRSA, compared to classical IRSA [6]. Deep-IRSA [7] and Deep-RC-IRSA (from the previous part), but our results are still for short and limited frame lengths.

6 System Model for Sensing-based IRSA (S-IRSA)

We introduce a version of IRSA with sensing (Sensing-based IRSA, S-IRSA). The main new modifications to the basic IRSA system model are listed below:

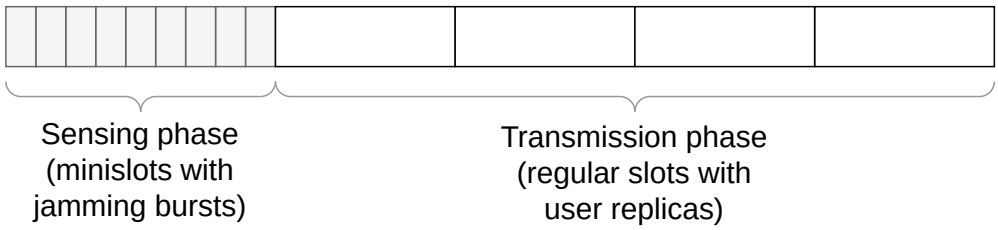


Figure 7: Extended frame: minislots of a sensing phase followed by regular slots of a transmission phase

- In classical IRSA and RC-IRSA, each active user decides on which slots it will transmit at the beginning of each frame: either directly with a list of slots (RC-IRSA), or indirectly by picking first a degree from a degree distribution and then picking several slots at random accordingly (classical IRSA). In our proposed approach, we extend the usual frame of IRSA by prefixing it with as phase of *minislots*, as represented in Fig. 7. We denote this phase as the *sensing phase*. The sensing phase is followed by the IRSA transmission phase where users will select their transmission strategies (i.e. the slots where each user will send its replicas).

- The goal of the sensing phase is that users collect information about the environment by only sensing the channel and optionally sending signals on the minislots. Then, they exploit this extra information to attempt to synchronize and avoid collisions.

- In the sensing phase, on each minislot, a user decides to be either active or stay inactive. If it is active on the minislot, it then sends a *jamming signal* for the duration of the minislot. We denote these transmissions as *jamming bursts*. In the literature, as in [4], they use the terms "black bursts", "bursts", "busy tones", ..., etc. The behavior of one user, during the entire sensing phase, can be summarized as a zero-one sequence, with one value for each minislot.

- In the sensing phase, at each minislot: each user also senses the energy of the jamming transmissions on the channel. In this work, we start with an idealized model, where 1) the users are able to exactly measure the amount of energy of the simultaneously transmitted jamming bursts, hence are able to exactly detect the number of transmitting users on the minislot, 2) the users are operating in full-duplex mode, therefore, are fully able to sense the channel while simultaneously transmitting a jamming burst, 3) they are able to complete the sensing for one minislot, before they have to decide to transmit or not a jamming burst for the next minislot. As a result of these assumptions, all nodes in the network share a common knowledge: the outcome of the sensing phase that can be summarized by an integer sequence, with one value for each minislot corresponding to the number of users simultaneously transmitting a jamming burst on this minislot.

- In the transmission phase: the nodes operate as in an RC-IRSA frame, by transmitting replicas of their data packets in the slots that they selected. The difference with RC-IRSA is that their slots' selection can be influenced by the collected information in the sensing phase. Note that we do not assume that the users are doing any sensing during the transmission phase itself.

- As with IRSA, the BS listens to the signals of the transmission phase and decodes the replicas of each user with SIC. We do not assume that the BS listens to the transmissions during the sensing phase.

Note that there are additional variations of the sensing system model that we are not exploring in this work:

- Half-duplex instead of full-duplex: with half-duplex, a user cannot sense the channel if they transmit a jamming burst.
- Binary sensing: the user can only detect if at least one user is transmitting on the same minislot, but it has no information about the number of transmitting users. This is the assumption in many other works using jamming [4].
- Implicit sensing phase: instead of having an explicit sensing phase with minislots, the system could adopt a frameless version, and each user could use energy measurements on the (previous) IRSA slots.
- More precise energy model: instead of assuming that the received power is uniformly identical for all users, a path-loss model could be adopted.

7 Insights on Exploiting Sensing Information and Design Sensing-Based Protocols

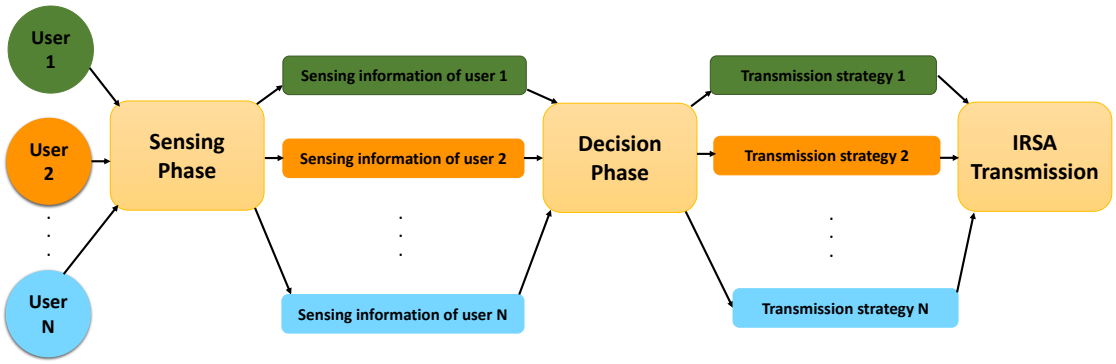


Figure 8: The general problem of IRSA with sensing

In this section, we discuss the general problem of introducing a sensing phase to IRSA and pose the problem of how to exploit it in the best possible way. Fig. 8 shows the general problem of IRSA with sensing: in the sensing phase, the users need to select their actions on each minislot (transmit or not a jamming signal); then with the knowledge of their actions and the observations of the energy on the channel (their sensing information), they have to decide a transmission strategy (the sets of slots where they will transmit replicas). Finally, the classical transmission of replicas is performed, and the BS decodes iteratively with SIC as for IRSA.

In our system model, the initial active users on a frame are actually *undifferentiated*. This is because we are in a context of mMTC communication where active nodes are essentially a small

	minislot 0	minislot 1	minislot 2	minislot 3
A	1	0	0	1
B	1	0	0	1
C	0	1	0	1
D	1	0	1	0
Total Sum of Energy	3	1	1	3

Figure 9: A simple illustrative example of the sensing phase of S-IRSA with 4 users and 4 minislots

random subset of a very large set of devices and as a design choice, we assume that they have an identical behavior (i.e. not dependent on a pre-defined node identifier for instance). Active users start by interacting, sensing the channel, and collecting some information. This collected information should be exploited to differentiate the users in order to enhance their transmission strategies. This could be done, by taking some decisions about, for instance, the slots where users send their replicas. The decision phase could be a specific algorithm or, as we opted, could involve a neural network model.

7.1 Differentiation: A Simple Illustrative Example

To illustrate how might sensing help to improve performance, Fig. 9 shows a simple example of the sensing phase of 4 users interacting on 4 minislots. In the example, user A is transmitting jamming bursts on minislots 0 and 3, user B as well, etc. In our model, all the users have the same information about the number of transmitters in each minislot through sensing. Indeed, the sequence of observations on the minislots $\begin{pmatrix}3 & 1 & 1 & 3\end{pmatrix}$ is known by all users.

An arguably interesting idea would be to take advantage of this globally known information, for instance, to help users to select some slots. We denote users who have transmitted alone a jamming signal on some minislots as *sole transmitters*[4]. This is the case for users C and D. Then observe that users C and D can fully be designated by the sentences "the sole transmitter on minislot 1" and "the sole transmitter on minislot 2" respectively. We also denote the consequence that they can be uniquely designated as being *fully differentiated*.

This differentiation can be exploited, for instance, by defining deterministic rules for slot selections. Indeed, imagine the following convention:

- First, all minislots with sole transmitters are considered to be fully differentiated (by all users).

[4]This is akin to the *singletons* in transmission phase with IRSA, but observe that SIC cannot be used in the sensing phase.

- Some slots of the transmission frame will be reserved for the exclusive use of those sole transmitters: one slot for each of them. The transmission slots are reserved in the same order as the minislots, i.e. the very first slot of the transmission frame will be reserved for the first user who became a sole transmitter in the sensing frame.

- The remaining slots will be used by the remaining users, that have not been fully differentiated. Quite naturally, non-differentiated users could use classical IRSA to compete for the remaining slots in the transmitting frame.

In the example of Fig. 9, C is the first sole transmitter (minislot 1), D is the second one (minislot 2), hence the rule would grant them respectively the first slot and the second slot of the following transmission frame. A and B are never sole transmitters, hence they might use classical IRSA: they select a degree, and then they randomly select some slots starting from slot 3 till the end of the transmission frame. This helps to improve performance.[5]

The entire example relies on the fact that some nodes are fully differentiated on some minislots. A natural question arises, can other similar rules be established? And can users be *partly differentiated*? The sensing phase contains some information that could be the source for differentiation and could be translated into some kind of coordination between the users. The major issue is to find the best exploitation of this collected information to create a kind of coordination between users. Simultaneously, in the sensing phase, what is the best protocol for interacting and selecting minislots for transmitting jamming bursts?

The entire random access literature (or at least the RA literature with sensing) constructs protocols to achieve solutions for previous questions; however, these protocols are constructed explicitly and IRSA is not considered. A current trend is to introduce machine-learning techniques to automatically adapt protocols to different scenarios. In our case, using ML can offer one decisive advantage: one might not need to specify *which* sensing information to use, *how* to use it, and *what* is the best interaction in the sensing phase. Instead, one can envision automatically learn to how to correctly interact and perfectly exploit sensing data. This is the ultimate goal of our work in this part.

In the next section, we briefly show some examples of possible definitions of *partial differentiation* to showcase the fact there are more options than just full differentiation.

7.2 Examples of Full Differentiation and Partial Differentiation

In this section, we first show how users can be fully differentiated. For the sake of the discussion, we assume here that the minislot phase is arbitrarily large. We observe that the sensing channel model of Section 6 is similar to the "collision channel with feedback" common in usual random access (RA) and contention resolution algorithms (CRA) [22, 24]: the difference is that a successfully transmitted packet in a RA protocol is here equivalent to be a sole transmitter. From this perspective, a collision is equivalent to having two or more transmitters on the same minislot. Hence, any classical slotted random access protocol can be easily transposed into a sensing phase protocol, where users transmit jamming bursts instead of packets and repeat them in case of "collisions" following the rule of a CRA.

In the case of an arbitrarily large sensing frame, each user will continue to interact until it becomes a sole transmitter. With any sensible RA, in the end, all users will be fully differentiated.

[5] It is not immediately clear that the performance will be improved, but looking at Table 2, we see that without sensing: with 4 users on 4 slots one expects to decode ≈ 2.06 of them, while with 2 users on 2 slots, the expectation is ≈ 1.33. Thus, on the example, applying the proposed conventions, we expect to decode ≈ 2 (differentiated users) + 1.33 (classical IRSA expected gain for 2 users competing for 2 slots) = 3.33 users, compared to ≈ 2.06 without sensing, a clear gain.

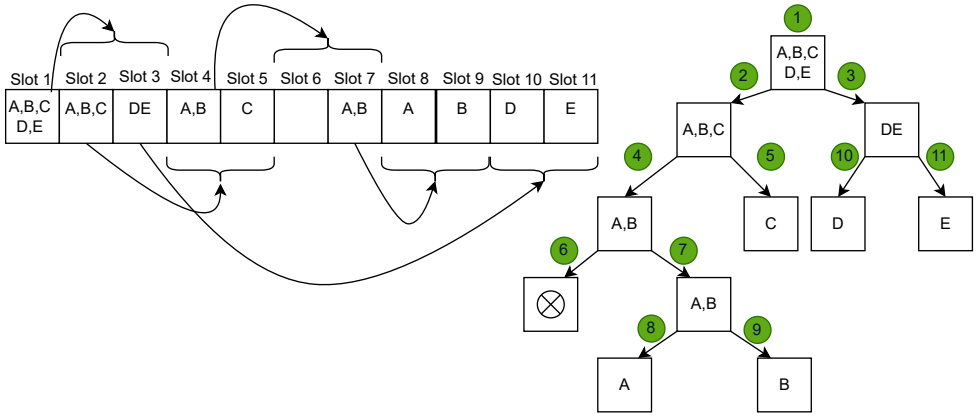


Figure 10: An example of a splitting algorithm with 5 users A, B, C, D, E

Using the transmission slot allocation rule of Section 7.1, the transmissions in the transmission frame follow a TDMA (Time-Division Multiple Access) schedule, and IRSA is not required. This will maximize the efficiency of the transmission phase (no collision) but this will be at a cost of a very large sensing frame.

It is also possible to introduce a clear definition of the concept of *partial differentiation* in the case of specific protocols. Indeed, it is the case for some CRA algorithms, namely splitting algorithms (also called tree algorithms) [22]. Bertsekas and Gallager describe them as follows [22, Sec. 4.3.1]:

> "The first splitting algorithms were algorithms with a tree structure. When a collision occurs, say in the kth slot, all nodes not involved in the collision go into a waiting mode, and all those involved in the collision split into two subsets (e.g., by each flipping a coin). The first subset transmits in slot $k+1$, and if that slot is idle or successful, the second subset transmits in slot $k+2$. Alternatively, if another collision occurs in slot $k+L$ the first of the two subsets splits again, and the second subset waits for the resolution of that collision." [22, Sec. 4.3.1]

An example of the full operation of a splitting algorithm is represented in Fig. 10, with five users A, B, C, D, E, and 11 slots. As shown in the figure, all the users are being fully differentiated at the slot 11. Note that it is also possible to stop the algorithm before it ends, in other words, if the number of slots is not sufficient to differentiate all users, the algorithm stops before reaching the goal of differentiating all users.

One property of such algorithms is that the set of users that are in a collision on one slot is always included in the set of users that were in collision in one same previous slot (or they have never transmitted before). Using this property, it is then possible to designate users through *the last slot on which they have retransmitted* and partition them into disjoint sets. In Fig. 10, for instance, users could be put in separated sets as follows:

- After the end of slot 3, the partition is obtained from slots 2 and 3. Thus, it is: $\{A, B, C\}, \{D, E\}$.
- If we consider slot 9, users A, B and C are fully differentiated but not the two others, so the partition is then: $\{A\}$, $\{B\}$, $\{C\}$, $\{D, E\}$

Hence, at any point in time, users are always partitioned into clear subsets, so we can define clearly the notion of "partially differentiated". A fully differentiated user is a user that is only in a subset of size 1, and a *partly differentiated* user is a user that is in a subset of size > 1. In the case where all users still fall in only one set, they are still all undifferentiated.

Hence, it is possible to introduce partial differentiation through some specific sensing protocols for S-IRSA. However, we do not know if it would be the best approach to improve the performance of the transmission phase, nor how to best exploit the potential of this partial differentiation. This is the motivation for adopting a machine learning approach to sensing protocols.

8 DS-IRSA: Deep Learning, Sensing-based IRSA

In this section, we present our approach to the version of IRSA with sensing, S-IRSA, that has a sensing phase followed by a transmission phase. We use a DRL approach to optimize its performance. In the next sections, we explain in detail the protocol design and how we apply the DRL approach. Furthermore, we give a simple example to clarify the concept of user differentiation.

8.1 DS-IRSA: Applying DRL to S-IRSA

DS-IRSA (Deep Learning, Sensing-based IRSA) is an approach that applies DRL to S-IRSA in the same way as Deep-IRSA is a DRL approach to classical IRSA. It does this by extending the DRL approach in Sec. 4.3, and by adding a sensing phase. A neural network model is used to select actions. Details are provided later, but an overview of the actions in each phase is as follows:

(a). The sensing phase, which consists of slots of small duration, is referred to as "minislots", where users can send jamming bursts. In DS-IRSA, we consider that each user, on each minislot, will either transmit a jamming burst or not: this decision is taken by the neural network model. Each user has also the capability to sense the channel and measure the total energy on the minislot. The information gradually collected during the sensing phase is fed online to the neural network model. The goal is that the model exploits the sensing information to introduce some differentiation between users, and synchronizes the users for the IRSA transmission phase. As discussed previously, a user could be differentiated at the sensing phase, if it has not collided with any other user, and it is left to the model, to induce some kind of differentiation and to exploit it.

(b). In the transmission phase, users have to select slots for IRSA transmission. This is done by the same neural network model that now chooses the actions of active users based on the sensing information. Each action is now a codeword of zeros and ones that specifies where the user should send its replicas on the slots. Our model represents a policy (as it outputs actions) and it is trained through a DRL Policy Gradient Method detailed in the next section.

8.2 Policy Gradient Methods

In this work, we use one of the policy gradient methods, Proximal Policy Optimization (PPO) [5] to optimize our proposed IRSA variant (DS-IRSA). In this section, we explain in detail the principles of PPO closely following its original description in [5]. Policy gradient methods [25] are a type of RL techniques that relies upon optimizing parametrized policies concerning the expected return (long-term cumulative reward) by gradient ascent. Other classical methods are action-value methods, such as Q-Learning [25, Chap. 6], that learn the values of actions and then

select actions based on their estimated action values. Policy gradient methods are not action-value methods, and they learn a parameterized policy that can choose actions without a value function and are trained typically with a variant of the Policy Gradient Theorem [25, Sect. 13.2].

In PPO, a value function is still used to learn the policy parameters, but its aim is only to improve training convergence, and it is not directly involved in action selection. In the following, we further explain the principles of PPO. In PPO, the policy denoted π_θ, is a stochastic policy that takes the observed state s_t from the environment and suggests an action a_t to take as an output. It is given by a neural network model parametrized by a set of coefficients (weights) θ, that determines the probability $\pi_\theta\left(a_t \mid s_t\right)$ to select action a_t. During training, RL episodes are run, and the stochastic gradient ascent is used to iteratively improve the policy. This requires an estimator of the policy gradient from the sampling obtained through the episodes. The most commonly used gradient estimator of the policy gradient is derived from the Policy Gradient Theorem [25, Sect. 13.2] and has the form [5, Eq. 1]:

$$\hat{g} = \hat{E}\left[\bigtriangledown_\theta \log \pi_\theta\left(a_t \mid s_t\right) \hat{A}_t\right] \tag{5}$$

where $\hat{A}_t$ is an estimation of the value of the *advantage* function at time step t. The definition and the value of $\hat{A}_t$ is the difference between the discounted long-term cumulative reward R_t up to time step t, and the baseline estimate $b_s(t)$, that is an estimation of the expected return from the time step t onwards. They define $\hat{A}_t$ as:

$$\hat{A}_t = R_t - b_s(t) \tag{6}$$

The value of $\hat{A}_t$ reflects how much better was the impact of the taken action (this is given by the cumulative reward R_t) based on the expectation of what normally should happen ($b_s(t)$), considering that we are in the state s_t.

In Trust Region Optimization (TRPO) Method [5], the log function in Eq. 6 is replaced with the division by the old value of the policy $\pi_{\theta_{old}}$, and by adding a KL constraint. The KL constraint is a measure of how the old policy is different from the updated policy, and it is added to make sure that the updated policy does not move far away from the current policy, as follows:

$$\underset{\theta}{\text{maximize}} \qquad \hat{g} = \hat{E}\left[\frac{\pi_\theta\left(a_t|s_t\right)}{\pi_{\theta_{old}}\left(a_t|s_t\right)}\hat{A}_t\right] \tag{7}$$

$$\text{subject to} \qquad \hat{E}_t\left[KL\left[\pi_\theta(.|s_t), \pi_{\theta_{old}}(.|s_t)\right]\right] \leq \delta \tag{8}$$

The issue of the optimization with the TRPO method is that it adds extra overhead to the optimization process, so additional modifications are required.

Proximal Policy Optimization (PPO) [5] is an online policy gradient algorithm which optimizes the clipped surrogate objective (explained later in Eq. (9)). To understand the principle of PPO, first, let us introduce $r_t(\theta)$, which is simply the probability ratio between the newly updated policy output and the output of the old version of the policy:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

The central optimization objective behind PPO is the expectation operator of the minimum of two functions: the normal policy gradient objective $r_t(\theta)\hat{A}_t$ and a truncated version of $r_t(\theta)$ between $[1-\epsilon, 1+\epsilon]$:

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}(t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}(t)\right)\right] \tag{9}$$

The term $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}(t)$, modifies the surrogate objective by clipping the probability ratio, which removes the incentive for moving $r_t(\theta)$ outside the interval $(1-\epsilon, 1+\epsilon)$. The minimum of the clipped and unclipped objective is taken, so the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective. This means that the change in probability ratio that would make the objective improve is ignored (when the advantage function is positive), and it is included only when it makes the objective worse (when the advantage function is negative) (see [5], Fig. 1).

The final loss function that is used to train the neural network in PPO is as follows:

$$L_t^{CLIP+VS+S}(\theta) = \hat{E}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right] s(t) \right] \tag{10}$$

where: $L_t^{CLIP}(\theta)$ is the clipped PPO objective. $c_1 L_t^{VF}(\theta)$ is used to update the baseline $b_s(t)$, which specifies how beneficial it is to be in a certain state or in other words, it computes the expected return from the current state onwards. The intuition behind combining these two terms in the same objective function is that the value estimate function shares some of its parameters with the policy function. The last term $c_2 S\left[\pi_\theta\right] s(t)$ is to guarantee that the agent does enough exploration during the training process. c_1 and c_2 are coefficients, S denotes an entropy bonus and $L_t^{VF}(\theta)$ is s a squared-error loss of the value function $\left(V_\theta(s_t) - V_t^{targ}\right)^2$.

In this chapter, we use the DRL algorithm PPO and its implementation by OpenAI and others (stable baselines [26]). We chose this method as it has the stability and reliability of trust-region methods.

In the next section, we describe our DRL application to DS-IRSA.

8.3 Learning Environment and DRL Architecture

The application of DRL is done according to the following principles:

8.3.1 The Environment

IRSA is recast in the DRL framework, it becomes a multiagent system, where each node is an agent. The environment is the available physical resource, in this case, i.e., the channel, where time is divided into two phases, a sensing phase and a transmission phase (see Fig. 7), and each phase is divided to correspond to a sub-frame divided into slots or minislots. Every user interacts with the environment by taking actions and receiving rewards. The training of both phases, the sensing phase, and the transmission phase, is done simultaneously.

8.3.2 The Actions

The set of actions of an agent $m \in M$, is represented as A_m, and it is essentially composed of two parts:

- The sequence of actions in the sensing phase $A_m^{\text{sens.}}$ which has a length L, where L is the total number of minislots. Each value of this vector is a zero if the agent does not transmit a jamming signal, or a one if the agent transmits a jamming signal. In other words, the action $A_{m,t}^{\text{sens.}}$ of an agent m is to transmit if $A_{m,t}^{\text{sens.}} = 1$ at time t, or wait if $A_{m,t}^{\text{sens.}} = 0$.
- The action of the transmission phase, $A_m^{\text{trans.}}$, which represents the selection of the slots where the agent will transmit its replicas. In other words, the codeword selection: $s_i \triangleq (s_i[0], s_i[1], ..., s_i[N-1])$ for $i = 0, 1, 2, ..., N-1$ and N is the total number of slots. s_i represents the choice of slots of the agent m on where it will transmit its replicas. The agent m transmits on the slot t if and only if $s_i[t] = 1$

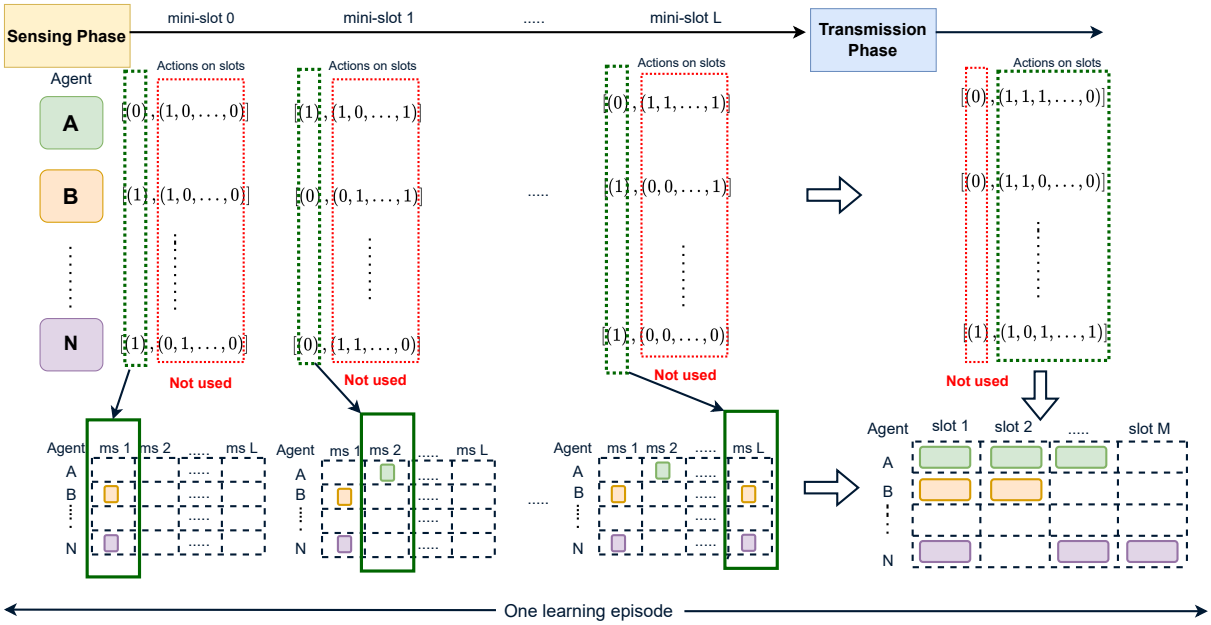


Figure 11: One learning episode of DS-IRSA with agent A, agent B, ..., agent N

In our implementation, a single neural network is implementing the policy. In Fig. 11, we show the action selection process during one episode. During the sensing phase, for each agent $m \in M$ and for each minislot $i \in L$, the neural network outputs an action $A^{\text{sens.}}_{m,i}$ to be executed on the minislot i. Note that the output of the model is always composed of two parts: $A^{\text{sens.}}_{m}$ and $A^{\text{trans.}}_{m}$, but only one of them is used, and the other is ignored depending on the current phase.

Fig. 11 shows that, during the training of the sensing phase, actions in green rectangles (the actions on the minislots $A^{\text{sens.}}_{m}$) are the only part of the action that is considered and applied on each minislot $i \in L$ separately, minislot by minislot. The second part of the action (the actions on the slots $A^{\text{trans.}}_{m}$), in red rectangles, is not used in the sensing phase. On the other hand, in the transmission phase, the second part of the action (the actions on the slots $A^{\text{trans.}}_{m}$), which represents a codeword of zeros and ones is applied on all the slots at the same time for all agents $m \in M$.

8.3.3 The States

The state of an agent $m \in M$ is a combination of some observed values and with additional random input values acting as entropy sources:

- The first component of the state is the observed energy on the minislots by the agent, which is represented by a vector of length L of integer values between 0 and M. Recall that L is the maximum number of minislots and M is the total number of agents.
- The second component of the state is the number (index) of the current minislot.
- The third component of the state is the actions that were taken by the user for previous minislots.

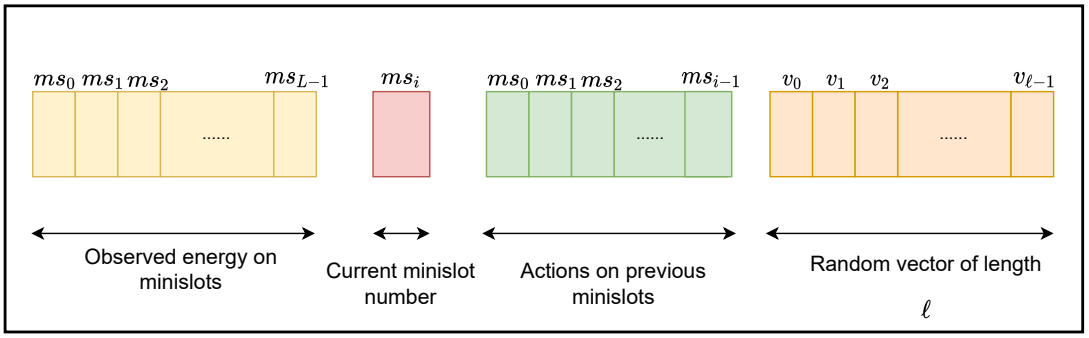


Figure 12: State computation of DS-IRSA

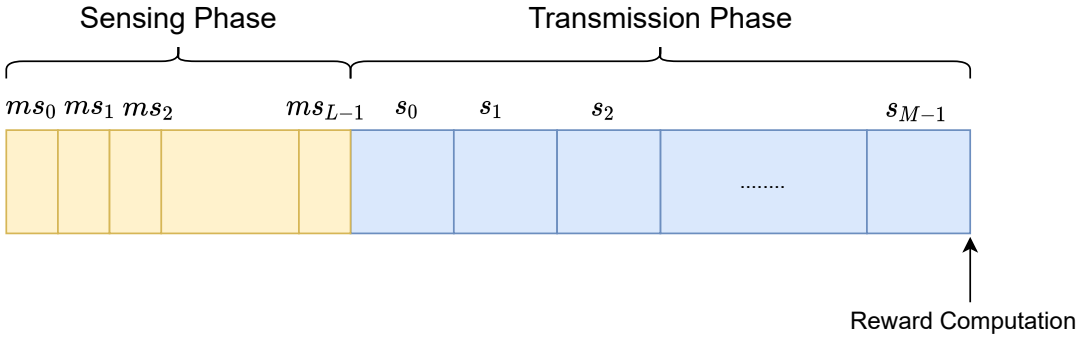


Figure 13: Reward computation for DS-IRSA

- The fourth component of the state is a random vector of a fixed length ℓ. In our simulations, we choose a vector of length $\ell = 10$ that has random values between $[0, 15]$. It acts as an entropy source to help the model output different codewords in the same state[6].

  In Fig 12, we show how the state is constructed.

8.3.4 The Reward

The reward: Let R_m be the reward that the agent $m \in M$ obtains at the last time slot M. The reward depends on the action of the agent m, $A_m(t)$ and other agents' actions $A_{m'}(t)$. The reward for the agent m is defined as:

$$R_m = P_N \tag{11}$$

where P_N is the number of decoded packets at the end of the episode, after the slot N and the iterative decoding at the BS, where N is the total number of slots in the frame. The discount factor is set to 1. Fig. 13 shows the reward computation time. The reward is computed at the end of the frame. Before the end of the frame, the reward is set to 0.

[6]observe that otherwise, with 0 minislots for instance, the model would output the same policy for all users, which cannot be optimal. Using $\ell = 10$ and values in $\{0, 1, \ldots 15\}$ is more than a sufficient number of entropy sources so that this is no longer limiting the performance.

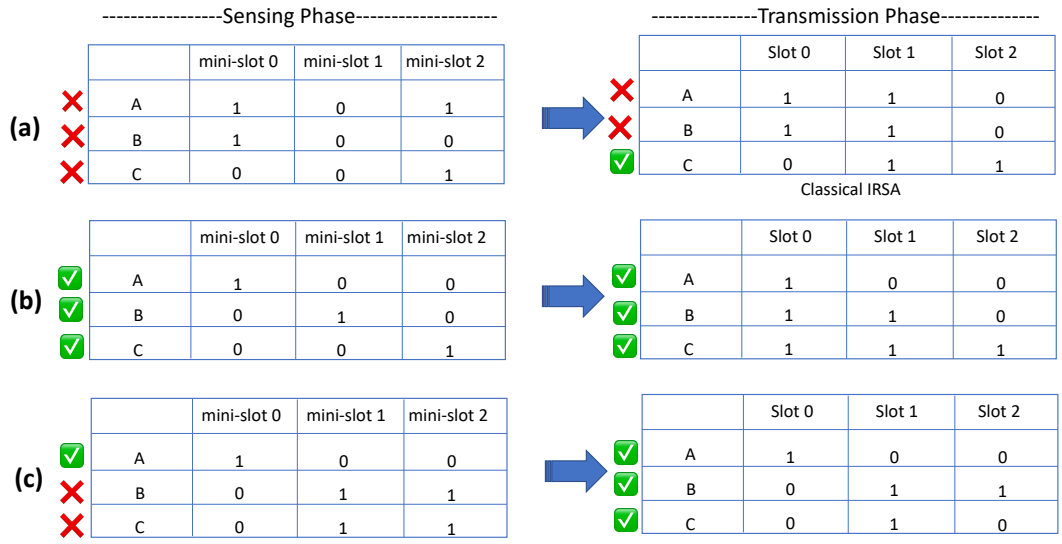


Figure 14: An example of DS-IRSA: (a). none of the users have been identified during the sensing phase. (b). all users have been identified during the sensing phase. (c). two out of three users have been identified during the sensing phase.

8.4 An Example of Differentiating the Users Using DS-IRSA

Fig. 14 shows three possible scenarios of DS-IRSA of 3 users, 3 slots and 3 minislots.

Fig. 14.a shows the failure of fully differentiating any of the three competing users, as none of these three users has succeeded to send a jamming burst in the sensing phase, without being colliding with the two others. In this case, we might get only little (if any) extra information from the sensing phase and the transmission phase, which can have close (or identical, it is an open question) performance to the one of a classical IRSA transmission. Another extreme case is shown in Fig. 14.b, where the three users have succeeded to be fully differentiated in the sensing phase. In this case, a slot could be reserved for each of the three users (see also Section 7.2). Note that as we employ SIC, more replicas could be sent, as Fig. 14.b shows in the transmission phase. Slot 2 has been reserved for user C, slot 1 is also reserved for user B. Note that user C can send other replicas (on slot 1 and slot 2) as its packet is assumed to be correctly removed from its reserved slot 3. It is also the case of user B, who sends twice on slot 2 and on slot 1.

The third final illustrated case in Fig. 14.c shows an intermediate case where one user (user A) is fully differentiated in the sensing phase and the two other users (users B and C) are not differentiated since all their actions are identical. In this case, the sampling strategy outlined in Section 7.2 prioritizes the differentiated user A by reserving him the slot 1. Users B and C have no choice than competing to send on slots 2 and 3 as in the classical IRSA scenario, and they cannot avoid the risk of collision with each other.

9 A Mathematical Analysis of S-IRSA with 2 Users and 2 Slots

In this section, we provide a mathematical analysis of the performance of the case of S-IRSA with 2 users and 2 slots and an arbitrary number of minislots. It will serve later as a benchmark for DS-IRSA; the same reasoning could be applied for S-IRSA with more slots but still 2 users. We do not have mathematical results for the cases with three users or more (which we believe to be significantly more complex).

9.1 A Sensing Phase with One Minislot

We consider S-IRSA with 2 users competing for 2 slots with a sensing phase of only one minislot. Adding one minislot in the sensing phase changes the problem as follows: assume that π_0 is the probability to send 0 in the minislot (i.e. to be inactive). The users could be differentiated, if they send different bits[7] on the minislot, $(0,1)$ or $(1,0)$, otherwise, they could be decoded with a probability $P_s = \frac{2}{3}$ as in the case without sensing (see Section 4.1 and Eq. (3)). Thus, we can write the new success probability as follows:

$$P_s = \frac{2}{3}\left[\pi_0^2 + (1-\pi_0)^2\right] + 2\pi_0 \cdot (1-\pi_0) \tag{12}$$

Following the same approach in Eq. 1, gives an optimal value of $\pi_0 = \frac{1}{2}$, which in turn increases the probability of success to $P_s = \frac{5}{6}$. Note that, in this part, we express the throughput as "the average number of decoded users" instead of the definition that is usually used with IRSA analysis. In other terms, we express it in "decoded users/frame" instead of "decoded users/slot". The throughput is obtained from P_s and is:

$$T = N \cdot P_s = 2 \cdot \frac{5}{6} \approx 1.67 \quad \text{decoded users/frame}$$

Note that in this specific case, adding a sensing phase of one bit helps to increase the throughput by 23.8% compared to the case without sensing in Section 4.1.

9.2 A Sensing Phase with Two Minislots

In this section, we add another minislot to the sensing phase described in the previous section. We also observe an important property for the case of two users: although each user should decide minislot by minislot their action on the next minislot, there are only two possible outcomes: either both users made the same choice of action, or they made a different choice of actions. In the first case, the minislot cannot bring any differentiation, and the users can just ignore what happened on the minislot. In the last case, one user is fully differentiated and as a consequence, the other one as well: then it does no longer matter what are the next actions of the users – and conversely, even if the users later ignore what happened on the minislot, they still would be fully differentiated.

In both cases, both users can ignore the feedback from the previous minislot(s) and still implement an optimal strategy. Thus, they can select their minislot transmission strategies at the beginning of the minislot phase: in this case, we can denote their jamming burst transmission strategy for the minislot phase as a "minislot codeword".

The two users will have $2^{\text{minislots}} = 2^2$ possible minislot codewords to use in the sensing phase. The users will not be differentiated if their activity in the sensing phase is represented by the

[7] Note that the nodes send jamming signals, but we use the term "bits" for the ease of explanation

same minislot codeword, i.e., $(00,00)$, $(01,01)$, $(10,10)$ or $(11,11)$. There are $16-4=12$ possible combinations of minislot codewords that allow both users to be differentiated. Suppose that: $\pi_{00}, \pi_{01}, \pi_{10}, \pi_{11}$ are the probabilities of using the minislot codewords: $00, 01, 10, 11$ respectively. The success probability is obtained by considering that if both users are fully differentiated it will be 1, and if they are not, the minislot had been useless and the success probability will be given by Eq. (3), i.e. $\frac{2}{3}$. Hence, it is given by the following equation:

$$P_s = 1 - \left(\pi_{00}^2 + \pi_{01}^2 + \pi_{10}^2 + \pi_{11}^2\right) + \frac{2}{3}(\pi_{00}^2 + \pi_{01}^2 + \pi_{10}^2 + \pi_{11}^2) \tag{13}$$

By using the method of Lagrange multipliers, we can find the probability to use each minislot codeword such that we maximize the success probability:

$$\begin{aligned}
&\mathcal{L} = 1 - \left(\pi_{00}^2 + \pi_{01}^2 + \pi_{10}^2 + \pi_{11}^2\right) + \frac{2}{3}\left(\pi_{00}^2 + \pi_{01}^2 + \pi_{10}^2 + \pi_{11}^2\right) - \lambda\left(\pi_{00} + \pi_{01} + \pi_{10} + \pi_{11} - 1\right) \\
&\frac{\partial \mathcal{L}}{\partial \pi_{00}} = -2\pi_{00} + \frac{4}{3}\pi_{00} - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \pi_{01}} = -2\pi_{01} + \frac{4}{3}\pi_{01} - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \pi_{10}} = -2\pi_{10} + \frac{4}{3}\pi_{10} - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \pi_{11}} = -2\pi_{11} + \frac{4}{3}\pi_{11} - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \lambda} = \pi_{00} + \pi_{10} + \pi_{01} + \pi_{11} - 1 = 0
\end{aligned} \tag{14}$$

we get the following solution: $\left\{\pi_{00} = \pi_{01} = \pi_{10} = \pi_{11} = \frac{1}{4}\right\}$. This solution gives a probability of success $P_s = \frac{11}{12}$, and the throughput becomes:

$$T = N \cdot P_s = 2 \cdot \frac{11}{12} \approx 1.83 \quad \text{decoded users/frame}$$

which has increased by 9.6%, compared to the throughput that we obtained in case of one minislot in Sec. 9.1.

9.3 Generalization to a Sensing Phase with k Minislots

Let k be the number of minislots in the sensing phase. The number of possible minislot codewords in the sensing phase is $\omega = 2^k$ possible minislot codewords. We will compute the optimal probability to use each minislot codeword such that the probability to fully identify and then to decode both users is maximized.

Let us write the success probability for such a system:

$$P_s = 1 \cdot \left[1 - \left(\pi_0^2 + \pi_1^2 + \pi_2^2 + ... + \pi_{\omega-1}^2\right)\right] + \frac{2}{3} \cdot \sum_{i=0}^{\omega-1} \pi_i^2 \tag{15}$$

By writing the Lagrangian and solving the associated equations:

$$\begin{aligned}
&\mathcal{L} = 1 \cdot \left[1 - \left(\pi_0^2 + \pi_1^2 + \pi_2^2 + ... + \pi_{\omega-1}^2\right)\right] + \frac{2}{3} \cdot \sum_{i=0}^{\omega-1} \pi_i^2 - \lambda \left(\sum_{i=0}^{\omega-1} \pi_i - 1\right) \\
&\frac{\partial \mathcal{L}}{\partial \pi_0} = -2\pi_0 + \frac{4}{3}\pi_0 - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \pi_1} = -2\pi_1 + \frac{4}{3}\pi_1 - \lambda = 0 \\
&\dots \\
&\frac{\partial \mathcal{L}}{\partial \pi_{\omega-1}} = -2\pi_{\omega-1} + \frac{4}{3}\pi_{\omega-1} - \lambda = 0 \\
&\frac{\partial \mathcal{L}}{\partial \lambda} = \left(\sum_{i=0}^{\omega-1} \pi_i - 1\right) = 0
\end{aligned} \tag{16}$$

From the first ω equations, we deduce: $\pi_0 = \pi_1 = ... = \pi_{\omega-1} = \frac{-3\lambda}{2}$. Using the last equation (the $\omega + 1$ equation), we compute $\lambda = \frac{-2}{3\omega}$ and this gives:

$$\pi_0 = \pi_1 = \pi_2 = ... = \pi_{\omega-1} = \frac{1}{\omega}$$

The final success rate is given by :

$$P_s = 1 \cdot \left[1 - (\sum_{0}^{\omega-1} \frac{1}{\omega^2})\right] + \frac{2}{3} \sum_{0}^{\omega-1} \frac{1}{\omega^2} = 1 - \frac{1}{3} \sum_{0}^{\omega-1} \frac{1}{\omega^2} = 1 - \frac{1}{3\omega} = 1 - \frac{1}{3.2^k} \tag{17}$$

We deduce that adding a sensing phase before transmitting the packets in the scenario of IRSA with 2 users and 2 slots leads to a significant improvement in the probability to decode both users (the success probability), and the obtained throughput. The important question that we answer in the next section: can we extend the same analysis to a scenario of IRSA with M users and N slots?

10 Numerical Results

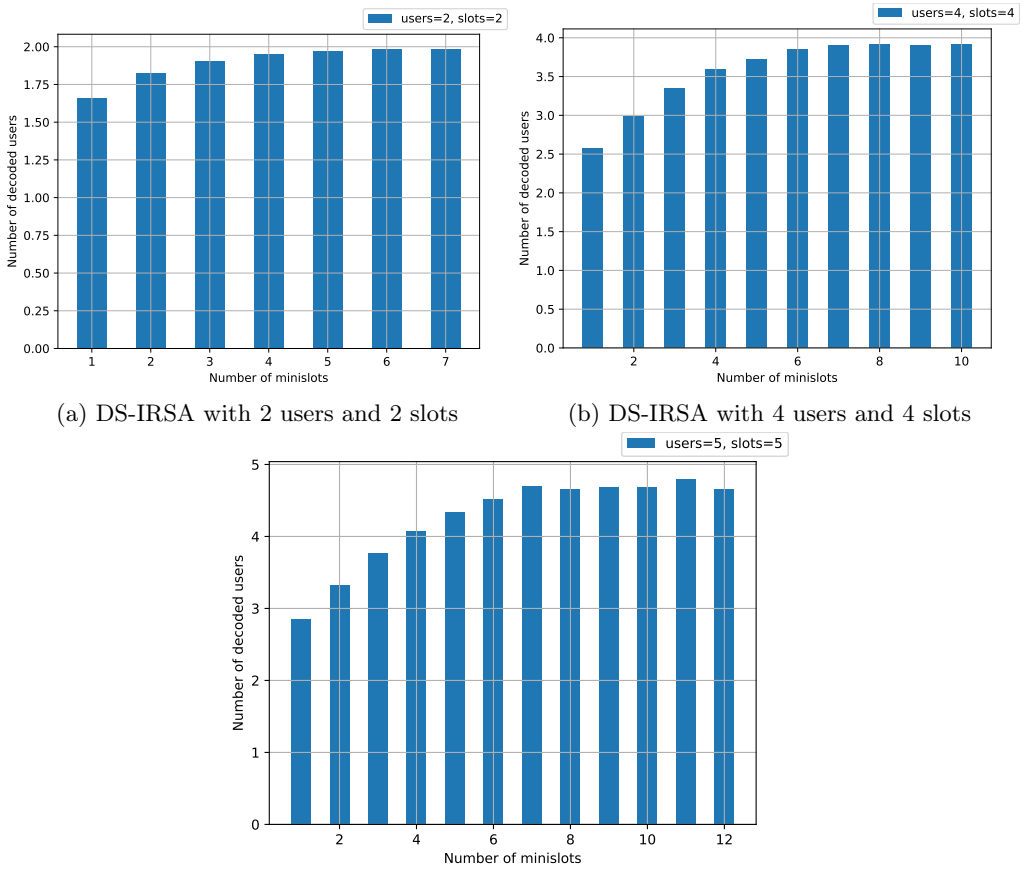
In this section, we present the obtained numerical results with DRL using our simulator. We developed our own IRSA simulator written in Python. The simulations allow constructing frames and generating user transmissions. IRSA iterative decoding is performed using a collision model after which the decoding information is obtained (decoded users, throughput, etc.). For DRL, we used OpenAI stable baselines. The neural network model implementing the policy (and the baseline value function estimate) is used with the default parameters: it contains 2 layers of 64 neurons amounting to approximately 4000 weights.

The arrival of users in the system is fixed to M users per frame. Training is done in steps. Each step accounts for an action taken by one user (agent). A full episode is completed when all the agents take their final actions, which is the slot selection for the transmission phase, after which the reward is calculated (the reward is zero at initialization and before the end of the frame and discounting factor is set to $\gamma = 1$).

In Table. 3, we show the obtained throughput after applying our DRL approach DS-IRSA. The two phases were trained simultaneously. Each value in the table represents a different scenario of DS-IRSA where we varied the number of users and the number of slots. We obtain

u/s	2	3	4	5	6
2	1.982	1.986	1.986	1.990	1.994
3	1.971	2.996	2.997	2.979	2.969
4	1.967	2.73	3.84	3.73	3.77
5	1.963	2.80	3.463	4.168	4.233
6	1.937	2.76	3.30	3.152	3.873

Table 3: The obtained throughput (expressed in terms of "average number of decoded users per frame") of DS-IRSA with 7 minislots using our DRL approach



(a) DS-IRSA with 2 users and 2 slots

(b) DS-IRSA with 4 users and 4 slots

(c) DS-IRSA with 5 users and 5 slots

Figure 15: The impact of increasing the number of minislots in the sensing phase on the obtained throughput of DS-IRSA

each value in Table. 3, by doing the average of 10 simultaneous simulations (training), after recording the number of decoding users at each simulation. All the presented scenarios in the table have 7 minislots in the sensing phase. We observe that DS-IRSA with 7 minislots is close to the maximum throughput of 2 recovered users in the case of 2 competing users (the first row of the table). DS-IRSA attains also almost the maximum throughput of 3 in the case of three users competing for ≥ 3 slots. We also note that adding a sensing phase of 7 minislots to IRSA has helped to increase the obtained throughput compared with the obtained values of throughput

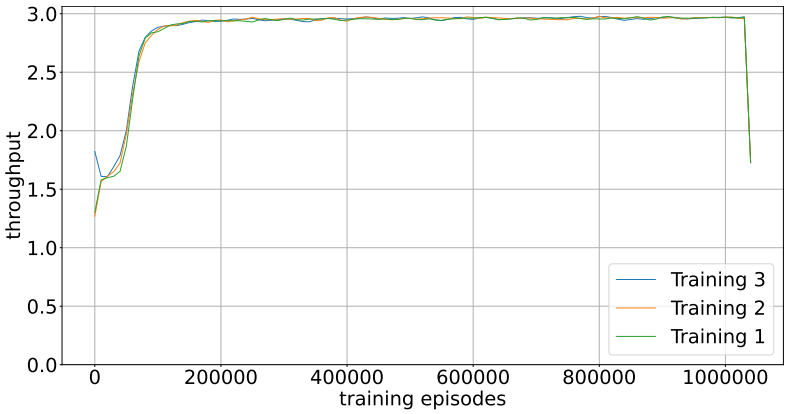
without sensing in Table 2 for all the studied scenarios. On the other hand, with sensing and when the number of users is ≥ 4, the throughput starts to converge slowly, and we do not recover all users. The reason is that the number of minislots that we used (7 minislots) is probably not sufficient to synchronize all users during the transmission phase, so more collisions occur, and not all users are recovered.

Fig. 15 shows the impact of increasing the number of minislots on the obtained throughput. Note that the number of episodes in the training phase for all the scenarios of Fig. 15 is one million for the Fig. 15a and 5 millions episodes for Fig. 15b and Fig. 15c. The figure shows that using more minislots in the sensing phase helps to increase the obtained throughput towards achieving the maximum performance of 1 [decoded user/slot]. In Section 9.3, it has been theoretically proven that increasing the number of minislots in the sensing phase increases the achieved throughput for DS-IRSA with 2 users, 2 slots, and k minislots which converges quickly towards the maximum of 2. The DRL approach achieves a throughput of 1.98 for DS-IRSA with 2 users, 2 slots and 7 minislots in the sensing phase (Fig. 15.a). When studying a scenario of DS-IRSA with strictly more than two slots and users, the DRL shows that using more minislots in the sensing phase will increase the achieved throughput, but this increase towards the optimal throughput is not always clearly observed as the convergence of the DRL approach is not guaranteed when the number of slots and the number of users increase. In this case, extra minislots are needed when the number of competing users increases, and this, in turn, increases the actions' space and can increase exponentially the complexity of the training.

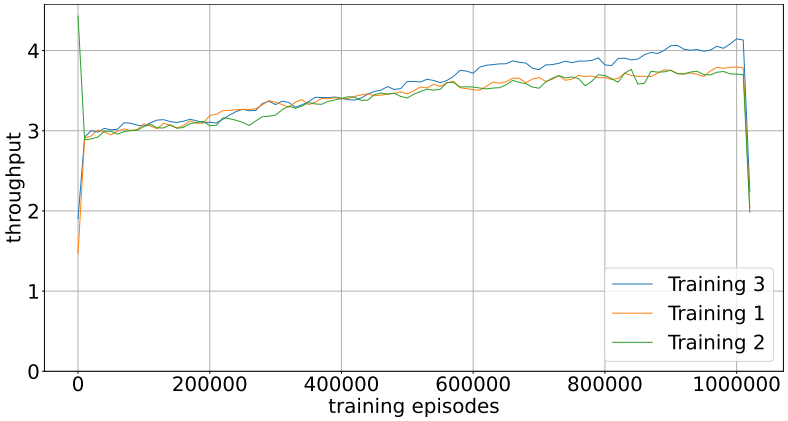
In Fig. 15.b, using 7 minislots in the sensing phase, DS-IRSA achieves a throughput of 3.84 (average number of decoded users per frame) where using 10 minislots is increasing the throughput up to 3.91. On the other hand, the obtained throughput for 5 users using 9 minislots, in Fig. 15.c, is 4.68, which is still far from the maximum throughput. As the number of users increase, the number of minislots needed to synchronize the users increases, which increases the training complexity in terms of time and convergence stability.

Fig. 16 shows the convergence of the learning process of DS-IRSA for different scenarios. For each scenario, we performed three different trainings with different seeds. Note that the training starts by randomly initialized values for the weights of the neural network model, and each value is different depending on the used seed. In Fig. 16a, the maximum throughput of 3 is quickly reached, as the number of minislots is sufficient to find an optimized transmission strategy for the 3 competing nodes. On the contrary, in Fig 16b and Fig 16c, the learning convergence towards the maximum throughput is slower as the number of competing users is larger in both cases. Note that in both figures, Fig 16b and Fig 16c, the number of minislots and the number of learning episodes were not sufficient to converge towards the maximum throughput of 6 and 7 users per slot respectively. In the last two examples, the performance with 7 users and slots leads to a lower absolute average number of decoded users par frame than with 6 users and slots, which is somewhat unexpected and surely due to convergence issues. Additionally, checking the number of minislots proportional to the number of slots and users, we also have the throughput of only ≈ 3.4 for 6 users, 6 slots, and 14 minislots, compared to Fig. 15a. Thus, experimentally, with the current implementation and training procedure, adding more minislots to the sensing phase for both scenarios will not guarantee convergence, as the problem of finding the optimal codebook becomes more complicated. In the last two scenarios (Fig 16b and Fig 16c), the learning curves seem to continue their increase slowly even at the end of the episodes. The continuation of the training process could help to increase the obtained throughput, but at the cost of a long learning time.

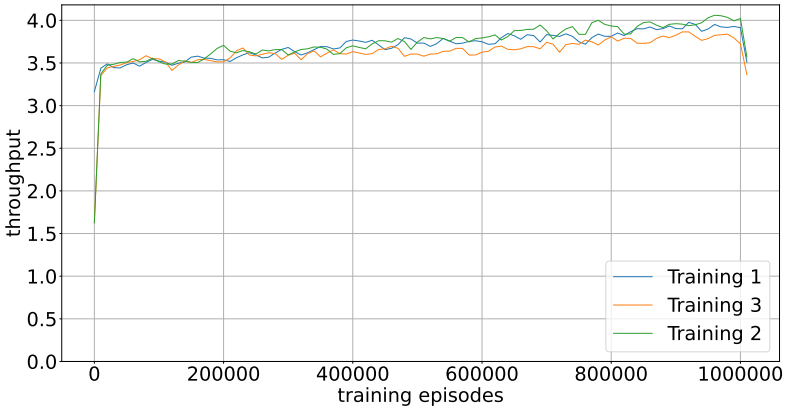
Finally, we represent the learned DS-IRSA protocol as a decision tree, following each observed state (that includes past selected actions and past minislot observations). After the training process is completed, we explore the actions selected by the trained model with DRL, depending



(a) DS-IRSA convergence with 3 users and 3 slots and 7 minislots



(b) DS-IRSA convergence with 6 users and 6 slots and 7 minislots



(c) DS-IRSA convergence with 7 users and 7 slots and 7 minislots

Figure 16: The learning convergence of DS-IRSA for different scenarios

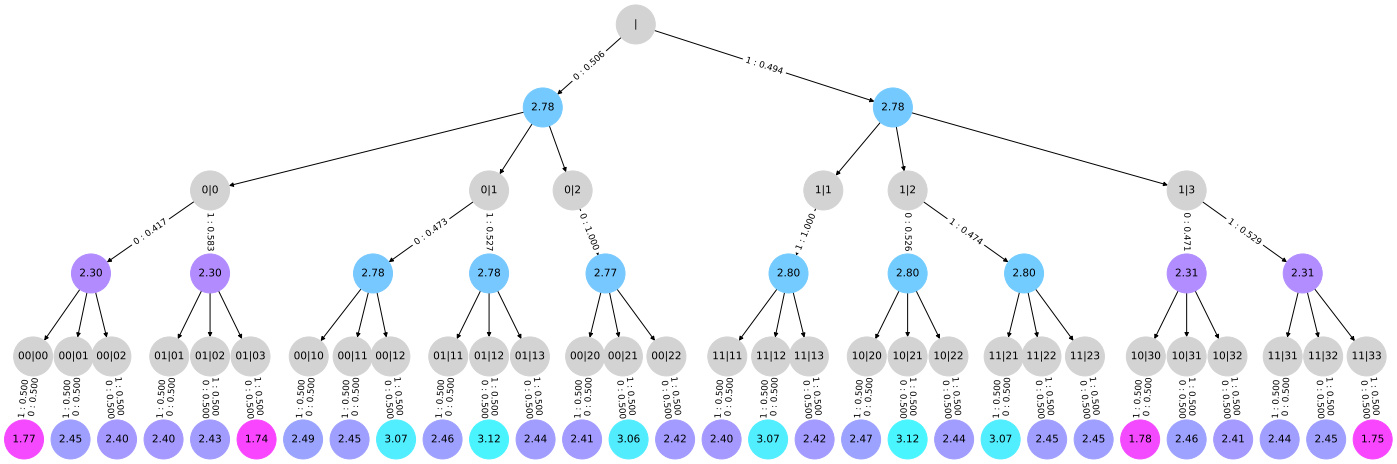
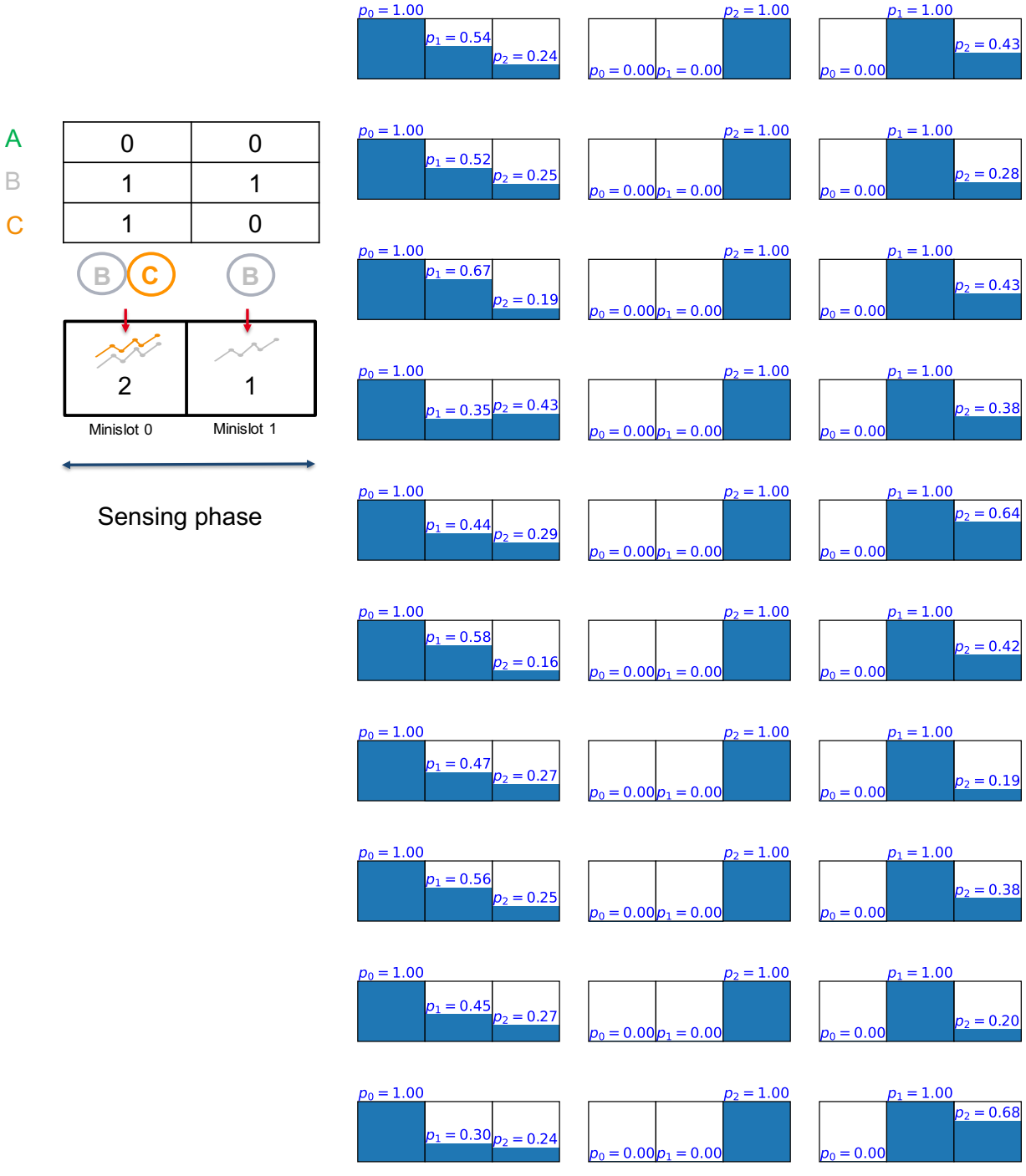


Figure 17: A tree representation of the learned DS-IRSA protocol with Deep Reinforcement Learning

on the state. Since there is a random vector as part of the input of the model, we perform an average of the outputs with different random vectors (each such trial is denoted a *simulation*). Fig. 17 shows a tree representation of the learned DS-IRSA protocol for 3 users competing for 3 slots and with 2 minislots in the sensing phase. The gray circles represent the observed state, which is the previously taken action(s) (on the left) and the observed energy on the corresponding minislots (on the right). For instance, "01|13", has two parts, "01" for actions and "13" for minislots. It means that the node has taken action 0 on the first minislot, followed by action 1 on the second minislot, while the observed energy at the end of the first minislot is 1, and the observed energy at the end of the second minislot is 3. The implementation of PPO that we are using, is associating a model that estimates the value function in addition to the model that provides the policy (the actions), with shared layers. The colored circles represent the estimate of the value function, and that is, an estimate of the expected cumulative returns value in the given state. In our case, this corresponds to the reward, which is the number of decoded users. The labels of the arrows between the circles represent the probability to take each action (to send 0 or 1 on the slot, e.g "0 : 0.506" and "1 : 0.486" at the top, indicate a probability to send a jamming burst with probability 0.486 and to stay silent with probability 0.506). The values in Fig. 17 are taken as an average of 100 simulations. These represented values on the tree indicate how the protocol behaves after it has been learned by the DRL approach.

In Fig. 18, we consider the same parameters as in Fig. 17, with 3 users, A, B and C, competing for 3 slots. The sensing phase has 2 minislots. The scenario considered is represented in Fig. 18a

After the protocol has been learned via the DRL approach, we also what happens after the sensing phase of this exact scenario: the codewords used in the transmission phase by the users. Since we are using PPO, and because of the way our actions are designed, the model will output the probability that each user takes action 1, i.e. transmit a replica, for each slot. Hence in the considered scenario, the output is a vector of 3 probabilities, as represented by Fig. 19. Also, because some source of entropy (random noise) is used as the input of the model, those output probability vectors can vary depending on this random noise. Fig. 18b represents these probability vectors for user A for 10 different simulations. Each simulation has been performed by using the same trained model, with the same entry state, but changing the random entropy vector (noise vector), as it is illustrated in Fig. 19. We can see from Fig. 18b that, user A, sends



(a) Sample scenario with 3 users A,B,C and 2 minislots

(b) The probability to send a replica on each of the three slots for user A

(c) The probability to send a replica on each of the three slots for user B

(d) The probability to send a replica on each of the three slots for user C

Figure 18: Scenario, and probabilities to send action a replica on each of the 3 slots for each of the three competing users A,B and C in the scenario

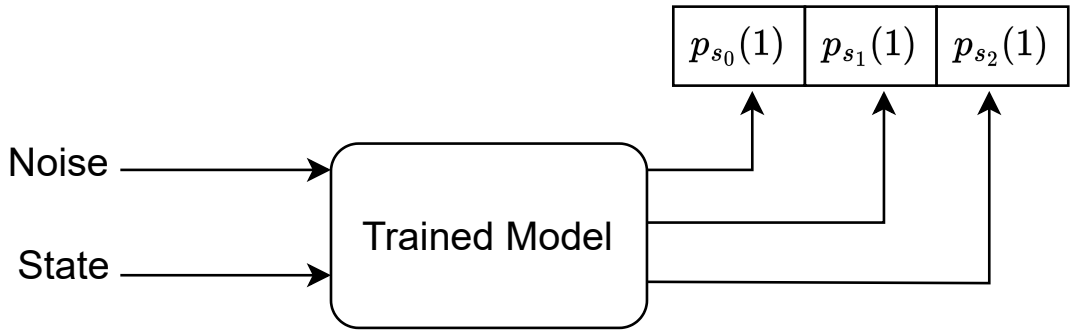


Figure 19: A tree representation of the learned DS-IRSA protocol with Deep Reinforcement Learning

a replica with a probability 1 on slot 0, while it sends a packet with a probability ≤ 1 on the slots 1 and 2. The other two users, user B and user C do not send packets on slot 0, i.e. $p_0 = 0$ (Fig. 18c and Fig. 18d). This can be interpreted as the neural network model reserving the slot 0 for user A. User B and user C send a packet with a probability 1 on slot 2 and slot 1 respectively, as it is shown in Fig. 18c and Fig. 18d. Note that the slot 1 is not "reserved" for user C, as user A can still send on the second slot with $p_0 \geq 0$. Note also that, slot 2 is not reserved for user B as both users A and C can send on this slot with $p_2 \geq 0$. Nevertheless, the packets of the three users can be still always be decoded due to the decoding process with SIC, so collisions can be resolved.

11 Conclusion

In this report, we aimed to optimize IRSA with small frame size scenarios. For that aim, we divided the work into two parts:

In the first part of this report, we first mathematically presented the problem of optimizing IRSA for small finite frame size and then, we optimized it through differential evolution. We applied Deep Reinforcement Learning techniques to solve this problem. The results have shown that our DRL approach, Deep RC-IRSA, attains the optimal values that we found through differential evolution. More generally, our work provides a generic method to optimize IRSA and its different variants, and importantly it is able to do that by selecting the slots instead of just selecting the degree, which shed the light on potentially a much better way of optimization such protocol. Our proposed method proves that it is a promising alternative to known methods in the literature (differential evolution, density evolution, etc.), especially for some more complicated variants of IRSA as IRSA with power diversity or IRSA with sensing. The main question that we propose at the end of this part is, whether adding a sensing phase before IRSA transmission could be useful and costly affordable to synchronize the nodes and avoid collisions. In this case, can our DRL approach exploit sensing information and reach more than the optimal values we found for classical short-frame IRSA? This will be addressed in the next part.

In the second part, we proposed a sensing protocol based on IRSA and trained with machine learning to synchronize the nodes during the transmission and avoid collisions. For that aim, we proposed DS-IRSA, Deep Learning Sensing-based IRSA protocol which is composed of two phases: a sensing phase, where the nodes can sense the channel and send short jamming signals, followed by a classical IRSA transmission phase. We use the DRL algorithm PPO and one of

its implementations (by OpenAI and others, "stable baselines" [26]) to optimize its performance. Our proposed protocol has shown an excellent performance to achieve an optimal performance of almost 1 [*decoded user/slot*] for small frame sizes (≤ 5) slots and with enough minislots. Our proposed protocol has also been shown to achieve higher throughput than classical IRSA protocol or other optimized IRSA variants through deep learning. The problem of optimizing DS-IRSA with our DRL approach becomes more complicated in terms of stability and learning time when the size of the frame increases, e.g. beyond 6 or 7 users and slots for a million training episodes. Future work could be to optimize the learning approach to reduce the learning complexity for larger frame sizes.

References

[1] F. Clazzer, A. Munari, G. Liva, F. Lazaro, C. Stefanović, and P. Popovski, "From 5G to 6G: Has the Time for Modern Random Access Come?" 03 2019.

[2] G. Liva, "Graph-Based Analysis and Optimization of Contention Resolution Diversity Slotted ALOHA," *IEEE Transactions on Communications*, vol. 59, no. 2, pp. 477–487, February 2011.

[3] M. C. Moroğlu, H. M. Gürsu, F. Clazzer, and W. Kellerer, "Short Frame Length Approximation for IRSA," *IEEE Wireless Communications Letters*, vol. 9, no. 11, pp. 1933–1936, 2020.

[4] H. Al-Mefleh and O. Al-Kofahi, "Taking Advantage of Jamming in Wireless Networks: A Survey," *Computer Networks*, vol. 99, pp. 99–124, 2016.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[6] G. Liva, "Graph-Based Analysis and Optimization of Contention Resolution Diversity Slotted ALOHA," *IEEE Transactions on Communications*, vol. 59, no. 2, pp. 477–487, 2011.

[7] I. Ayoub, I. Hmedoush, C. Adjih, K. Khawam, and S. Lahoud, "Deep-IRSA: A Deep Reinforcement Learning Approach to Irregular Repetition Slotted ALOHA," in *10th IFIP International Conference on Performance Evaluation and Modeling in Wireless and Wired Networks (PEMWN)*, 2021, pp. 1–6.

[8] I. Hmedoush, "Connectionless Transmission in Wireless Networks (IoT)," PhD Thesis, Sorbonne University, EDITE, and Inria Paris, May 2022. [Online]. Available: https://hal.inria.fr/tel-03718327

[9] W. Kautz and R. Singleton, "Nonrandom Binary Superimposed Codes," *IEEE Transactions on Information Theory*, vol. 10, no. 4, pp. 363–377, 1964.

[10] H. A. Inan, S. Ahn, P. Kairouz, and A. Ozgur, "A Group Testing Approach to Random Access for Short-Packet Communication," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 96–100.

[11] H. Zhu and G. B. Giannakis, "Exploiting Sparse User Activity in Multiuser Detection," *IEEE Transactions on Communications*, vol. 59, no. 2, pp. 454–465, 2011.

[12] C. R. Srivatsa and C. R. Murthy, "User Activity Detection for Irregular Repetition Slotted Aloha Based MMTC," *IEEE Transactions on Signal Processing*, vol. 70, pp. 3616–3631, 2022.

[13] J. Massey and P. Mathys, "The Collision Channel Without Feedback," *IEEE Transactions on Information Theory*, vol. 31, no. 2, pp. 192–204, 1985.

[14] E. Paolini, G. Liva, and A. G. i Amat, "A Structured Irregular Repetition Slotted ALOHA Scheme with Low Error Floors," in *IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[15] C. Boyd, R. Vehkalahti, O. Tirkkonen, and A. Laaksonen, "Code Design Principles for Ultra-Reliable Random Access with Preassigned Patterns," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2604–2608.

[16] D. Duchemin, J.-M. Gorce, and C. Goursaud, "Code Domain Non Orthogonal Multiple Access versus ALOHA: A simulation based study," in *25th International Conference on Telecommunications (ICT)*, 2018, pp. 445–450.

[17] Y. Zhang, Y. Chen, Y.-H. Lo, and W. S. Wong, "The Zero-Error Capacity of a Collision Channel With Successive Interference Cancellation," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 1653–1657.

[18] H. Yu, Y. Kang, Z. Shi, Y. Shao, Y. Lin, and Y. Zhang, "Design of Deterministic Grant-Free Access with Deep Reinforcement Learning," in *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, 2020, pp. 944–948.

[19] A. Destounis, D. Tsilimantos, M. Debbah, and G. S. Paschos, "Learn2MAC: Online Learning Multiple Access for URLLC Applications," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 1–6.

[20] B.-M. Robaglia, A. Destounis, M. Coupechoux, and D. Tsilimantos, "Deep reinforcement learning for scheduling uplink iot traffic with strict deadlines," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[21] B. Verweij, S. Ahmed, A. J. Kleywegt, G. Nemhauser, and A. Shapiro, "The Sample Average Approximation Method Applied to Stochastic Routing Problems: a Computational Study," *Computational optimization and applications*, vol. 24, no. 2, pp. 289–333, 2003.

[22] D. Bertsekas and R. Gallager, "Data Networks," *(2nd edition), Prentice Hall*, 1992.

[23] E. Pesce and G. Montana, "Improving Coordination in Small-Scale Multi-Agent Deep Reinforcement Learning Through Memory-Driven Communication," *Machine Learning*, pp. 1–21, 2020.

[24] M. L. Molle and G. C. Polyzos, "Conflict Resolution Algorithms and Their Performance Analysis," *University of Toronto, CS93-300, Tech. Rep*, 1993.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, 2nd ed., ser. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2018.

[26] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable Baselines," https://github.com/hill-a/stable-baselines, 2018.

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399