MIMO simulation in 5G networks: Py5cheSim and DeepMIMO integration

Diego Sánchez
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
diego.sanchez.donadini@fing.edu.uy

Mateo Trujillo
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
mateo.trujillo@fing.edu.uy

Paula Varela
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
paula.varela@fing.edu.uy

Claudina Rattaro
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
crattaro@fing.edu.uy

Lucas Inglés
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
lucasi@fing.edu.uy

Pablo Belzarena
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
belza@fing.edu.uy

***Abstract*—This article describes the integration of two software tools: Py5cheSim and DeepMIMO. The former is a Python-based 5G mobile network simulator specifically designed for resource allocation algorithm evaluation. It stands out as one of the few open-source simulators capable of emulating network slicing at the radio resource level, allowing the assessment of resource allocation algorithms at both slice and user levels within them. On the other hand, DeepMIMO is a data generator for mmWave/massive MIMO channels. The main contribution of this work is the integration of both systems, allowing the simulation of realistic scenarios in 5G networks. As a result of this integration, a new version of Py5cheSim was obtained. Leveraging the advantages of Py5cheSim's implementation for new resource allocation algorithms, a new MIMO-based algorithm was incorporated into the tool (functionality that was only rudimentarily supported in Py5cheSim v1.0). It is worth noting that this new development is fully compatible with the previous version of the base network simulator, ensuring easy adoption of the new features by the community. The obtained results demonstrate a significant improvement in simulation accuracy and a greater capacity to represent the challenges of 5G networks in real scenarios.**

***Index Terms*—5G mobile networks, simulations, massive MIMO, network slicing, resource allocation.**

***Resumen*—El presente artículo describe la integración de dos herramientas de software: Py5cheSim y DeepMIMO. El primero es un simulador de redes móviles 5G escrito en Python que está especialmente diseñado para la evaluación de algoritmos de asignación de recursos. Es uno de los pocos simuladores de código abierto que permite emular *network slicing* a nivel de recursos radio, y por lo tanto, permite evaluar algoritmos de asignación de recursos a dos niveles: *slices* y usuarios dentro de ellas. Por otro lado, DeepMIMO es un generador de datos para canales de banda milimétrica con soporte para MIMO masivo. La contribución principal de este trabajo es la integración de ambos sistemas permitiendo simular escenarios realistas en redes 5G. De esta integración se obtuvo como resultado una nueva versión del simulador. Explotando las ventajas en la implementación de nuevos algoritmos de asignación de recursos que provee Py5cheSim, se incorporó a la herramienta un nuevo algoritmo basado en MIMO (funcionalidad que estaba soportada de manera muy básica en Py5cheSim v1.0). Es importante destacar que el nuevo desarrollo es completamente compatible con la versión anterior del simulador base de redes, lo cual garantiza una fácil adopción de las nuevas características por parte de la comunidad. Los resultados obtenidos muestran una mejora significativa en la precisión de las simulaciones y una mayor capacidad para representar los desafíos de las redes 5G en escenarios reales.**

***Index Terms*—redes móviles 5G, simulaciones, MIMO masivo, network slicing, asignación de recursos.**

Este trabajo contó con el apoyo de la Agencia Nacional de Investigación e Innovación (ANII), Uruguay (FMV_1_2019_1_155700, Inteligencia Artificial aplicada a Redes 5G) y de la Dirección Nacional de Innovación, Ciencia y Tecnología, Ministerio de Educación y Cultura, Uruguay – Fondo Vaz Ferreira (FVF-2021-128–DICYT, Herramientas de simulación de redes móviles de futura generación).

I. Introducción

En la última década ha surgido la tecnología móvil de quinta generación (5G) con el objetivo de satisfacer las demandas actuales de comunicaciones móviles. Las recomendaciones IMT-2020 de ITU-R WP5D [1] definen tres escenarios de uso para abordar el creciente uso de la red celular, tomando en cuenta las opiniones de diferentes organizaciones regionales y de operadores de la industria móvil. Estos son: servicios de banda ancha mejorada eMBB (*Enhanced Mobile Broadband*), servicios que requieren una latencia muy baja y alta confiabilidad URLLC (*Ultra-reliable and low-latency communications*) y servicios mMTC (*Massive Machine Type Communications*). Para cumplir con estos requisitos, 5G NR (*New Radio*) propone novedades, tales como operabilidad en bandas de ondas milimétricas, numerología variable, MIMO (*Multiple Input Multiple Output*) masivo y *network slicing*. Las tres primeras son técnicas empleadas para satisfacer la demanda de tasas de datos extremas y la alta demanda en áreas con grandes densidades de conexiones. Por otra parte, el concepto de *network slicing* permite la configuración personalizada de diferentes *slices* para satisfacer los diversos casos de uso contemplados por la nueva tecnología; siendo la gran

novedad de 5G llevar este concepto hasta la red de acceso incluidos los recursos radio [2].

Estos servicios y novedades han dado lugar a la exploración de nuevas áreas de trabajo e investigación, tales como la asignación de recursos realizada por las radiobases (*scheduling*, tanto *interslice* como *intraslice*) y la simulación en escenarios realistas de redes 5G. En este contexto surge Py5cheSim como simulador para redes móviles de nueva generación [3], [4]. Py5cheSim ha constituído un aporte significativo en esta área, siendo uno de los pocos simuladores de acceso gratuito y con una interfaz amigable con el usuario que está orientado al análisis de algoritmos de asignación de recursos (aspecto que, como es sabido, no está totalmente determinado en el estándar). De este modo, ha posibilitado el desarrollo de nuevas investigaciones como por ejemplo [5], [6]. Como se mencionó anteriormente, las redes 5G tienen la posibilidad de operar en bandas de ondas milimétricas y por lo tanto se espera que exploten la posibilidad de MIMO masivo y *beamforming*. Para lograr simulaciones confiables y comparables en estos contextos es indispensable contar con una implementación fiel del modelo de canal. Es por ello que como contribución principal de este trabajo se realizó la integración de este simulador y el *framework* DeepMIMO [7] resultando en una nueva versión de Py5cheSim.

El resto del artículo se estructura de la siguiente forma. En la sección II se presentan las dos herramientas principales de este trabajo: Py5cheSim y DeepMIMO. En la sección III se presentan tres aspectos: la integración de ambos sistemas, las modificaciones en la versión original del simulador base Py5cheSim y la implementación de un algoritmo de asignación de recursos basado en MIMO que además de validar la integración realizada, representa un aporte en sí mismo a la herramienta base. En la sección IV se repasan algunos escenarios de validación y finalmente se concluye el trabajo en la sección V.

II. Herramientas utilizadas

A continuación se presentan las características principales de Py5cheSim y DeepMIMO. Para mayores detalles se sugiere consultar las referencias citadas de cada herramienta.

A. Py5cheSim

Es un simulador para redes 5G de código abierto que sirve como base para el desarrollo e implementación de nuevos algoritmos de asignación de recursos. En particular en lo presentado en este artículo, se trabajó sobre la versión 1.0 del simulador[1].

Más en detalle, Py5cheSim utiliza la librería Simpy para simular el envío y recepción de paquetes y generar métricas de rendimiento a nivel de celda sobre *schedulers* implementados por el usuario en una configuración establecida. Se enfoca en la asignación de recursos entre diferentes usuarios y servicios, con diferentes requerimientos. El simulador posibilita tanto la configuración (y/o desarrollo) de algoritmos *interslice* como *intraslice schedulers* para la implementación de RAN (*Radio Access Network*) Slicing.

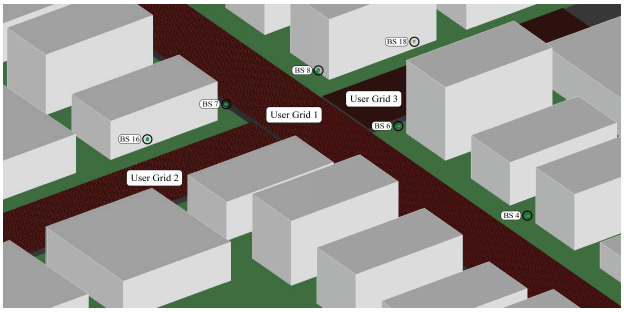

Figura 1. Escenario *Outdoor* 1. Ese escenario permite trabajar en las bandas 3.4 GHz - 3.5 GHz - 28 GHz - 60 GHz. Permite simular 1,000,000 de usuarios y 18 radios base. Consultar más detalles en DeepMIMO wep page.

A pesar de contar con muchas ventajas gracias a su nivel de abstracción y su simplicidad en el uso e implementación de nuevos *schedulers*; a diferencia de otros simuladores usados a nivel académico como [8], [15], [9] y [10], Py5cheSim no cuenta con un modelo de canal realista y por ende la simulación de un canal MIMO en su versión original está muy simplificada. Más en detalle, en la versión 1.0 el canal inalámbrico entre los usuarios y las radiosbase se define con un valor de SINR (*Signal to Interference & Noise Ratio*) aleatorio (o manualmente configurado) entre 0 dB y 40 dB, y no se realiza una distinción a nivel de PRB (*Phisical Resource Block*) sobre la calidad del canal, sino que se utiliza un único valor de SINR para toda la banda. Con estas limitaciones en mente es que se optó por integrar Py5cheSim a un dataset que permitiera obtener un modelo de canal realista tanto en escenarios indoor así como outdoors.

B. DeepMIMO

DeepMIMO es un generador de datos para canales de banda milimétrica con soporte para MIMO masivo. Es posible ejecutar esta herramienta tanto en MatLab como en Python. Los datos que brinda este *framework* quedan determinados dado el conjunto de parámetros a setear y el escenario de *ray-tracing*. Más específicamente, los canales de DeepMIMO están construidos a través de datos de *ray-tracing* obtenidos de un simulador pago (Wireless InSite [11] desarrollado por la empresa Remcom). Estos escenarios de *ray-tracing* se encuentran disponibles de manera gratuita en la página de DeepMIMO[2]. En la Figura 1 se observa la vista tridimensional de uno de ellos. Una limitación de los escenarios que se disponen con esta herramienta es que, en todos ellos, los usuarios simulados son estáticos.

Este *framework* continúa su desarrollo incorporando nuevos escenarios. Actualmente soporta cuatro escenarios *outdoors* (uno de drones), tres *indoors* y uno basado en la ciudad de Boston.

Con la integración entre DeepMIMO y Py5cheSim, en este artículo se presentan las características de una nueva versión de

[1]Py5cheSim v1.0 github

[2]DeepMIMO página web

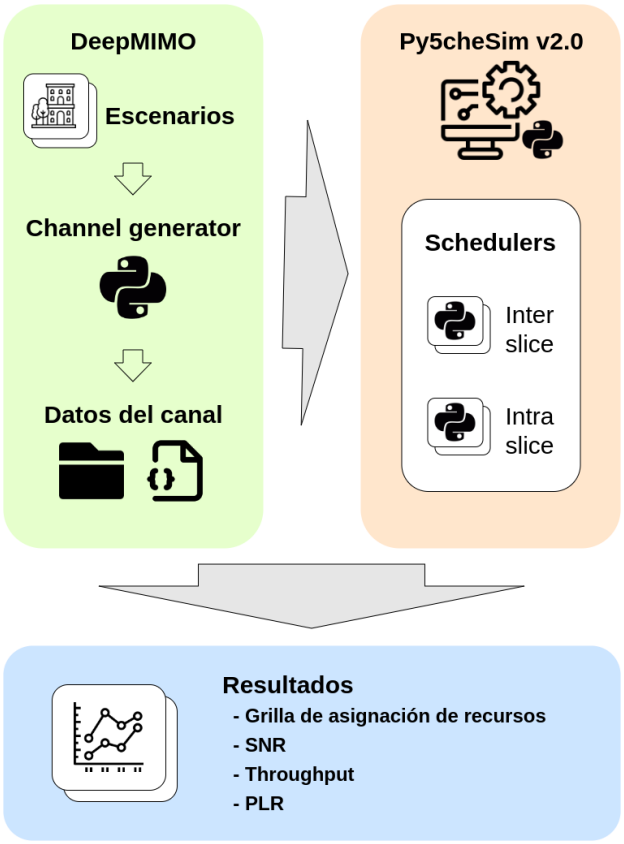

Figura 2. Diagrama del funcionamiento general del sistema.

este último. Allí, los escenarios son más realistas y permiten a los usuarios probar algoritmos que antes no eran compatibles o estaban implementados en una forma muy simplificada.

III. Arquitectura general del sistema

Los nuevos desarrollos presentados en este artículo se encuentran disponibles en el repositorio [12]. Por una documentación más detallada se sugiere consultar [13]. A lo largo del documento, cuando se hace referencia a un *script*, módulo o parte del código se sugiere revisar el repositorio mencionado que cuenta con un ejemplo de uso detallado.

El flujo de una simulación en la nueva versión de Py5cheSim se puede ver en la Figura 2 y se compone de dos etapas. La primera de ellas consiste en obtener el estado del canal inalámbrico entre radiobases y usuarios, el cual se consigue desde un escenario de DeepMIMO ejecutando el *script channel_generator.py*. Allí se deben definir las características generales del escenario y de los usuarios. Como parámetros importantes, del escenario se debe especificar la frecuencia central y el ancho de banda disponible; y de los usuarios, su velocidad y posición inicial sobre la grilla de usuarios definida en el escenario de DeepMIMO.

La segunda etapa consiste en definir en el *script simulation_v2.py* los grupos de usuarios que se van a utilizar en la simulación, compuestos principalmente de un perfil de tráfico y de requerimientos en cuanto a *delay*. Además, se debe desarrollar o seleccionar un *interslice scheduler* que será utilizado para organizar los recursos de todas las *slices* de la celda, así como también se deberá definir un *intraslice scheduler* por grupo de usuarios a simular.

El resultado de la ejecución de la segunda etapa devuelve información sobre el rendimiento de la asignación de recursos realizada mediante los *schedulers* utilizados. Esta información es presentada a través de gráficas en función del tiempo de valores de *throughput*, SNR (*Signal to noise ratio*), tasa de pérdida de paquetes, usuarios conectados, MCS (*Modulation Coding Scheme*), paquetes en *buffer* y uso de recursos físicos. En la nueva versión de Py5cheSim se devuelve también la grilla de asignación de recursos, la cual muestra para cada PRB cuáles fueron los usuarios servidos por ella en cada TTI (*Transmission Time Interval*). Esto facilita la visualización del multiplexado espacial en el caso de utilizar MU-MIMO.

Para lograr la integración y adaptación de Py5cheSim para que funcione según lo explicado anteriormente, se llevaron a cabo desarrollos que se pueden agrupar en tres categorías bien definidas e independientes entre sí. Estas son: obtención de datos del canal inalámbrico, simulación de comunicaciones MIMO y asignación de recursos. A continuación se detallan cada una de ellas, que juntas conforman la nueva versión del simulador disponible en el repositorio [12].

A. Integración con DeepMIMO

La integración del *framework* DeepMIMO se realizó a través del *script channel_generator.py* mencionado, en el cual se deben especificar las características de la configuración de radio, por ejemplo, el número y tipo de antenas con las que cuenta cada usuario, y de los grupos de usuarios a considerar, que se traducirán en *slices* en Py5cheSim. Un punto importante a resolver es que esta integración no es directa cuando se desea simular ambientes realistas, es decir, aquellos en los que los usuarios son móviles.

Recordando que en DeepMIMO los usuarios simulados son estáticos, pero todos los escenarios disponibles permiten implementar una gran cantidad de éstos, para simular el movimiento de un usuario en Py5cheSim (en adelante denominado usuario dinámico) se optó por tomar muestras de los datos del canal de usuarios DeepMIMO diferentes en las distintas escenas que componen la simulación. Particularmente, los movimientos disponibles para un usuario son verticales u horizontales. Por lo tanto para usuarios dinámicos los datos del canal que se toman corresponden con los datos de los usuarios DeepMIMO que se encuentran en la vertical o la horizontal en que se encuentra el usuario dinámico considerado en Py5cheSim (ver ejemplificación el la Figura 3). Más específicamente, en el *script channel_generator.py* a los grupos de usuarios se les debe especificar el tipo de movimiento, la velocidad y la posición inicial sobre la grilla de usuarios de DeepMIMO. En base a esta información se generan varias escenas distintas cada cierto tiempo especificado inicialmente, en cada una de ellas los usuarios de Py5cheSim (usuarios de los que se quieren obtener la métricas y demás) se posicionan

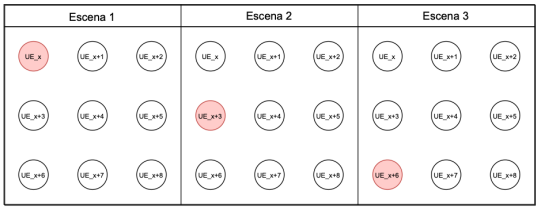

Figura 3. Esquema de toma de datos para un usuario dinámico de Py5cheSim que se mueve verticalmente (rojo), en una grilla de 9 usuarios DeepMIMO. Se observa como el usuario considerado en Py5cheSim va tomando el lugar de uno distinto de los de la grilla DeepMIMO en cada escena.

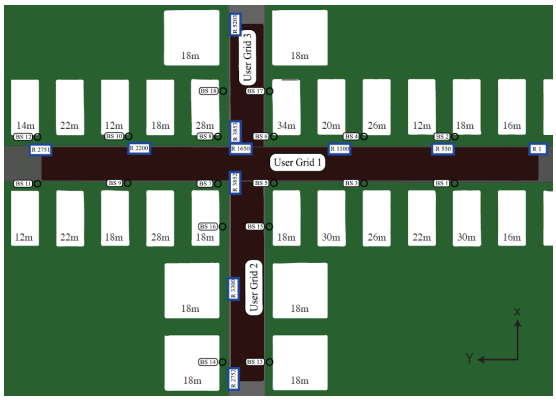

Figura 4. Escenario *O1* de DeepMIMO, el cual cuenta con tres grillas de usuarios distintas.

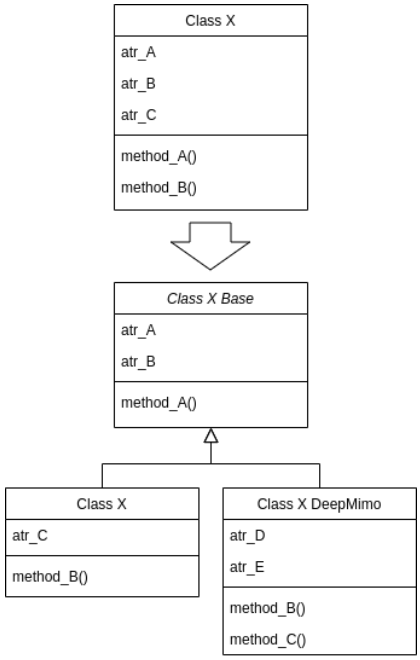

Figura 5. Diagrama de modificación para cada una de las clases existentes. Una clase *Class X* en la versión 1 de Py5cheSim (arriba) pasa a ser implementada en tres clases distintas (abajo): *Class X Base*, *Class X* y *Class X DeepMIMO*.

sobre distintos usuarios de la grilla del escenario de DeepMIMO (ver Figura 4).

Para cada escena, se genera un archivo por grupo de usuarios (por *slice*) en el cual se almacena el estado del canal en cada PRB definido, así como la dirección del rayo principal correspondiente a cada usuario. Este archivo será el utilizado en la nueva versión de Py5cheSim según lo explicado en la sección siguiente en la Figura 6. Este nuevo estado del canal (recordar que en la versión inicial consistía en un valor aleatorio de SINR) está conformado por el cálculo del SNR y la cantidad de *layers* máximas que soporta la comunicación en dicho PRB, además del reporte del rayo principal que es muy útil como una aproximación para determinar si dos usuarios distintos pueden compartir o no un PRB al implementar MU-MIMO.

B. Modificaciones realizadas a Py5cheSim 1.0

En la sección anterior fue descrita la salida de la primera etapa de la simulación. En virtud de integrar esta característica al simulador y de lograr la compatibilidad entre la nueva versión y su antecesora, se optó por modificar las clases existentes de la forma descrita en la Figura 5. La misma es posible debido al que simulador está desarrollado bajo el paradigma de programación orientada a objetos.

Este esquema plantea colocar los métodos y atributos comunes a ambas versiones en una clase base *Class X Base*. Las funcionalidades particulares de la primera versión se implementan en la clase *Class X*, la cual hereda de la clase base y mantiene el nombre original de la misma para ser compatible con *scripts* de la primera versión. Luego, las funcionalidades de la segunda versión de Py5cheSim (presentada en este artículo) son implementadas mediante métodos y atributos en la clase *Class X DeepMIMO*, la cual también hereda de la clase *Class X Base*. De esta manera se logran implementar las nuevas funcionalidades sin repetir código y manteniendo la compatibilidad hacia atrás.

Con el esquema presentado anteriormente se ha implementado la actualización del estado del canal. Esto ha implicado modificar el modelo del canal de los usuarios, con el fin de que puedan almacenar los nuevos datos de relación señal-ruido (SNR) y rango definidos sobre los distintos PRBs, así como el rayo principal que es común a toda la portadora OFDM. Además, estos valores son actualizados periódicamente en cada escena. Ver el diagrama de secuencia de la actualización del estado de canal en la Figura 6.

Bajo ese mismo paradigma, se ha extendido la funcionalidad de *network slicing*. Esta, al igual que en la primera versión del simulador, se implementa mediante dos tipos de *schedulers* diferentes: un *interslice scheduler* y varios *intraslice schedulers*. Sin embargo, estos últimos debieron ser modificados, ya que ahora se asignan PRBs específicos (no es lo mismo un PRB que otro pues se tiene información del estado de canal para cada uno de ellos). En el caso del *interslice scheduler*, se implementó una versión muy básica que distribuye de forma equitativa los PRBs base entre las distintas *slices*. En cuanto al *intraslice scheduler*, se implementó una versión igualmente básica, la cual distribuye los PRBs asignados a la *slice* entre

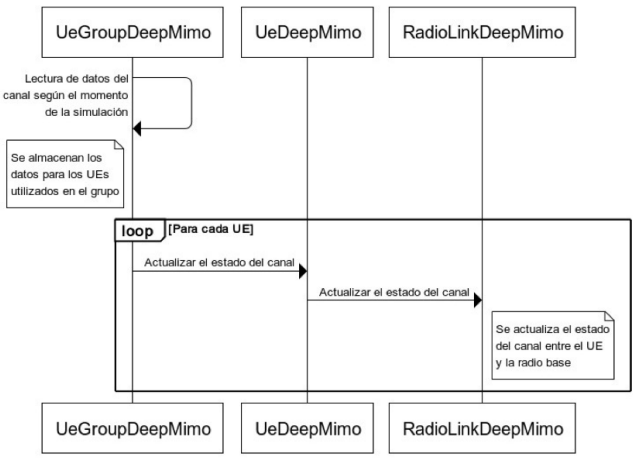

Figura 6. Diagrama de secuencia: proceso de actualización del estado del canal de Py5cheSim v2.0.

los distintos usuarios con paquetes pendientes por enviar. En la siguiente sección, se presentará una implementación más compleja que toma en cuenta el estado del canal de los usuarios de la *slice* sobre los PRBs asignados a ella.

Es importante mencionar que otra funcionalidad que se extendió con el mismo método fue el despacho de paquetes de usuarios. En esta nueva versión del simulador, esta funcionalidad debe tener en cuenta el estado del canal sobre los PRBs asignados al usuario para obtener la Modulación y Codificación Específica (MCS) y, de esa manera, determinar la cantidad de RB necesarios.

Por último, se implementó el soporte para MIMO masivo mediante la asignación de recursos a los usuarios. Cuando se actualiza el canal de un usuario, se especifica la cantidad máxima de *layers* que soporta la comunicación con la radio base para cada PRB. Luego, es tarea del *intraslice scheduler* determinar si se utilizará SU-MIMO o MU-MIMO y con cuántas *layers*.

C. Implementación de scheduler basado en MIMO

Debido a que en la segunda versión del simulador se introduce como nueva característica el modelado de canal más detallado, el cual brinda información a los *intraslice schedulers* sobre la posibilidad de que los usuarios realicen o no MU-MIMO en un mismo PRB, se decidió implementar un *intraslice scheduler* basado en el algoritmo NUM (la implementación se basó en [14]). Esta implementación servirá como referencia para el desarrollo de futuros *schedulers* en esta versión de Py5cheSim.

El algoritmo elegido para implementar permite asignar PRBs a uno o más usuarios cuando estos utilizan MU-MIMO, y lo logra formando todos los grupos posibles de usuarios que pueden compartir un PRB según el criterio del rayo principal. Luego, se calcula una métrica para cada uno de ellos y se le asigna el PRB en cuestión al grupo que obtuvo la mejor métrica. Esta métrica la denotamos como $r_i(l)$ y se calcula de la siguiente manera:

$$r_i(l) = \sum_{i \in M_j} c_i(k,l) U^{'}(\overline{r_i(l)}), \tag{1}$$

donde M_j es el grupo de usuarios j dentro del conjunto $\mathcal{M}$ del total de usuarios, conjunto de posibles grupos de usuarios posibles que pueden compartir el PRB número k. El parámetro $c_i(k,l)$ es el *throughput* del usuario i en el PRB k e instante l y tiene la siguiente expresión:

$$c_i(k,l) = \sum_{s=1}^{\eta_i} N_{PRB}^{RE} \log_2(1 + \beta \rho_{i,s}^{\mathcal{M}_j}(k,l)). \tag{2}$$

En esta última ecuación, N_{PRB}^{RE} es la cantidad de símbolos OFDM por portadora, β depende del BER que se desea lograr según $\beta = \frac{1,5}{-\log(5BER)}$, $\rho_{i,s}^{Mj}(k,l)$ es el SNR que lo tomamos directamente de los archivos generados en la primera etapa en PRB número k e instante l, y por último, η_i es la cantidad de *layers* del usuario i.

El valor r_i se denomina *throughput* total, e indica la suma de los *throughputs* en los grupos que el usuario i pertenece. Este valor se puede expresar mediante la siguiente expresión:

$$r_i = \sum_{j=1}^{\mathcal{M}_c} \sum_{k \in D_j} c_i(j), \tag{3}$$

donde j indica los grupos de usuarios que i forma parte, M_c la cantidad de grupos total y D_j es el conjunto de PRBs asignado al grupo de usuarios j. En este caso, se utiliza el promedio del *throughput* total. Una manera de calcular este promedio es utilizar el método *moving exponential average filter* [14]:

$$\overline{r_i}(l) = \delta \overline{r_i}(l-1) + (1-\delta) r_i(l-1). \tag{4}$$

Por último, $U(x)$ es una función de utilidad a elección del usuario del algoritmo. Una comúnmente utilizada es $\log(x)$. En definitiva, para asignar un PRB a un grupo de usuarios en el instante l, el algoritmo busca maximizar la métrica $r_i(l)$ dada por la Ecuación (3).

Para completar esta sección se menciona que adicionalmente a la asignación de PRB, se realizó un mecanismo para la asiganción de la cantidad de *layers* a los usuarios. Si a un conjunto de usuarios se le asigna un grupo de PRBs, la cantidad de *layers* de éstos será la misma y será igual al menor de los rangos reportados por éstos. De esta manera se asegura que un usuario no reciba más *layers* de las que puede recibir.

Este algoritmo fue implementado y se encuentra disponible en la versión 2.0 de Py5cheSim. Esta implementación constituye un valor agregado a la nueva versión del simulador.

IV. Validación

Las validaciones se llevaron a cabo para las funcionalidades introducidas previamente. Las validaciones sobre el canal inalámbrico y la integración a Py5cheSim quedan implícitamente incluidas en la presente validación (para mayores detalles, consultar [13]). En este artículo presentaremos la verificación de la performance del algoritmo de asignación de PRB y *layers* presentado en la sección próxima anterior.

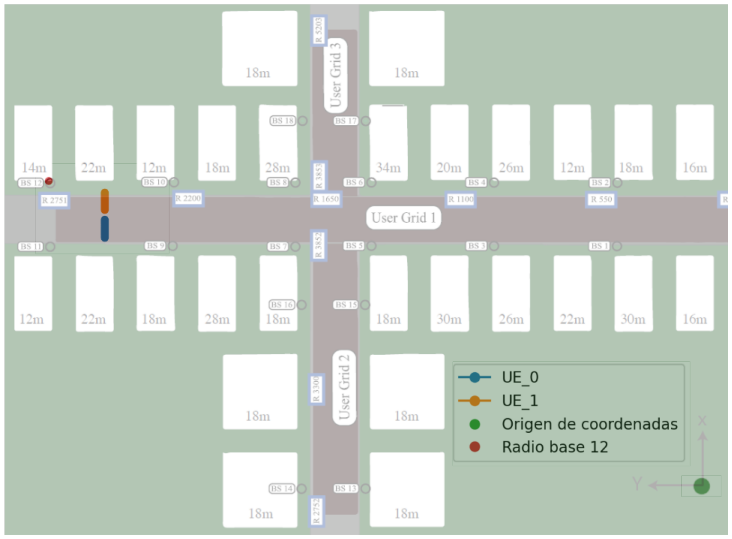

Figura 7. Recorrido de ambos usuarios sobre el escenario *Outdoor 1* del *framework* DeepMIMO. Comienzan en lados opuestos de la calle y avanzan simultáneamente hasta casi encontrarse.

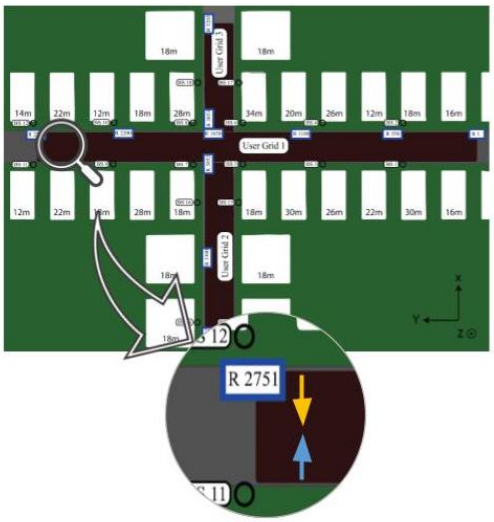

Figura 8. Recorrido de ambos usuarios sobre el escenario *Outdoor 1* del *framework* DeepMIMO. Comienzan en lados opuestos de la calle y avanzan simultáneamente hasta casi encontrarse. Zoom de Figura 7.

El escenario utilizado en las simulaciones fue el denominado *Outdoor 1* del *framework* DeepMIMO que se ha mostrado a lo largo del documento. A modo de ejemplo y para demostrar el funcionamiento del *scheduler* y demás funcionalidades; en este escenario se consideró una única radiobase (RB12) y se consideraron dos usuarios dinámicos (UE_1 y UE_2) que comenzaron alejados y sin compartir el rayo principal. A lo largo de la simulación se movieron a una posición en la cual la diferencia angular de sus rayos principales fue inferior a 10 grados (ver Figuras 7 y8 donde se observa el movimiento de los usuarios notándose que se van acercando).

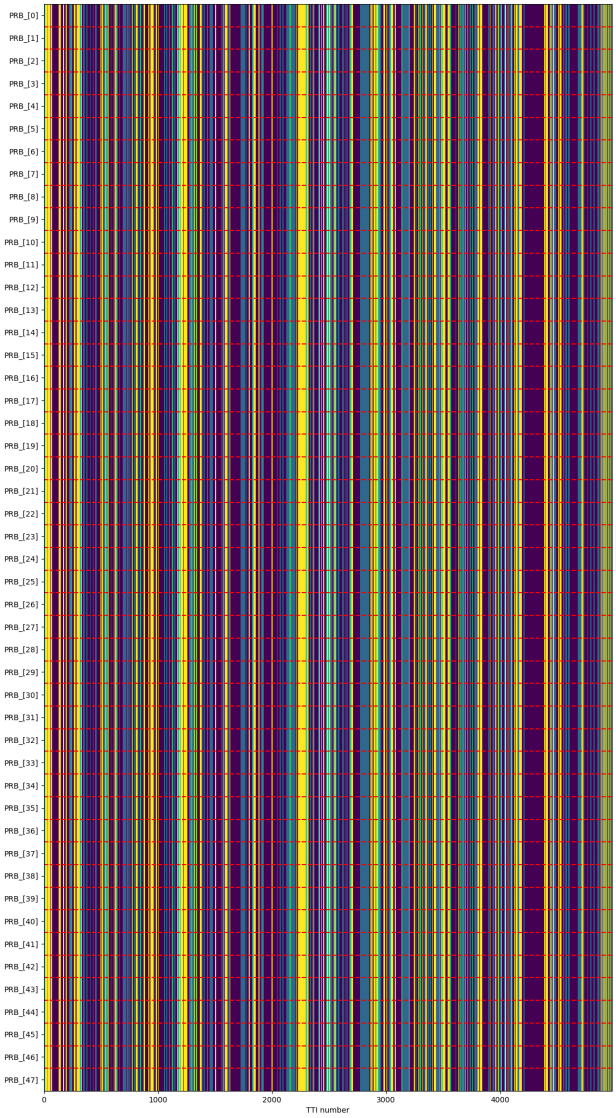

Figura 9. Asignación de recursos obtenida al utilizar dos usuarios que se acercan hasta compartir rayo principal. Líneas azules corresponden a recursos asignados al UE_1, líneas amarillas recursos asignados al UE_2, líneas verdes recursos asignados a ambos usuarios y las violetas corresponden a recursos no utilizados. La simulación fue configurada para durar 6 segundos.

En la Figura 9, utilizando el código de colores detallado en la leyenda de la figura, se puede observar que los usuarios comparten recursos hasta el segundo 4, instante en el cual comienzan a compartir rayo principal debido a que el umbral fue establecido en 10 grados para esta simulación. Como el resultado fue el esperado, es decir, los usuarios compartieron PRBs mientras no compartían rayo principal, y dejaron de hacerlo cuando comenzaron a compartir rayo principal (a partir de los 4 segundos no se observan más líneas verdes). Esto demuestra el correcto desempeño del *scheduler* implementado.

Como se mencionó anteriormente, se sugiere consultar demás validaciones realizadas en [13].

V. Conclusiones

En este trabajo se ha logrado obtener una nueva versión del simulador Py5cheSim, la cual es de código abierto y gratuita, desarrollada en Python, y con el código fuente disponible en GitHub [12].

La integración exitosa del *framework* DeepMIMO en la nueva versión del simulador permitió la extensión del modelo de canal, lo que resultó en usuarios dinámicos con un nivel de detalle por PRB y soporte para MIMO masivo mejorado en comparación con su antecesor. Además, se realizaron modificaciones para asegurar la compatibilidad con simulaciones definidas en la primera versión.

La inclusión de información detallada sobre el canal a la que pueden acceder los distintos *schedulers* evidenció la necesidad de dejar disponible un algoritmo para que futuros usuarios puedan comprender mejor cómo utilizar dicha información. En este sentido, se desarrolló un *intraslice scheduler* que utiliza esta información y cuyo funcionamiento está basado en el trabajo descrito en el artículo *Optimum Resource Allocation in MU-MIMO OFDMA Wireless Systems* [14]. La existencia de algoritmos que tengan en cuenta la nueva información manejada por los *schedulers* demuestra la utilidad del trabajo realizado.

Sin embargo, se reconoce que Py5cheSim v2.0 hereda algunas simplificaciones del simulador original, como la falta de soporte para tráfico bidireccional, manejo de errores en la transmisión por el canal físico, y soporte para otros perfiles de tráfico. Estos aspectos pueden ser abordados en trabajos posteriores para mejorar aún más la funcionalidad del simulador.

Agradecimientos

Este trabajo se desarrolló en marco de un proyecto de fin de carrera por lo tanto los autores desean agradecer al tribunal y a todos los docentes que formaron parte directa o indirectamente.

Referencias

[1] IMT-2020 Radio Interface Standardization Trends in ITU-R. NTT DOCOMO Technical Journal, 19(3), Jan. 2018

[2] E. Dahlman. “5G NR The Next Generation Wireless Access Technology.” Academic Press, 2021. ISBN 978-0-12-822320-8.

[3] G. Pereyra. “Scheduling in 5g networks : Developing a 5g cell capacity simulator.” Master’s thesis, Universidad de la República (Uruguay). Facultad de Ingeniería. IIE, sep 2021

[4] G. Pereyra, C. Rattaro and P. Belzarena, “Py5cheSim: a 5G Multi-Slice Cell Capacity Simulator”, 2021 XLVII Latin American Computing Conference (CLEI), Cartago, Costa Rica, 2021, pp. 1-8, doi: 10.1109/CLEI53233.2021.9640086.

[5] L. Inglés (2023.). “Aprendizaje profundo para la asignación de recursos en redes 5G”. Tesis de maestría. Universidad de la República (Uruguay). Facultad de Ingeniería.

[6] G. Pereyra, L. Inglés, C. Rattaro and P. Belzarena (2022). “An open source multi-slice cell capacity framework”. CLEI Electronic Journal, vol. 25, no 2, may. 2022, pp. 1-21.

[7] A. Alkhateeb. “DeepMIMO: A generic deep learning dataset for millimeter wave and massive MIMO applications.” In Proc. of Information Theory and Applications Workshop (ITA), pages 1–8, San Diego, CA, Feb 2019.

[8] S. Pratschner, B. Tahir, L. Marijanovic, M. Mussbah, K. Kirev, R. Nissel, S. Schwarz, M. Rupp, “Versatile mobile communications simulation: The Vienna 5G link level simulator”, EURASIP Journal on Wireless Communications and Networking, v 2018, n 1, p 1–17, 2018, Springer

[9] 5G Toolbox from Matlab, https://la.mathworks.com/products/5g.html, Accessed: 2023-01-04

[10] N. Patriciello, S. Lagen, B. Bojovic, and L. Giupponi, “An E2E simulator for 5G NR networks” , Simulation Modelling Practice and Theory, volume 96, page 101933, 2019, Elsevier

[11] Wireless InSite 3D Wireless Prediction Software. RemCom Inc. https://www.remcom.com/wireless-insite-em-propagation-software.

[12] D. Sanchez, M. Trujillo and P. Varela (2023). Py5cheSim v2.0 (Version 1.0.0) [Computer software]. https://github.com/charluy/PFC

[13] D. Sanchez, M. Trujillo and P. Varela (2023.). Py5cheSim v2.0 : Extensión de funcionalidades de un simulador de redes 5G para ensayo de asignación de recursos. Tesis de grado. Universidad de la República (Uruguay). Facultad de Ingeniería..

[14] C. Bontu , J. Ghimire and A. El-Keyi, 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring). Optimum Resource Allocation in MU-MIMO OFDMA Wireless Systems. 2020

[15] S. Ju, O. Kanhere, Y. Xing and T. Rappaport, “A millimeter-wave channel simulator NYUSIM with spatial consistency and human blockage”, 2019 IEEE global communications conference (GLOBECOM), pages 1–6, 2019, IEEE.