UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería Telemática
Trabajo Fin de Grado

Coloración de grafos con restricciones y aplicaciones en redes

Autor: Adrián Gallego Sánchez

Tutor/es: David Orden Martín y José Manuel Giménez Guzmán

TRIBUNAL:

Presidente: José Manuel Giménez Guzmán

Vocal 1º: Iván Marsá Maestre

Vocal 2º: David Orden Martín

FECHA: 06 de Julio de 2017

"Knowledge, like air, is vital to life. Like air, no one should be denied it."
— Alan Moore, V for Vendetta"

Tabla de contenido

Índices

Índice de Imágenes

Índice de Cuadros de Texto

Índice de Tablas

1. Introducción

1.1. Resumen

El avance en las técnicas de **simulación** y modelado de redes a través de **grafos** ha permitido el desarrollo de distintos métodos de **optimización** de estas. En redes como las generadas por el estándar Wifi IEEE 802.11n, estos métodos han hecho posible la **optimización** de las conexiones de distintos tipos de dispositivos a los Puntos de Acceso disponibles, es decir, la reducción de las interferencias entre estos. El objetivo de este Trabajo de Fin de Grado[1] es el de aplicar dichas técnicas a distintos tipos de redes y generar varias **restricciones** que adapten el modelo a distintas condiciones que se pueden dar en la realidad, lejos del modelo de comportamiento ideal.

1.2. Abstract

The advance in the simulation and modeling techniques for networks through **graphs**, has allowed the development of different **optimization** methods. In networks such as those generated by the IEEE 802.11n Wifi standard, these methods have achieved to **optimize** the connections of different types of devices to the available Access Points, that is, to reduce the interference between them. The objective of this End-of-Grade Work is to apply these techniques to different types of networks and generate several **constraints** that adapt the model to different conditions which can occur far from the ideal behavior model.

1.3. Palabras Clave

Coloración, grafos, restricciones, centralidades, simulación, optimización.

1.4. Resumen Extendido

El objetivo principal de este TFG es el de llevar a cabo una serie de **simulaciones** que permitan reflejar el comportamiento de varios tipos de escenarios bajo una serie de **restricciones,** de la manera más exacta, sobre redes de distinto volumen o densidad de dispositivos (clientes, proveedores, PA…). Para ello, se van a emplear técnicas de **coloración** de **grafos** a partir de métodos de negociación automática v.gr. "Hill Climber, Simulated Annealing…" que permitirán llevar a cabo la **optimización** de las conexiones entre los distintos dispositivos que compongan la red. En el caso particular de las redes IEEE 802.11n, estas técnicas permitirán que la asignación de canales entre los puntos de acceso que compongan la red y que estén situados en un perímetro o radio de interferencia mutua, puedan llegar a alcanzar una mayor eficiencia a la hora de asignar las distintas frecuencias disponibles, consiguiendo así una menor interferencia que la dada por la asignación aleatoria de canales por defecto.

Las **restricciones** a las que se hace referencia en el párrafo anterior estarán fundamentadas en casos concretos que desvíen el escenario del punto ideal en el que todos los nodos que componen el **grafo** encargados de asignar frecuencias puedan ser "manipulados" por el negociador automático en cuestión. Se hará especial hincapié en la **restricción** en la que un porcentaje determinado de dichos nodos no pueda ser manipulado, sino que tengan una frecuencia fija e inmutable. Este caso, llevado fuera de las **simulaciones** que abordan este TFG, es un claro ejemplo de aquellas redes en las que uno o varios de los puntos de acceso que las componen no son controlados por un mismo gestor o proveedor, provocando que no se pueda actuar sobre estos y que la técnica o algoritmo de negociación automático

[1] Desde este punto se hará referencia a dicho trabajo con las siglas "TFG".

empleado se vea limitado a optimizar las frecuencias sobre aquellos dispositivos que pueda manipular con la condición de que existan frecuencias fijas.

De manera añadida, otro de los objetivos perseguidos en este TFG es el de investigar y estudiar el comportamiento bajo estas **restricciones** con el añadido de una serie de condiciones secundarias o de contorno, entre ellas, se ha contemplado: usar una misma frecuencia para todos los nodos con frecuencia fija, usar frecuencias asignadas de manera aleatoria a dichos nodos, aplicar métodos de elección de los subnodos elegidos en base a funciones aleatorias o **centralidades**[2] y, para acabar, métodos de ordenación de los subnodos elegidos, entre otros, para los casos en que se ha empleado una **centralidad** como método de selección, nodos con mayor **centralidad**, menor y distribución normal. El objeto primero de exponer los escenarios a este tipo de condiciones es el de concluir en qué casos se obtiene una mayor eficiencia en los escenarios dados, en términos de utilidad [*1]* y "Fairness" [1]. Dichas conclusiones serán extraídas de tablas de Excel en las cuales se verán reflejados los resultados para cada posible escenario.

Para finalizar los objetivos de este trabajo, cabe mencionar que se ha procurado realizar una tarea de **optimización** del código empleado para generar las **simulaciones** de manera que, futuros o actuales usuarios del citado código vean su labor facilitada y su inversión temporal reducida en un porcentaje importante[3].

Para poder desarrollar y alcanzar los objetivos ya citados se han empleado una serie de scripts en Python 2.7 generados por el grupo de investigación con el que se ha llevado a cabo en paralelo este trabajo. Dichos scripts permitían generar los datos necesarios para extraer las conclusiones referentes a cada escenario con unas características compuestas, la labor del autor de este TFG ha sido la de, partiendo de dichos scripts, modificar o añadir la funcionalidad necesaria para cumplir todo lo anteriormente mencionado.

Finalmente, aparte de todo el desarrollo de la aplicación, también se ha llevado a cabo una importante labor de análisis del gran volumen de datos adquirido a partir de los resultados de cada uno de los experimentos lanzados, esta, quizá, haya sido una de las tareas más arduas y que más tiempo ha requerido.

[2] Véase el apartado "Tª Introductoria: Métodos de elección/ordenación".

[3] Véase "Experimentos: Diseño y Desarrollo: Optimización GPU".

2. Teoría introductoria

En redes inalámbricas de alta densidad o redes complejas, uno de los principales factores a tener en cuenta a la hora de hablar de **optimización** es la tasa de interferencias, es decir, las interferencias dadas entre los dispositivos que componen la red. Para ello, como se menciona en la introducción del artículo "Nonlinear negotiation approaches for complex -network optimization (…)"[1] junto a [2][3], es importante considerar la aplicación de técnicas de negociación automática no lineales de manera que se consiga reducir al mínimo el efecto del solapamiento frecuencial[4] que tiende a darse en este tipo de redes, en especial, en redes del estándar IEEE 802.11n entre los distintos canales disponibles para los puntos de acceso. Así pues, partiendo de estas premisas, se ha seguido esta línea de investigación y se han aplicado este tipo de mecanismos condicionados a una serie de **restricciones**[5], mencionadas a continuación, que permitirán abordar un gran rango de casos en un mismo escenario.

2.1. Modelado

Puesto que se ha continuado la línea de trabajo usada por los componentes del grupo de investigación, se ha mantenido el tipo de modelos (esencialmente redes Wifi) y herramientas usadas por estos para el desarrollo de este TFG. En consecuencia, para representar las redes con las que se pretende trabajar, estas han sido modeladas mediante una Tª de la Matemática Discreta conocida como Tª de **Grafos**, en concreto, mediante técnicas de **coloreado** de **grafos**. A continuación, se exponen algunos de los conceptos básicos de la teoría de **grafos** con intención de facilitar la comprensión de este trabajo:

- **Grafo:** Objeto compuesto por dos conjuntos finitos, un conjunto E de **aristas** o arcos, y un conjunto V de **vértices** (nodos). Para ambos conjuntos se aplican las siguientes normas:
  - $Si\ e \epsilon E\ => (v,u) \in VxV \mid v \in e\ y\ u \in e$
  - $Si\ e \in E\ =>, y\ además\ de (v,u)\ \exists\ (x,y) \in VxV => x \in e\ y\ v \in e => \{v,u\} = \{x,y\}$
- **Aristas:** Una arista e ∈ E es dirigida si el primer y el segundo vértice en la anterior condición son iguales, esto es: v=x y y=u.
- **Bucle:** Una arista que conecta un vértice consigo mismo.
- **Subgrafo:** Se considera G1=(V1,E1) un subgrafo de G, si se cumple que $v1 \subseteq V, E1 \subseteq E$ y las condiciones del inicio se mantienen.
- **Conexión:** Los vértices *(v,u) ∈ V* están conectados si existe un camino de *v a u* tal que *(v1,v2), (v2,v3),..., (v(k-1),u) ∈ E.*
- **Grado:** Número total de aristas que confluyen en un vértice.
- **Densidad:** La relación entre el número de aristas de un grafo G y el número máximo de posibles aristas, es decir, el número de aristas de un grafo totalmente conectado.
- **Métricas:**
  - **Distancia:** Representada como d(v,u) entre los vértices v y u, es la distancia más corta entre ambos vértices.
  - **Diámetro:** Representado como d(G) de un grafo G, es la máxima distancia d(v,u) para todo v,u *∈ V.*
  - **Orden:** Número de vértices de un grafo G.

[4] Término que hace referencia al suceso dado cuando varios dispositivos en una misma red emiten en la misma banda de frecuencias y están dentro del radio de interferencia mutua.
[5] Véase apartado "Métodos de elección/ordenación".

La técnica de **coloración** de **grafos**, con el objetivo de representar y optimizar redes formando **grafos**, tiene una gran trayectoria, desde antes de los ochenta ("The Complexity of Near-Optimal Graph Coloring" [4], "New methods to color vertices of a graph"[5]) hasta la actualidad ("Spectrum graph coloring and applications to WiFi channel assignment"[6]). Teniendo el **grafo** a desarrollar, por regla general en los modelos para redes complejas, las siguientes características: se establece un color para cada frecuencia plausible, se colorean aquellos vértices que operen con dicha frecuencia con el color asignado y se establecen aristas con aquellos dispositivos que reciban datos a través de esta frecuencia. Sin embargo, en el caso a desarrollar, se lleva este "estándar" un poco más allá y se distinguen dos tipos de **grafo**:

- **Grafo de Asociación (NNG):** Representa el modelo de red, relacionando mediante las aristas aquellos dispositivos asociados a un determinado punto de acceso.

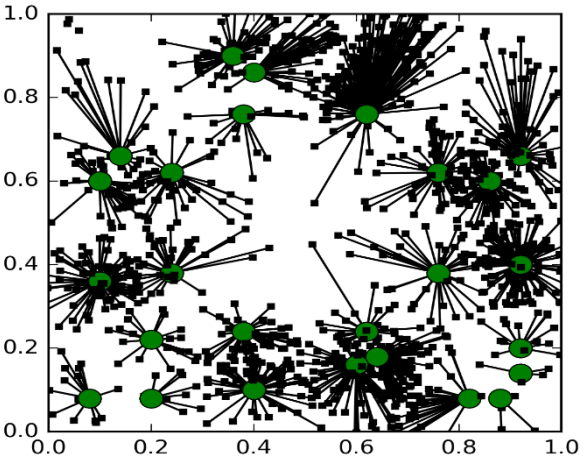


Imagen 1 - Grafo NNG (EPS,UAH) - IEEE 802.11n

- **Grafo de Interferencias (UDG):** En este caso, las aristas representan las interferencias entre dispositivos y puntos de acceso, pudiendo darse tres casos: interferencias **entre dos puntos de acceso**, interferencias **entre punto de acceso y dispositivo** (en caso de no estar asociados) e interferencias **entre dos dispositivos** (siempre que no estén asociados a un mismo punto de acceso).

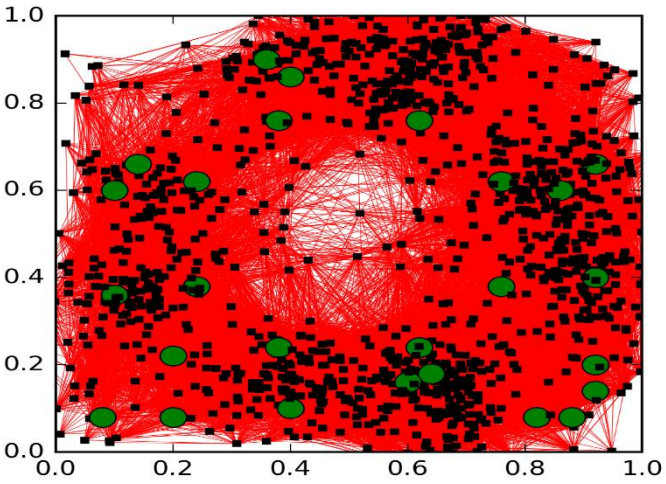


Imagen 2 - Grafo UDG (EPS,UAH) - IEEE 802.11n.

En ambos casos, los vértices pueden ser dos tipos de dispositivos: ***puntos de acceso*** (aquellos nodos que permiten la conexión de otros dispositivos a través de una determinada

frecuencia) y ***clientes*** (aquellos nodos que se conectan a otro dispositivo a través de una determinada frecuencia.

Por otro lado, los **grafos** UDG tienen en cuenta la potencia de interferencia entre nodos en base al peso o valor de las aristas. Para obtener dicho valor, se tienen en cuenta factores como: la **distancia entre nodos con interferencia mutua**, **periodos de actividad**, es decir, se consideran como posibles focos de interferencia aquellos nodos que estén activos, descartándose aquellos que atraviesen un periodo de inactividad y, por último, se otorga un **peso inicial a cada par de colores** dado por la matriz de interferencias del artículo [21], la cual va a permitir obtener la relación de interferencia entre frecuencias o canales adyacentes para el caso del estándar 802.11n.

Cabe destacar que, dado que los modelos hasta ahora generados por el equipo de investigación han sido para redes operando bajo el estándar IEEE 802.11n, se hubo de tener en cuenta una serie de condiciones para observar los efectos de las interferencias, estos son:

- **Calidad de Señal:** Solo se han tenido en cuenta las interferencias generadas por otros nodos operando bajo el protocolo IEEE 802.11n. Obviando cualquier otro dispositivo que emita en la frecuencia de 2.4GHz.

- **Radio de Cobertura**: A cada punto de acceso se le ha asignado un radio de cobertura bajo el cual la señal que transmite es óptima. En consecuencia, cualquier nodo que esté fuera de dicho radio de cobertura, aunque genere cierta interferencia, ha sido despreciado.

- **Clientes**: Se ha operado bajo la premisa de que todo cliente asociado a un punto de acceso no puede generar interferencias con otros clientes asociados a ese mismo punto de acceso.

2.2. Negociadores Automáticos

Una vez más, se hará uso de las implementaciones ya existentes en los scripts de Python antes mencionados para aplicar las distintas técnicas de negociación usadas. La información de este apartado ha sido extraída, esencialmente y con pequeñas modificaciones del artículo "COREDEMA", sección 3 [1]. Los autores de ambos, artículo e implementaciones, definen estas técnicas en base a tres características:

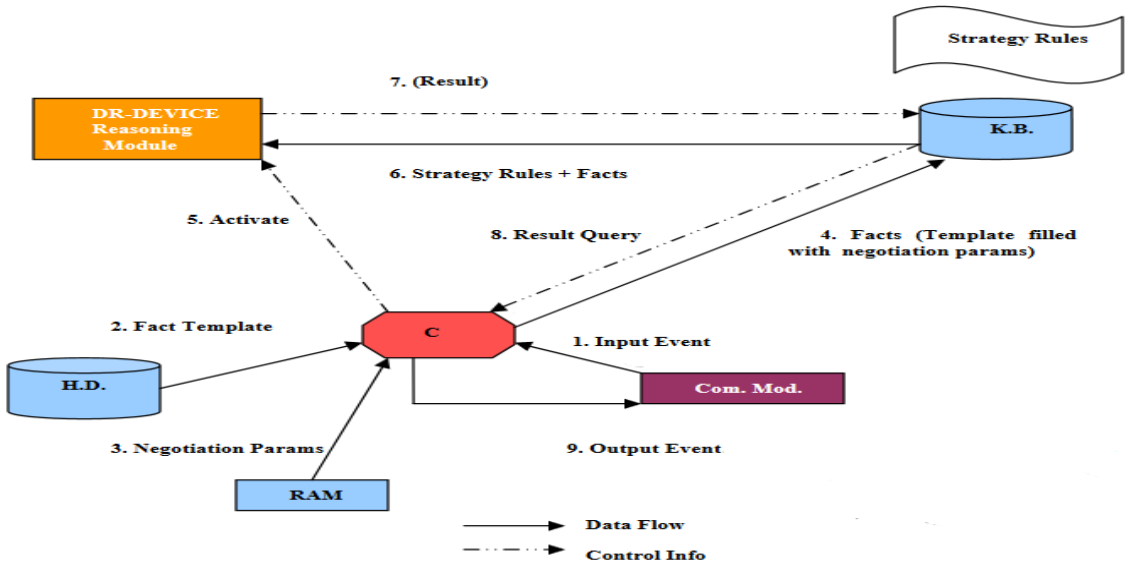


Imagen 3 – Arquitectura Agentes Negociación.

- **Dominio de Negociación:** Qué se negocia y quién lo negocia.
- **Protocolo de interacción:** Reglas que definen el proceso de negociación.
- **Mecanismos de decisión:** Estrategias que guían a los agentes de negociación a lo largo de las diferentes fases del proceso. En este TFG se han usado 3 de estos mecanismos, los cuales son detallados más abajo.

2.2.1. Dominio de Negociación

Para los escenarios generados y usados en este TFG, se ha asumido un dominio multiatributo, el cuál viene dado por la siguiente expresión $S = \{ S_i \mid i \in 1,\dots,nAP \}$, donde *nAP* es un número "n" de puntos de acceso en la red dada y S_i representa el canal o frecuencia asignado al Punto de Acceso "i", siendo los posibles valores de estos canales $S_i \in \{ 1,\dots,11 \}$. A pesar de que el estándar IEEE 802.11n contempla, para la banda de 2,4GHz, el potencial de uso de 13 canales de 40MHz, en este TFG se aplican las reglas seguidas por los autores del antes citado artículo, esto es, **11 canales de 20MHz.**

Siguiendo estas mismas reglas, se tienen dos proveedores (*ISPs*) que tendrán control sobre la asignación de canales sobre aquellos puntos de acceso que les pertenezcan. Así pues, se pueden definir los agentes de negociación con la siguiente expresión $P = \{p_1,p_2\}$, estos serán los encargados de generar el resultado de la función de utilidad aplicada en cada caso.

2.2.2. Protocolo de Interacción

Como ya se había mencionado, una de las condiciones o dimensiones básicas para automatizar un proceso de negociación entre agentes es la existencia de un protocolo de interacción, el cual se encargará de definir y permitir las secuencias de acciones propias a la negociación necesaria en este TFG. En este caso, debido a que la mayoría de escenarios de negociación serán altamente no lineares, se ha propuesto el uso de un protocolo de

mediación simple de texto que, en su versión simplificada sigue los pasos del siguiente diagrama de flujo:

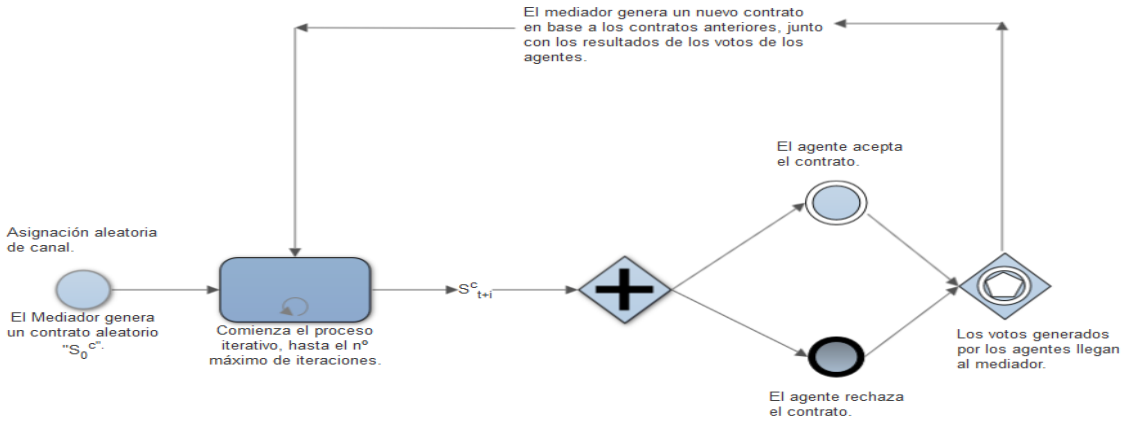


Imagen 4 – Diagrama Protocolo Interacción.

Como se puede observar, al inicio el mediador genera un contrato aleatorio que equivale a asignar un canal de frecuencia *Wifi* aleatorio a cada punto de acceso de cada proveedor. En cada nueva iteración el mediador propone un nuevo contrato a los proveedores, basándose en contratos anteriores y en los votos recibidos por los agentes sobre dichos contratos. Los agentes pueden aceptar o no el contrato, repitiéndose este proceso hasta alcanzar el número máximo de iteraciones o encontrar una situación de parada. Siendo $t \in \{0,\dots,M\}$ y $M \equiv$ NºMáximo iteraciones.

2.2.3. Mecanismos de Decisión

En este punto, conociendo las reglas del proceso de negociación, se ha de hablar de cómo se generan los contratos (en el caso del mediador) y en base a qué condiciones se votan, es decir, bajo qué condiciones se acepta o se deniega un nuevo contrato (en el caso de un agente). Con este objetivo, el mediador opera de la siguiente manera:

- Si los agentes operando en el escenario, en este caso ambos proveedores, votan a favor del contrato generado $\boldsymbol{S_t^c}$, este se empleará como contrato base $\boldsymbol{S_b}$ para generar el siguiente contrato $\boldsymbol{S^c_{t+1}}$.
- Para generar el nuevo contrato $\boldsymbol{S^c_{t+1}}$ el mediador actúa sobre el contrato base $\boldsymbol{S_b}$ y muta una de sus propiedades de manera aleatoria, esto implica seleccionar un punto de acceso aleatorio y le otorgarle un nuevo canal aleatorio.
- Al llegar a $\boldsymbol{M}$ el mediador toma el último contrato aceptado por ambos agentes como contrato final.

Para los agentes de negociación se han empleado mediadores que basan sus decisiones en algoritmos de búsqueda local. Este tipo de algoritmos permiten un uso reducido de memoria y la obtención de soluciones plausibles en grandes (o infinitos) espacios de estados. A continuación, se exponen estos mediadores en detalle:

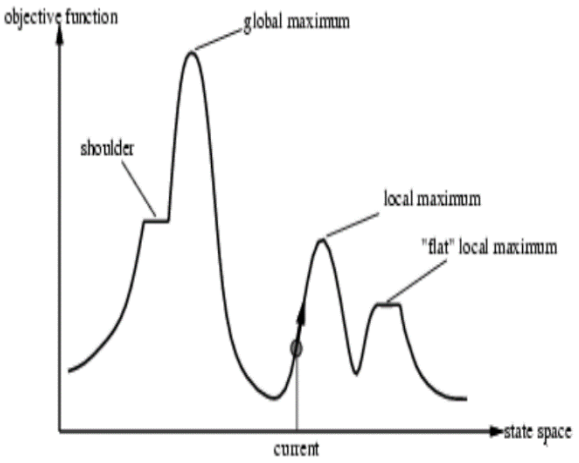


Imagen 5 – Hill-Climber Local Optima.

- ***Hill-Climber (HC):***

Este tipo de técnicas han sido utilizadas para generar gráficos de particiones [7], aprender comportamientos en robots [8] y para construir sistemas *"Steiner" [9]* entre otros. En este TFG, el comportamiento a seguir por este mediador es el de maximizar la utilizad, por ello solo votará a favor de un nuevo contrato en el caso de que sea mejor que el anterior contrato aceptado. Al inicio, al no existir contrato previo aceptado, siempre votará a favor.

A pesar de que este tipo de negociador/mediador suele obtener excelentes resultados en espacios unimodales o no lineares, en una gran mayoría de aplicaciones en el mundo real tiende a quedarse "estancado". Las principales razones de esto son expuestas en [10], en resumen, este algoritmo tiende a quedarse en máximos locales, problema conocido como ***"foothill-problem"***. Para evitar este inconveniente, existen variaciones de este algoritmo como pueden ser: ***HC con paseos aleatorios, HC estocástico o HC junto a algoritmo genético.***

- ***Annealer (SA):*** El anterior mediador, como ya se ha expuesto, tiene el potencial problema de estancarse en óptimos locales y no alcanzar un óptimo global, dado el tamaño del espacio de soluciones. La técnica "**Simulated Annealing**" (temple simulado) evita el estancamiento en máximos locales.
  Este algoritmo proviene del proceso físico de templado de metales, en el cual, para conseguir un metal con las propiedades deseadas, es necesario controlar la velocidad del templado o enfriamiento, en caso de llevarse a cabo de manera satisfactoria, el estado final del metal es un estado de mínima energía. En resumen, para evitar los máximos locales este mediador acepta contratos peores al último generado con una cierta probabilidad que viene controlada por la temperatura de templado. De manera formal, podría ser expresado como:
  "Dado un conjunto finito *S* de posibles contratos y una función real $\boldsymbol{f}: \boldsymbol{S} \rightarrow \mathfrak{R}$ que se corresponde con la función de utilidad, la **optimización** global viene dado por la búsqueda de $\boldsymbol{s \epsilon S}$, tal que $\boldsymbol{f}(\boldsymbol{s}) \geq \boldsymbol{f}(\boldsymbol{s}') \; \forall \boldsymbol{s}' \in \boldsymbol{S}$. Este mediador requiere contemplar el concepto de "vecindario" sobre el espacio *S*, siendo el vecindario $\boldsymbol{N(s_t)}$ del contrato actual $\boldsymbol{s_t \epsilon S}$ el conjunto de nuevos contratos **s'** que pueden generarse desde $\mathbf{s_t}$, es decir, $\boldsymbol{N(s)}$ serán pequeñas variaciones de $\boldsymbol{s}$. Hasta aquí, el comportamiento es esencialmente el mismo que en *Hill Climber*, como ya se ha mencionado la diferencia radica en que en este caso se puede escapar de los máximos locales aceptando de manera probabilística contratos peores al último generado, $\mathbf{s_t}$. Sean $\Delta \boldsymbol{u}$ la pérdida de utilidad asociada al nuevo contrato $\mathbf{s_{t+1}}$ y $\boldsymbol{\tau}$ la temperatura de templado, la probabilidad $\boldsymbol{P_a}$ de aceptación de un contrato peor será $\mathbf{P_a} = \mathbf{e}^{\frac{\Delta \mathbf{u}}{\tau}}$. "

  Por último, a través de las *M* iteraciones en que el algoritmo será ejecutado, la temperatura $\boldsymbol{\tau_1, \tau_2, \ldots, \tau_t \geq 0} \; \forall \boldsymbol{t}$ guiará la **optimización**. En la siguiente tabla se puede observar el comportamiento de este mediador:

Algoritmo *Simulated Annealing*

0. $s_0 \leftarrow Contrato\ Aleatorio$
1. ***For*** $t = 0$ ***to*** $M - 1$
2. *Se elige* **s'** $\leftarrow Elemento\ aleatorio\ de\ N(s_t)$
3. **if** $f(s') \geq f(s_t)$
4. $s_{t+1} \leftarrow s'$
5. **else**
6. $s_{t+1} \leftarrow s' con\ probabilidad\ P_a$
7. $s_{t+1} \leftarrow s_t\ con\ probabilidad\ (1 - P_a)$
8a. *Retorna* s_M
8b. *Retorna* $s_t, 0 \leq t \leq M | f(s_t)_{max}$

Tabla 1 - Simulated Annealing.

Cabe destacar que si la temperatura τ baja lo suficientemente despacio existe una probabilidad $P \approx 1$ de alcanzar el máximo global, esto es:

$$\mathbf{P_r(s_M \in R) \rightarrow 1\ mientras\ que\ M \rightarrow \infty}$$

Donde $R \subset S$, denota el conjunto de contratos óptimos globales.

2.2.4. Otras Técnicas

Aparte de las técnicas de negociación vistas en el apartado anterior, también se ha contemplado el uso, para obtener una referencia, de dos métodos más. En este apartado, además de mencionar y detallar estos métodos, se hará un pequeño inciso con otras posibles técnicas que podrían haberse empleado/implementado para llevar a cabo los experimentos de este TFG, quedando, a elección de futuros investigadores, implementarlas en sus propios trabajos:

- **Ref. Aleatoria:** La técnica más simple de implementar, cada punto de acceso elige un canal o frecuencia de manera aleatoria[6].

- **Ref. Least Congested Channel Scan (LCCS):** Se ha querido implementar y observar los resultados de una técnica de especial eficiencia en redes de tipo WLAN no coordinadas. Esta técnica fue propuesta por Achanta M. en [11], a grandes rasgos, este sistema permite que cada punto de acceso busque el canal con menor carga (aquel que tenga el menor número de clientes asociados). Una vez encontrado, conmuta a dicho canal hasta que en un próximo escaneo se encuentre un canal menos congestionado. Para conseguir esto, cada punto de acceso analiza cada canal en busca de tramas lanzadas por AP vecinos (cada trama contiene información acerca del número de clientes asociados a cada AP), conociendo así los clientes asociados a cada uno de sus vecinos. De esta manera, después de escanear todos los canales, el punto de acceso es capaz de conocer cuántos clientes están asociados a cada canal y, por tanto, capaz de elegir aquel canal con menor congestión o carga. Un dato importante a destacar del algoritmo LCCS es que presupone que todos los clientes generan la misma cantidad de tráfico, en consecuencia, a mayor número de clientes conectados, mayor cantidad de tráfico.

[6] En base a una función del módulo Numpy, pseudo-aleatoria.

Algunas de las alternativas que se han planteado para sustituir o complementar los resultados de esta última técnica han sido:

- **Algoritmo propuesto por "Mi, P. & Wang, X." en [12]:** Este algoritmo tiene la problemática de no determinar la interferencia para los clientes y el valor del parámetro "Fairness" no está garantizado, debido a esto, a pesar de ser interesante, no tiene un gran peso dentro del caso que cubre este TFG.

- **Leung, K. K. and B.-J. Kim. [13], MinMax approach:** En este método, la utilización efectiva de un canal viene dada por la fracción de tiempo en que el canal está asignado a un punto de acceso o este se denota ocupado debido a la interferencia co-canal de un punto de acceso vecino. Se distinguen dos tipos de interferencias, las cuales permiten minimizar la máxima utilización de un canal en aquel punto de acceso con mayor carga. En resumen, el algoritmo empieza asignando canales aleatorios a cada punto de acceso en la red y continúa ajustando de manera aleatoria la asignación de canal en aquel punto de acceso que represente un cuello de botella, de manera que la utilización efectiva del canal en este punto de acceso sea minimizada, resultando así en una red poco congestionada.

- **CACAO (Client Assisted Channel Assigment Optimization):** Propuesto por **Xiaonan Yue et al. en [14]**, este método permite encontrar la carga de los canales a partir del nivel de interferencia dado en un AP. En este caso, los clientes conmutarán de canal de manera periódica para encontrar la carga requerida en dichos canales, dicha carga es evaluada en términos de "Tasa de bits". El principal problema de este algoritmo es que los clientes también sufren interferencias cuando están dentro del rango de detección de la portadora de los nodos interferentes.

- **A.Mishra et al.[15] , Channel hopping approach:** Cada punto de acceso tiene asignado una secuencia única de canales, debido a esto, el punto de acceso irá "saltando" de un canal a otro de dicha secuencia en intervalos de tiempo. La secuencia es asignada a cada punto de acceso de tal manera que cada punto de acceso "salta" al siguiente canal al finalizar el intervalo de tiempo pertinente.

2.3. Métodos de elección/ordenación/Restricciones

Hasta este punto se han explicado los tipos de **grafos** utilizados para modelar el problema al que se enfrenta este TFG, así como los distintos mecanismos de negociación automática que permiten la **optimización** de la asignación de frecuencias en una red, en el caso de las redes gobernadas por el protocolo 802.11n, la asignación de canales en los puntos de acceso con el objetivo de minimizar la interferencia entre estos. Sin embargo, lo expuesto hasta el momento no es más que la base sobre la que se fundamente el verdadero objetivo de este trabajo: aplicar distintas **restricciones** a los escenarios y optimizar los casos en medidas de utilidad y "Fairness".

En consecuencia, en esta sección se pasan a explicar los tres pilares que conforman dichas **restricciones**: condiciones limitantes, métodos de elección de los subnodos[7] y métodos de ordenación.

2.3.1. Condiciones Limitantes

Es posible distinguir entre dos tipos de condiciones en este caso: condiciones que limitan el número de nodos sobre los que el negociador va a poder actuar, es decir, condiciones con respecto al número de nodos y, condiciones con respecto al **coloreado** de estos.

2.3.1.1. Condiciones respecto al número de nodos

Este tipo de condición ha sido impuesta con varios objetivos, todos ellos expuestos a continuación:

- **Permitir reflejar condiciones de escenarios reales** en los que, un número determinado de nodos, son controlados por otros proveedores, en otras palabras, son nodos sobre los que no se puede actuar. En el caso del protocolo Wifi y de los experimentos lanzados, dichos nodos serían puntos de acceso con un canal de frecuencia fijo e inmutable, sobre los cuales los métodos de negociación no podrán actuar.
- **Estudiar el comportamiento de los escenarios utilizados, en términos de utilidad y "Fairness"**, cuando dicho conjunto de nodos (a partir de ahora se hará referencia a este subconjunto como "grupo de subnodos o subconjunto") varía, es decir, observar cuáles serían los nodos o puntos de acceso más o menos conflictivos si su canal es invariable, v. gr. no tendrá el mismo efecto, en términos de utilidad, fijar los canales de los nodos más centrales[8] de un **grafo** que de aquellos que sean menos centrales.
- **Estudiar, a su vez, qué efecto tiene sobre el tiempo de ejecución, consumo de memoria/CPU**, utilidad resultante… al lanzar los experimentos. A priori, se espera que *el tiempo* de ejecución *disminuya de manera inversamente proporcional al porcentaje* de subnodos escogido, puesto que la condición que se impone acerca de la inmutabilidad del canal en dichos nodos, impidiendo a los negociadores actuar sobre ellos implicará, en términos computacionales, que cuando el negociador vaya a aplicar el algoritmo o mecanismo de interacción sobre algunos de estos subnodos, exista una condición que lo impida y salte a la siguiente iteración del algoritmo, reduciendo el número total de operaciones a desarrollar por este. Por otro lado,

[7] El término "subnodos" hace referencia al subconjunto de nodos cuya frecuencia será inmutable, es decir, aquellos nodos sobre los que los negociadores automáticos no podrán actuar.

[8] Para más información acerca de la "centralidad" de los nodos, véase el apartado: "*Métodos de elección de subnodos".*

también *se prevé que la utilidad total disminuya de manera inversamente proporcional al porcentaje de los subnodos elegido*, esto es, a mayor número de subnodos, menor utilidad ya que, el número de nodos sobre el cual el algoritmo puede actuar es menor, reduciendo así el potencial rango de **optimización** al que podría llegarse.

Habiendo explicado los objetivos de esta condición, se da paso a detallar los porcentajes contemplados. Dichos porcentajes del total de subnodos se han limitado a tres: **el 10%, el 20% y el 30%.** La elección de estos porcentajes se debió, principalmente, a lo expuesto en el tercer objetivo de este mismo apartado, encontrar un punto intermedio que optimizase la relación entre el tiempo de ejecución de los experimentos y la utilidad resultante.

2.3.1.2. Condiciones respecto al coloreados de nodos

Como queda indicado en el título de este trabajo, uno de los objetivos de este, es el de la **coloración** de **grafos**. En el caso que nos ocupa y como ya se ha explicado en la introducción del apartado "*Modelado*", colorear un nodo en los **grafos** empleados equivale a asignar una frecuencia de operación al nodo **coloreado**, en términos del protocolo 802.11n, implica asignar un canal a los puntos de acceso y sus clientes. Partiendo de esta base, se recuerda al lector, que se ha contemplado el uso de 11 canales de 20Mhz en el rango de frecuencias de los 2.4Ghz, en vez de los 13 posibles contemplados en el estándar del IEEE 802.11n, tampoco se ha investigado el rango de los 5Ghz, más característico de los estándares "*802.11a*" y "*802.11ac*".

Cabe destacar que, a pesar de tener un total de 11 canales disponibles en los experimentos, en su mayor parte, tan solo se hará uso de aquellos canales que no superpongan sus frecuencias, evitando así mayores interferencias. Como se puede observar en la *imagen* 6, estos canales son: **1,6 y 11.**

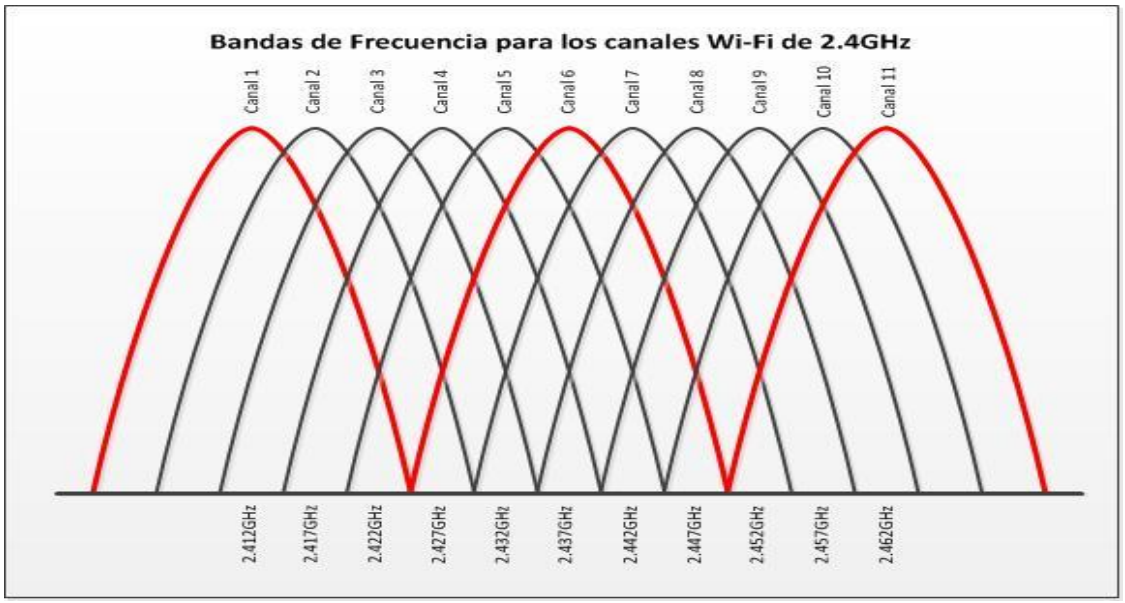


Imagen 6 - Hill-Climber Local Optima.

Los métodos aplicados en los experimentos para la **coloración** del grupo de subnodos, se exponen a continuación:

- **Fixed Color:** Todos los subnodos escogidos son coloreados con un mismo color, a todos se les aplica el mismo canal de los tres posibles contemplados. Este método pretende reflejar aquellos casos en los que todos los nodos de un mismo proveedor, aquellos sobre los que no se puede actuar, tienen el mismo canal asignado.
- **Random Color:** Cada uno de los subnodos recibe un color seleccionado de manera aleatoria entre los 11 posibles.
- **TSC-DSATUR:** Por último, se quiso introducir una técnica de **coloreado** basada en el algoritmo DSATUR, dicha técnica no ha llegado a implementarse en el presente trabajo, sin embargo, se detallarán las características de esta por si fuera de alguna utilidad en futuros proyectos, además de aportar una pequeña introducción a las ventajas en el uso de este tipo de técnicas.
  El problema matemático conocido como "*VCP" – Vertex Coloring problem,* y sobre el cual se trabaja, en parte, en este TFG, requiere que se asigne un color a todo vértice en un **grafo** dado de manera que los vértices adyacentes sean coloreados con un color distinto y que el número total de colores empleados sea el mínimo posible. Los orígenes del "*VCP*" se remontan a una carta escrita a *William Hamilton* por el renombrado matemático *Augustus de Morgan* en 1852, en ella, los principios del famoso *Teorema de los cuatro colores* podían ser dilucidados. Como se había empezado a mencionar, el problema *VCP* es un conocido y muy estudiado problema *NP-complejo* en teoría de **grafos** el cual ha atraído una gran atención de investigadores debido a las implicaciones que tiene en la *teoría computacional* y el gran rango de aplicaciones que tiene en ingeniería. Por ello, en este caso, se decidió trabajar sobre un caso particular de este problema el denominado en [5] como "*Threshold Spectrum Coloring*"[9], el cual se define como: dado un **grafo** *G* y un espectro de *k* colores, dotado de una matriz de interferencias W de dimensiones *k x k*, se persigue el objetivo de determinar el mínimo umbral $t \in \mathbb{R}_{\geq 0}$ tal que (G,W) admita un k-**coloreado** *c* en el que la interferencia en todo vértice es, como máximo, *t v. gr.* $I_v(G,W,c) \leq t, \forall v$. Siendo t conocido como el umbral k-cromático mínimo de (G,W).

  Teniendo esto en cuenta, la aplicación de un algoritmo secuencial para el **coloreado** de vértices era clave, por ello se eligió implementar una variación del algoritmo DSATUR[10] que puede ser observada en la siguiente tabla:

Algoritmo de coloreado *TSC-DSATUR* *Tabla 2 - Algoritmo TSC_DSATUR[5].*

Input:
G = (V;E): grafo a ser coloreado; S = {c_i}: espectro de colores/canales
W : matriz de interferencias
k | 2 ≤ k ≤ |S|: nºmáximo de colores a ser usado del espectro S
Output:
c: k-coloreado del grafo G
$c(v) := \emptyset, \forall v \in V;$
while $\exists v \in V \mid c(v) = \emptyset$ ***do***
1 $v = argmax_{x \in V; c(x)=\emptyset} grado\_saturación(x);$
2 $c(v) := argmin_{c_i \mid i \leq k} \sum_{u \in N(v); c(u) \neq \emptyset} W(c(u), c_i)$
end

[9] A partir de este punto se denominará "*TSC*".
[10] Para más información acerca de la evolución de este algoritmo, véase [16][17]

Como se puede observar en la tabla el algoritmo empieza con un color indefinido e itera seleccionando en cada iteración aquel vértice con mayor índice de saturación[11]. En caso de haber un empate, se escoge el vértice con mayor grado y, si hubiera un doble empate, se escoge un vértice aleatorio de aquellos cuyo grado sea alto. Una vez que un vértice *v* ha sido elegido, se le asignará un color del conjunto disponible (11 en este caso) procurando que la interferencia en dicho vértice sea minimizada.

2.3.2. Métodos de elección de los subnodos

Habiendo explicado en qué consisten las **restricciones** aplicadas en los experimentos y los métodos de elección de color para cada uno de los subnodos, se pasa a explicar en este apartado las técnicas empleadas para designar **qué** nodos del **grafo**, concretamente, qué puntos de acceso, conformarán el porcentaje de subnodos cuyo canal será inmutable. Con este objetivo, se han empleado dos tipos de métodos, uno basado en una elección aleatoria y otro en base al concepto de **centralidad**:

- **Random:** Los puntos de acceso escogidos para conformar el conjunto de subnodos son elegidos de manera aleatoria.

- **Centralidad:** El concepto de **centralidad** fue investigado por primera vez en el "*Group networks Lab.*" *del* MIT al final de la década de los 40, bajo la dirección de *Bavelas*. Siendo conducidos y detallados los primeros estudios por *Harold Leavitt* (1949), *Sidney Smith* (1950) y el propio *Bavelas* (1950). Con el paso de los años las aplicaciones para este concepto se han multiplicado, desde la integración de la vida política en la sociedad Indú (*Cohn* y *Martiott* - 1958), pasando por canales de comunicación para el desarrollo urbano (*Pitts* – 1965), hasta las aplicaciones en **grafos** que se emplean en este trabajo. Además, junto al desarrollo de estas aplicaciones se han ido proponiendo distintas medidas de la **centralidad**, las cuales pueden dividirse en dos ramas: medidas *radiales*, toman como punto de referencia un nodo dado que inicia o termina recorridos por la red, y medidas *mediales*, toman como referencia los recorridos que pasan a través de un nodo dado.
  Para este TFG, la definición de **centralidad** que se ha usado es: "La medida posible de un vértice en un **grafo** que determina su importancia relativa dentro de dicho **grafo**."
  Partiendo de ello, se han aplicado las siguientes medidas de **centralidad**:

  - **Centralidad de grado:** Se trata de la **centralidad** más "simple" y quizá la más intuitiva ya que se corresponde con el número de nodos a los que se encuentra conectado un determinado nodo del **grafo** (en teoría de **grafos** esto se conoce como el grado del vértice). Formalmente, puede definirse dado un grafo $G:=(V,E)$ siendo V su conjunto de vértices y E su conjunto de aristas, para cada nodo $v \in V$ como:

$$\mathbf{C_g(v)} = \mathrm{grado(v)}.$$

[11] Aquel vértice cuya mayor parte de vecinos ya hayan sido coloreados.

- **Centralidad de cercanía o "closeness"**: Se trata de la más empleada de las medidas radiales a las que antes se hacía referencia, fue definida por Murray Beauchamp en 1965[18]. Consiste en calcular el promedio de las distancias más cortas desde un nodo al resto de nodos, formalmente, puede definirse como: para un nodo *u* es la suma recíproca de las distancias más cortas desde *u* al resto de *n-1* nodos. Sin embargo, dado que la suma de distancias depende del número de nodos en un **grafo**, su medida es normalizada, en este trabajo, por la suma de mínimo posible de distancias *n-1, siendo d(v,u) la distancia entre v y u, y n el número total de nodos del* ***grafo:***

$$\boldsymbol{C(u)} = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$$

- **Centralidad de Intermediación o "betweenness":** Este tipo de medida de **centralidad** se encuentra dentro de la rama de *medidas mediales* ya que cuantifica la frecuencia con que un nodo actúa como puente a lo largo del camino más corto entre dos nodos. Formalmente, dado un nodo *v,* es la suma de la fracción de todas las parejas de caminos más cortos entre dos nodos que atraviesan v, esto es:

$$\boldsymbol{C_B(v)} = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

  Donde V es el conjunto de nodos del **grafo**, σ(s,t) es el número de caminos (s,t) más cortos y σ(s,t | v) es el número de esos caminos que pasan a través de algún nodo v distinto de s,t. Teniendo en cuenta que si s = t, σ(s,t) =1 y que si v $\in$ *s,t,* σ(s,t | v) = 0, según [19].

- **Centralidad de Vector Propio o "EigenVector":** Propuesta por *Phillip Bonacich en 1972*, mide la influencia de un nodo en la red en base a la **centralidad** de sus vecinos, correspondiéndose con el principal vector propio de la matriz de adyacencia del **grafo** dado. Debido a ello, los nodos más centrales en este tipo de medida se corresponden con los centros de grandes grupos cohesivos, es decir, son nodos conectados a muchos nodos que, a su vez, también tienen un valor alto de esta medida. Este tipo de medida ha sido muy utilizada en redes sociales para difundir información o para medir la relevancia de páginas webs (*Google, PageRank*[12]). Formalmente, la **centralidad** de vector propio de un nodo *i es:*

$$\boldsymbol{Ax} = \boldsymbol{\lambda x}$$

  Donde A es la matriz de adyacencias del **grafo** G con valor de vector propio λ. Gracias al teorema de *Perron-Frobenius*, se puede afirmar que existe una solución positiva y única si λ es el mayor valor de los vectores propios asociados a la matriz de adyacencias A.

[12] https://en.wikipedia.org/wiki/PageRank

- **Centralidad de comunicabilidad o "comunnicability":** También conocida como **centralidad** de ***subgrafo***, es la suma de todos los caminos cerrados que empiezan y terminan en un mismo nodo. Para este tipo de medida de la **centralidad** existen muchas variaciones, para este trabajo se ha implementado aquella que utiliza la descomposición espectral de la matriz de adyacencia propuesta en [20]:

$$\boldsymbol{SC}(\boldsymbol{u}) = \sum_{j=1}^{N} (v_j^u)^2 \, e^{\lambda_j}$$

  Donde v_j es un vector propio de la matriz A del **grafo** G que se corresponde con el valor λ_j.

NOTA: Aparte de estas **centralidades** se intentó implementar una variante de esta última conocida como "*Comunnicability -Betweenness*", la cual mide el número de caminos que conecta cada pareja de nodos como base de la **centralidad** de intermediación. Sin embargo, la implementación obtenida en el módulo *Networkx*[13] (para más información véase el apartado "Etapa Formativa") no era compatible con las listas y diccionarios empleados en los scripts desarrollados de Python.

2.3.3. Métodos de ordenación

Por último y para terminar esta sección, es necesario puntualizar que, para aquellos experimentos en los que se ha utilizado una medida de **centralidad** como método de selección del grupo de subnodos, se decidió añadir un método que ordenase los nodos elegidos. De esta manera se escogerían el porcentaje de nodos elegido en base a estos criterios de ordenación pudiendo observarse además los resultados de actuar sobre aquellos nodos con mayor o menor **centralidad**… Los métodos usados se exponen a continuación:

- **Mayor Centralidad:** Los nodos son ordenados en orden descendente, se escoge aquel porcentaje de puntos de acceso con mayor **centralidad**.
- **Menor Centralidad:** Los nodos son ordenados en orden ascendente, se escoge aquel porcentaje de puntos de acceso con menor **centralidad**.
- **Distribución Normal:** Al conjunto de medidas de la **centralidad** de cada punto de acceso se le aplica la distribución normal del script "*ChosenFollowingNormal*"[14]. Dicho script selecciona un porcentaje determinado de nodos de la campana de Gauss formada.

[13] Para más información véase el apartado "Etapa Formativa".

[14] Para más información véanse los apartados "Funciones" y "Referencias: Código".

3. Experimentos

En esta sección se describirán todos los procesos de aprendizaje y desarrollo que han sido necesarios para poder culminar el lanzamiento de los distintos experimentos, así como su análisis y las conclusiones esperadas y extraídas de los resultados obtenidos.

3.1. Herramientas empleadas

Como en todo proyecto, antes de comenzar el desarrollo es necesario recopilar y asimilar cierta información que permita trabajar con el entorno para el propio proyecto. Por estos motivos, esta fue la primera etapa que conformó las distintas fases de este TFG, siendo, cuanto menos, interesante. Aparte de toda la información expuesta en el anterior apartado de "Teoría Introductoria" acerca de la teoría de **grafos**, comportamiento de los escenarios usados bajo el protocolo 802.11n, diferentes negociadores automáticos y métodos de elección y ordenación del conjunto de subnodos, fue imperativo informarse acerca de las herramientas que ya estaban siendo utilizadas en los scripts de Python, la funcionalidad de cada uno de los distintos scripts, así como del propio lenguaje y otras herramientas que se decidieron usar para facilitar el alcance de los diferentes objetivos que propone este TFG.

En las siguientes subsecciones de este apartado, se explicará la funcionalidad de cada una de estas herramientas.

3.1.1. Python

Lenguaje escogido por los miembros del equipo de investigación y, por tanto, lenguaje empleado en el desarrollo de este trabajo. Creado por *Guido van Rossum,* es interpretado, orientado a objetos, independiente de plataforma, con una sintaxis muy clara y, uno de sus mayores atractivos, con un gran soporte de funciones y librerías que hacen plausible el diseño de múltiples tipos de aplicaciones. En este caso, se ha empleado la versión **2.7** de Python debido a que era la implementada en los ya citados scripts.

3.1.1.1. Scripts Python

Como ya se ha mencionado a lo largo de las diferentes secciones, para el desarrollo del TFG se ha partido de una serie de scripts facilitados por el equipo de investigación junto al que se ha llevado este. El conjunto lo conforman nueve scripts, cada uno con una funcionalidad resumida a continuación:

- ***"analizar_resultados.py":*** Como bien expresa su nombre, este script permite al usuario extraer los resultados de los experimentos lanzados por otro de los scripts. Su funcionalidad inicial tuvo que ser editada y ampliada para poder ajustarse a los objetivos del trabajo.
- ***"ChosenFollowingNormal.py":*** Facilitado por el tutor del trabajo, *David Orden*. Permite, dada una lista, aplicar una distribución de *Gauss* a esta y obtener una lista de salida con el resultado de dicha aplicación. No fue necesario aplicar ningún cambio.
- ***"división_methods.py":*** Conjunto de funciones que permiten dividir o hacer un reparto de nodos entre los proveedores existentes. Se contempla el uso de dos proveedores y una asignación de puntos de acceso aleatoria y, cuatro proveedores junto con una asignación de puntos de acceso por cuadrantes. No fue necesario aplicar ningún cambio.
- ***"launch_experiment.py":*** Como su propio nombre indica, se trata del script que generar y ejecuta los experimentos. De nuevo, se realizaron algunas modificaciones

que permitieron implementar la llamada a las funciones que aportaban las **restricciones** citadas en los anteriores apartados, entre otras.

- ***"problem_solvers.py":*** Contiene los diferentes mecanismos y técnicas de decisión de los negociadores automáticos que han sido empleados en este TFG, así como otros. Se realizaron modificaciones en aquellos empleados en el trabajo para poder implementar la **restricción** de no modificación de frecuencias en algunos nodos.
- ***"wifi_utility.py":*** Conjunto de funciones que cubren distintas utilidades necesarias en los **grafos** usados, desde las funciones creadas para el **coloreado** de **grafos**, hasta aquellas que calculan la utilidad del conjunto de nodos y aquellas que calculan tanto la potencia como las pérdidas.
- ***"wifigraphs.py":*** Compuesto por funciones que permiten la lectura de **grafos** y que obtienen su tipo y las posiciones de los nodos que lo componen. No fue necesario aplicar ningún cambio.
- ***"wifigraphs_parameters.py":*** Contiene una serie de variables que permiten generar y/o seleccionar un **grafo**. No fue necesario aplicar ningún cambio.

3.1.2. Networkx

Se trata de un Framework o módulo de Python que permite crear y manipular **grafos** orientados a redes. Ha jugado un papel esencial en este TFG, principalmente porque venía siendo utilizado por el equipo de investigación para el estudio de redes y por las facilidades que ha aportado en el cálculo y diseño de la funcionalidad implementada. Esta librería contempla el uso de distintos tipos de **grafo**: unidireccionales simples (**grafos** no dirigidos que pueden permitir la existencia de bucles), dirigidos o **digrafos** (las aristas tienen un sentido definido), **multigrafos** (**grafo** no dirigido que puede tener más de una arista entre dos nodos) y **dimultigrafos**. Para el caso que aquí se tiene, interesa hacer énfasis en el primer tipo de estos **grafos**, los unidireccionales simples, ya que son los empleados por el equipo de investigación y, en consecuencia, los empleados por el autor. Este tipo de **grafos** tienen una clase definida en la librería conocida por el nombre de "*Graph*", para iniciar una estructura vacía (sin vértices ni aristas), se emplea la siguiente llamada:

```
>>> G = nx.Graph()
```

Cuadro de Texto 1 - Creación de grafo en NetworkX.

Donde *G* es el **grafo** a construir y "*nx*" el módulo Networkx abreviado con una llamada "*import networkx as nx*".

Por otro lado, también ofrece funcionalidad para añadir vértices y aristas al **grafo** a partir de diferentes instancias como, por ejemplo: una lista, un diccionario, un archivo u otro **grafo**.

```
>>> G.add_nodes_from([2,3])
>>> G.add_nodes_from(range(100,110))
>>> H=nx.Graph()
>>> H.add_path([0,1,2,3,4,5,6,7,8,9])
>>> G.add_nodes_from(H)
>>> G.add_edge(1, 2)
>>> G.add_edges_from([(1,2),(1,3)])
>>> G.add_edges_from(H.edges())
```

Cuadro de Texto 2 – Ejemplos inserción aristas y vértices.

Por otra parte, permite añadir atributos a los nodos, esta cualidad ha sido vital en el desarrollo de los scripts puesto que ha permitido, asignar a cada nodo atributos como el color de cada nodo, los clientes asociados (en caso de ser un punto de acceso), la distancia, la pérdida de potencia… En resumen, se trata de una característica que abre un abanico completo de posibilidades de cara a caracterizar los nodos que componen la red sobre la que se opera. Por ende, esta utilidad también se extiende a las aristas, permitiendo otorgarles color y/o peso, entre otros…

```
>>> G.add_node(1, color=1)
>>> G.add_nodes_from([3], Ploss=0.25)
>>> G.add_edges_from([(1,2,{'color':'blue'}), (2,3,{'weight':8})])
>>> G[1][2]['weight'] = 4.7
```

Cuadro de Texto 3 – Ejemplos de asignación de atributos.

Adicionalmente y para finalizar este apartado, cabe mencionar que la clase *"Graph"* utiliza unas estructuras de datos conocidas como *"dict-of-dict-of-dict"*, compuestas del siguiente modo: el diccionario exterior o diccionario de nodos, contiene listas de adyacencia cuya llave son los nodos; el diccionario intermedio o diccionario de adyacencia, contiene listas de adyacencia cuya llave son los vecinos; por último, el diccionario interior o diccionario de los atributos de las aristas, representa los datos contenidos en cada arista siendo su llave los nombres de los atributos. Cualquiera de estos diccionarios puede ser reemplazado por uno generado por el usuario, en el caso que ocupa este trabajo, no ha sido necesario. Este tipo de estructuras de datos facilita mucho el acceso a estos, ya que están optimizados para ello.

3.1.3. XlsxWriter

Otra de las herramientas empleadas fue este FrameWork de Python, esta vez, con el objetivo de facilitar y optimizar el trabajo a realizar a la hora de pasar los resultados extraídos por el script *"analizar_resultados.py"* a las tablas de Excel. Su principal funcionalidad es la de permitir, a través de Python, generar documentos de tipo "xlsx" con fórmulas, números, hiperlinks… Entre las ventajas con las que cuenta cabe destacar su simplicidad, la compatibilidad con otros módulos de Python para documentos de Excel, el alto volumen de ejemplos y documentación disponibles y, para terminar, el alto grado de eficiencia que tiene, puesto que emplea muy poca memoria incluso en archivos de gran extensión. Por el contrario, tiene una gran desventaja y es que no permite la lectura o modificación de los archivos de Excel, sin embargo, dicha utilidad no es necesaria en este caso, así que se consideró un candidato perfecto.

Una vez más, el abanico de posibilidades que ofrece este módulo es muy abierto, por ello, tan solo se ha extraído información referente a aquella funcionalidad necesaria para optimizar el trabajo antes citado, esto es: cómo generar un archivo de Excel, cómo darle un determinado formato y cómo añadir distintos tipos de datos a este. A continuación, se muestra cómo generar un documento de Excel básico con nombre "analise_results.xlsx", compuesto por una hoja llamada "Data" y con dos columnas relativas al nombre de un **grafo** y el valor de su orden:

```
>>> import xlsxwriter
>>> workbook = xlsxwriter.Workbook('analise_results.xlsx')
>>>worksheet = workbook.add_worksheet('Data')
>>>data = (['EPS-0.25-1',0.2014],['EPS-0.5-1', 0.123],[' EPS-1-1', 0.151])
>>>row = 0
>>>col = 0
>>>for graph, utility in (data):
>>>        worksheet.write(row, col, graph)
>>>        worksheet.write(row, col + 1, utility)
>>>        row += 1
>>>workbook.close()
```

Cuadro de Texto 4 - Ejemplo de creación documento "xlsx".

Como se puede observar, tras importar el módulo, este permite operar sobre las clases *"workbook" y "worksheet"* para generar el documento deseado. Una vez realizado esto es posible escribir datos directamente sobre la hoja de datos sobre la que se esté trabajando. Para acabar con este apartado, se muestra un ejemplo de cómo dar formato a las celdas, cómo emplear fórmulas y distintos tipos de notación para hacer referencia a una celda:

```
>>> import xlsxwriter
>>> workbook = xlsxwriter.Workbook('analise_results.xlsx')
>>>worksheet = workbook.add_worksheet('Data2')
>>>bold = workbook.add_format({'bold': True})
>>>utility = workbook.add_format({'num_format': '#.##0'})
>>>worksheet.write('A1', 'Graph', bold)
>>>worksheet.write('B1', 'Utility', bold)
>>>data = (['EPS-0.25-1',0.2014],['EPS-0.5-1', 0.123],[' EPS-1-1', 0.151])
>>>row = 0
>>>col = 0
>>>for graph, utility in (data):
>>>        worksheet.write(row, col, graph)
>>>        worksheet.write(row, col + 1, utility)
>>>         row += 1
>>> worksheet.write(row, 0, 'Total',      bold)
>>> worksheet.write(row, 1, '=SUM(B2:B1)', utility)
>>>workbook.close()
```

Cuadro de Texto 5 – Ejemplo de creación documento "xlsx" con formato.

Como se puede observar, se procede de manera parecida al anterior cuadro de texto con la excepción de que se generan dos formatos, uno que permite poner los datos de una celda en negrita, y otro, que permite da un formato a aquellos datos con formato numérico. Se obliga también a que las dos primeras celdas de las dos primeras columnas de la hoja "*Data2*" contengan las cadenas "*Graph*" y "*Utility*" respectivamente. Para finalizar, se genera una última celda fuera del bucle que permite reflejar el resultado de sumar las utilidades de cada una de las celdas con un valor de utilidad.

3.1.4. PyOpenCL

Para acabar con el apartado de las herramientas usadas, se ha querido mencionar una herramienta con la cual se han investigado distintas modificaciones cuyo objetivo era optimizar, en gran medida, el tiempo de ejecución de los experimentos, pero que no ha llegado a implementarse en la versión definitiva del proyecto puesto que no se llegó a unos

resultados concluyentes. Se trata de otro módulo o FrameWork para Python que permite implementar programación *en paralelo* o sobre *GPU* en este lenguaje, realmente, podría decirse que se trata de un lenguaje independiente que trabaja sobre Python.

De nuevo, se trata de una herramienta con múltiples funcionalidades, por lo tanto, en este apartado, tan solo se expondrá una pequeña parte de estas, en concreto, aquella que brindaría utilidad a este TFG.

```
import numpy as np
import pyopencl as cl
from time import time

kernelsource = """
__kernel void square ( __global float* input, __global float* output)
{
  unsigned int i = get_global_id(0);
  output[i] = input[i] * input[i] ;
}
"""
ctx = cl.create_some_context ()
queue = cl.CommandQueue (ctx)
program = cl.Program(ctx, kernelsource).build()

h_input1 = np.arange(30000000).astype(np.float32)
h_output = np.empty(30000000).astype(np.float32)

mf = cl.mem_flags

d_input1 = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=h_input1)
d_output = cl.Buffer(ctx, mf.WRITE_ONLY, h_output.nbytes)

timer = time()
program.square(queue, h_input1.shape, None, d_input1, d_output)
queue.finish()
timer = time() – timer

print "The kernel ran in", timer, "seconds"

cl.enqueue_copy(queue, h_output, d_output)

print h_output

i = 0

timer = time()

while i < len(h_output):
  h_output[i] = h_input1[i] * h_input1[i] / 8
  i+=1
timer = time() – timer

print h_output
print "The seq prog. ran in", timer, "seconds”
```

Cuadro de Texto 6 – Ejemplo de Python con un kernel de PyOpenCL.

El objetivo de este script de prueba es el de llevar a cabo una operación matemática, el cuadrado de un número, sobre 30 millones de números y permitir observar la diferencia, en tiempo de ejecución, de los cálculos realizados sobre CPU (secuencialmente) y sobre GPU (en paralelo). Para comenzar, se necesita crear un *kernel* (sentencias de código a ser ejecutadas por cada hilo generado, dentro de la GPU o *device*), en este caso, el *kernel* tiene dos variables *globales* (dentro de este "lenguaje" existen varios niveles de memoria en función de qué objetos pueden acceder a ella, estos son: *memoria privada* – solo para datos generados por el propio hilo; *memoria local* – para datos compartidos entre hilos de un mismo bloque; memoria global – para datos compartidos entre bloques de hilos; y *memoria del host* – memoria de CPU.) para la entrada y la salida de datos, en resumen, el kernel obtiene el identificador del hilo que lo está ejecutando y utiliza dicho identificador para rellenar los datos del array de salida a partir del cuadro del valor para dicho identificador en el array de entrada. Para declarar variables y distinguir de manera rápida cuáles serán utilizadas en el *host o* en el *device* se utiliza una convención que consiste en iniciar los nombres de las variables con *h* o *d*. Así pues, se genera el contexto necesario para cargar el *kernel* en la GPU, se declaran las variables en el host y se reserva memoria en *GPU* para las variables que vaya a emplear el *kernel (*PyOpenCL permite caracterizar estos *buffers* con una multitud de atributos: tamaño, lectura/escritura, sentido de la lectura/escritura, tipo…). Tras ello, se lanza el *kernel,* se encola en la cola de tareas para GPU y, tras acabar se copian los datos del array de resultados en la correspondiente variable del *host* para poder ser impresos y comparados con los resultados de *CPU.* Para acabar, se lanza el programa sobre *CPU* y se comparan los tiempos de ejecución de ambos, así como los resultados. En la siguiente imagen se puede observar la salida obtenida:

```
In [2]: runfile('C:/Users/Usuario/Desktop/python/prueba.py', wdir='C:/
Users/Usuario/Desktop/python')
The kernel ran in 0.446000099182 seconds



The seq prog. ran in 126.819000006 seconds
```

Imagen 7 – Salida Script ejemplo PyOpenCL

Como se puede observar, la diferencia entre los tiempos de ejecución es abismal, del orden de centenas de segundos sobre la ejecución en *GPU.* Este fue uno de los principales motivos por los que se intentó implementar y adaptar esta herramienta en este TFG.

3.2. Planificación

Esta sección está dedicada a mostrar el proceso de bosquejo previo al desarrollo de este trabajo, siendo clave en un inicio para permitir al autor hacerse una idea del tiempo dedicado a cada etapa del trabajo. Se han distinguido cinco etapas funcionales, cada una de ellas con sub-etapas indicadas entre paréntesis:

- **Planificación:** En esta etapa se pretende prever el tiempo aproximado que se dedicará a cada una de las distintas etapas para obtener una idea general de cómo se va a desarrollar el trabajo.
- **Investigación:** Etapa que hace referencia a aquel periodo de tiempo dedicado al estudio y/o ampliación de los conocimientos necesarios y requeridos para poder comenzar el diseño y desarrollo de este TFG (búsqueda, estudio y aplicación de contenidos).
- **Diseño y Desarrollo:** Periodo temporal dedicado a la generación de código y testado de este que permita alcanzar los objetivos propuestos al inicio de este TFG. Incluye posibles modificaciones de este a lo largo de otras etapas (diseño de versiones, testado y experimentos).
- **Análisis:** Etapa dedicada a la extracción de conclusiones de los resultados obtenidos a través del código desarrollado (resultados esperados, procesado y conclusiones).
- **Documentación:** Etapa final cuyo objetivo es el de reflejar, en una memoria escrita, los resultados y conclusiones de las anteriores etapas (teoría, resultados y otros).

A su vez, se mostrará una comparativa mediante diagramas de Gantt que permita ver las diferencias entre el tiempo estimado y el tiempo real empleado para cada una de las diferentes etapas.

3.3. Duración

En este apartado se pretenden reflejar las posibles diferencias entre el tiempo "inicial" estimado y el tiempo real que ha costado el desarrollo de este proyecto.

3.3.1. Tiempo estimado

Prever el tiempo que va a tomar el completar un proyecto es difícil ya que no se dispone, desde el inicio, de toda la información necesaria. Por ello, normalmente, los tiempos reales empleados suelen dilatarse en comparativa con los tiempos estimados debido a errores no previstos, modificaciones… En el siguiente diagrama de Gantt, se puede observar cómo se estimó inicialmente los días requeridos para cada tarea o fase. En un principio, se estableció llevar a cabo reuniones/informes periódicos para mantener al día a los miembros del equipo acerca de los avances o las dificultades halladas.

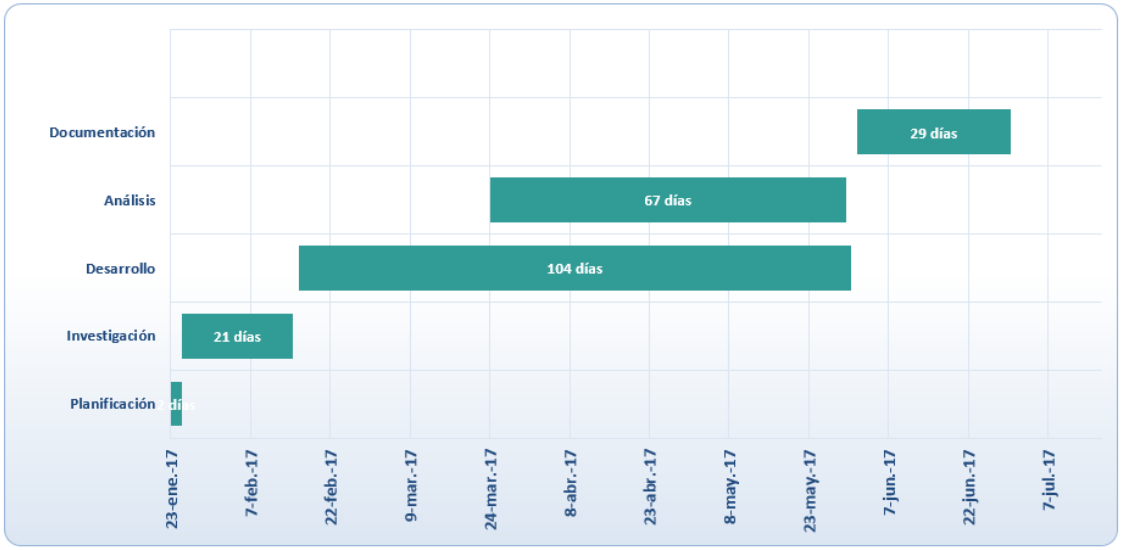


Imagen 8 - Diagrama de Gantt tiempo estimado.

3.3.2. Tiempo real

Como se había previsto, los tiempos iniciales se han visto dilatados, en su mayoría. Principalmente, debido a la superposición de tareas ya que mientras se iba desarrollando el proyecto, se comenzaron a analizar los primeros datos, se fueron realizando cambios y adaptaciones que facilitasen el análisis de futuros datos… También y, a pesar de que la fase de investigación quedó relegada al inicio del proyecto, esta ha tenido un papel vital a lo largo de todo el desarrollo, tanto para entender algunos de los conceptos empleados como para ampliar, por ejemplo, los tipos de centralidades usados. Por otro lado, la labor de documentación ha sido increíble puesto que el volumen de datos recogido también ha sido inmenso. Para finalizar, no se han podido llevar a cabo todas las reuniones que se pretendió en un principio, debido a viajes de los componentes del equipo de investigación, otros deberes académicos, etc. pero sí que se ha mantenido el objetivo de mantener informadas a las partes mediante correos cada vez que se daba un cambio trascendental en el trabajo o se obtenía un nuevo conjunto de resultados.

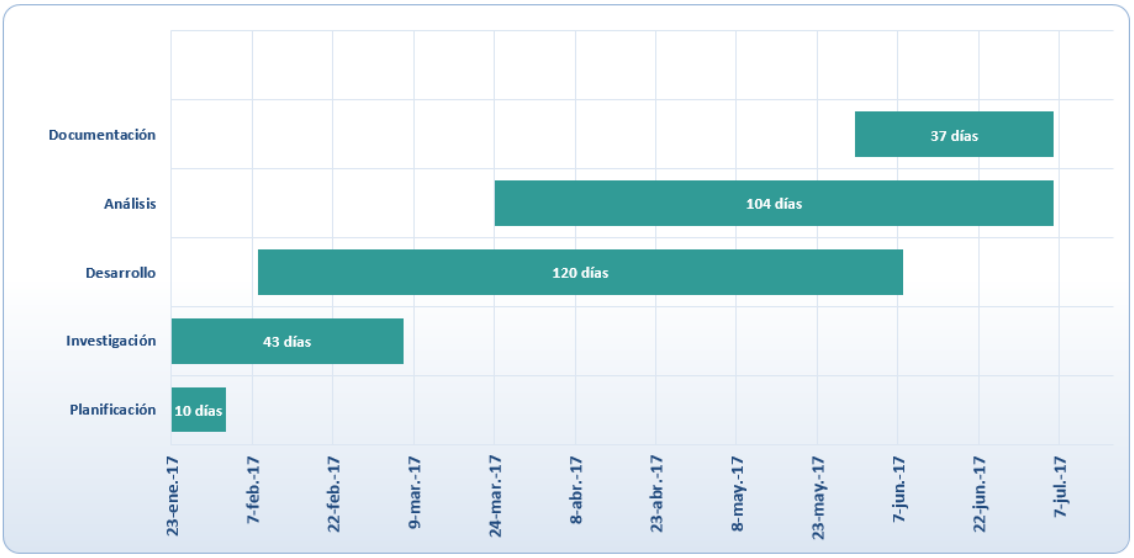


Imagen 9 - Diagrama de Gantt tiempo real.

3.4. Diseño y Desarrollo

Junto a la etapa de investigación y análisis, conforma uno de los periodos más importantes para este trabajo ya que de él dependerán, directamente, los resultados de este. Con el objetivo de explicar este proceso, se detallará todo el código generado para este supuesto, tanto las funciones creadas de cero, como las modificaciones llevadas a cabo en los scripts ya existentes. Por otro lado, también se explicarán las distintas versiones/mejoras del código que han sido generadas en el periodo de tiempo que ha durado esta fase.

Cabe destacar que esta fase ha sido la más empírica de todas debido a que, en muchos casos, las mejoras obtenidas de una versión a otra han sido resultado de la necesidad de automatizar un proceso, de tener información en caso de un experimento fallase, organización y utilidad a la hora de generar los experimentos, es decir, se han generado retroalimentándose con los resultados/fallos obtenidos del código hasta ese momento desarrollado.

Para acabar, el periodo de duración de esta fase ha sido uno de los más duraderos puesto que se ha desarrollado de manera paralela a otras fases como las de *Análisis y Documentación* y, además, a ojos del autor, siempre cabía la posibilidad de llevar a cabo una pequeña mejora en alguna parte del código.

3.4.1. Funciones

En este apartado se expondrá la funcionalidad de cada una de las funciones generadas[15] así como los argumentos necesarios para ejecutarlas, su significado, tipo y valor devuelto. Cada función será detallada en una página independiente en la cuál será posible observar:

- ***Nombre de la función.***
- ***Prototipo de la función.***
- ***Descripción funcional.***
- ***Tabla detallada para los argumentos.***
- ***Valor retornado y tipo.***

Se ha procurado asignar una funcionalidad única a cada una de estas funciones, de manera que cada una de ellas desempeñe una utilidad concreta dentro del código y sea fácil entender la secuencia de ejecución de estas al lanzar los experimentos.

[15] Para consultar el código de cualquiera de las funciones aquí referenciadas, consúltese el apartado "*Referencias->Código".*

3.4.1.1. Node_selection

```
def node_selection (I,apList,node_sel_mode,percentage,nodeSortMethod):
```

Cuadro de Texto 7 – Declaración función "node_selection".

El objetivo de esta función es del de permitir aplicar la **restricción** de las frecuencias fijas comentada anteriormente, a un subconjunto de nodos *V1⊆V del* ***grafo*** *G(V,E)*, generando este subconjunto en base a una serie de factores caracterizados por los argumentos que toma la función. Esta función llama a la función "*order_nodes*".

Parámetros:

Parámetro	*Tipo*	*Valor por defecto*	*Descripción*
I	*Networkx Graph*	*	**Grafo** de interferencias.
apList	*Lista*	*	Puntos de acceso pertenecientes al **grafo** I.
node_sel_mode	*Int*	*	Modo de selección de los puntos de acceso que formarán parte del conjunto de subnodos, se han implementado cinco modos: ***0:*** *Elección aleatoria.* ***1: Centralidad*** *de Grado.* ***2: Centralidad*** *de Cercanía.* ***3: Centralidad*** *de Intermediación.* ***4: Centralidad*** *de Vector Propio.* **5: *Centralidad*** *de Comunicabilidad.*
Percentage	*%*	*	Porcentaje de nodos, respecto al total del conjunto V del **grafo** G, que conformarán el subconjunto.
NodeSortMethod	*Int*	*	Método de ordenación en caso de seleccionar un modo de selección que aplique una **centralidad** (véase la función "*order_nodes*" para conocer los métodos implementados).

Tabla 3 – Parámetros de la función "node_selection". *(*) Parámetro obligatorio.*

Retorno/output

Lista de nodos ordenados y elegidos en base a los parámetros introducidos.

3.4.1.2. Order_nodes

```
def order_nodes (Nodes,apList,nap,nodeSortMethod)
```

Cuadro de Texto 8 – Declaración función "order_nodes".

Dado un diccionario de nodos, permite generar una lista de nodos ordenados en base a un método elegido por el usuario, tomando como valor de ordenación la **centralidad** de cada uno de los nodos (solo tiene en cuenta aquellos nodos que son puntos de acceso).

Parámetros:

Parámetro	*Tipo*	*Valor por defecto*	*Descripción*
Nodes	*Diccionario*	*	Diccionario de nodos con la **centralidad** como valor.
Nap	*Float*	*	Número de puntos de acceso ordenados a devolver.
nodeSortMethod	*Int*	*	Método de ordenación: ***0:*** *Más centrales (desc.).* ***1:*** *Menos centrales (ascd.).* ***2:*** *Dist.Normal.*

Tabla 4 – Parámetros de la función "order_nodes". *(*) Parámetro obligatorio.*

Retorno/output

Lista de nodos ordenados.

3.4.1.3. negotiator_selection

```
def negotiator_selection (**kwargs)
```

Cuadro de Texto 9 – Declaración función "order_nodes".

Esta función permite aplicar alguno de los negociadores ya mencionados en las anteriores secciones a los nodos de un determinado **grafo,** en el caso que contempla este TFG, serán necesarios dos **grafos,** el **grafo** de asociación G y el de interferencias I.

Parámetros:

Parámetro	*Tipo*	*Valor por defecto*	*Descripción*
G	*Networkx Graph*	*	**Grafo** de asociación.
I	*Networkx Graph*	*	**Grafo** de interferencias.
apList	*Lista*	*	Lista de puntos de acceso.
pList	*Lista*	*	Lista de proveedores.
nColors	*Int*	*	Número máximo de colores.
colorList	*Lista*	*	Lista con los colores asignados a cada punto de acceso de la "apList".
sub_apList	*Lista*	*	Lista con los puntos de acceso que conforman el subconjunto de nodos con frecuencias o colores fijos.
negotiatorSel_Mode	*Int*	*	Mecanismo de negociación: ***0:*** *Random.* ***1:*** *Simulated_Annealing.* ***2:*** *Hill_Climber.* ***3:*** *LCCS.*
Iterations	*Int*	***None***	Número de iteraciones para el mecanismo negociador (opcional).
Temperature	*Float*	***None***	Parámetro que define el comportamiento de SA (opcional).
Overlap	*Bool*	***None***	Parámetro que define el comportamiento de LCCS (opcional).

Tabla 5 – Parámetros de la función "negotiator_selection". *(*) Parámetro obligatorio.*

Retorno/output

Lista de los colores/frecuencias de cada punto de acceso del **grafo** G.

3.4.1.4. colorList_Generator

```
def colorList_Generator (iterations=None,temperature=None,overlap=None,**kwargs)
```

Cuadro de Texto 10 – Declaración función "colorList_Generator".

Finalmente, esta es la función encargada de aplicar la **restricción** de usar un solo color para todo el conjunto de subnodos o de aplicar colores aleatorios a estos, es decir, genera la lista de colores/frecuencias **iniciales** de manera aleatoria para todos los puntos de acceso y luego modifica aquellas frecuencias que pertenezcan a los puntos de acceso del subconjunto de nodos con frecuencias/canales fijos.

Parámetros:

Parámetro	*Tipo*	*Valor por defecto*	*Descripción*
G	*Networkx Graph*	*	**Grafo** de asociación.
I	*Networkx Graph*	*	**Grafo** de interferencias.
apList	*Lista*	*	Lista de puntos de acceso.
pList	*Lista*	*	Lista de proveedores.
nColors	*Int*	*	Número máximo de colores.
colorSel_Mode	*Int*	*	**Restricción** de color: ***0:*** *Color único.* ***1:*** *Colores Aleatorios.* ***2:*** *DSATUR.*
fixedColorVal	*Int*	***None***	Valor elegido cuando se aplica la **restricción** de color único.
sub_apList	*Lista*	*	Lista con los puntos de acceso que conforman el subconjunto de nodos con frecuencias o colores fijos.
negotiatorSel_Mode	*Int*	*	Mecanismo de negociación: ***0:*** *Random.* ***1:*** *Simulated_Annealing.* ***2:*** *Hill_Climber.* ***3:*** *LCCS.*
Iterations	*Int*	***None***	Número de iteraciones para el mecanismo negociador (opcional).
Temperature	*Float*	***None***	Parámetro que define el comportamiento de SA (opcional).
Overlap	*Bool*	***None***	Parámetro que define el comportamiento de LCCS (opcional).

Tabla 6 – Parámetros de la función "colorlist_generator". *(*) Parámetro obligatorio.*

Retorno/output

Lista de los colores/frecuencias de cada punto de acceso del **grafo** G, emplea el mismo valor devuelto por "*negotiator_selection*".

3.4.2. Versiones

A lo largo del desarrollo de una aplicación Software esta atraviesa una serie de etapas o fases, dentro de las cuales, se generan distintas versiones de la aplicación. Para este trabajo se han distinguido tres versiones basadas en las "*fases de desarrollo de software*" existentes en una aplicación orientada a usuarios: versión *Alfa*, versión *Beta* y versión de *disponibilidad general o RTM.*

3.4.2.1. Versión *Alfa*

Se trata de una primera versión de la aplicación, es inestable puesto que aún no se han contemplado todos los posibles errores ni se ha implementado la totalidad de la funcionalidad requerida. En la mayoría de casos, incluido este trabajo, se utiliza para desarrollar la base de la aplicación con una funcionalidad básica o mínima necesaria, entender el funcionamiento del conjunto de la aplicación y plantear mejoras para alcanzar la versión *Beta.* Esta versión sufrió tres revisiones, al ser modificaciones menores no se dedica un apartado para cada una de ellas, sino que se detallan en los siguientes incisos:

- ***Alfa 0.1:*** En primera instancia y para conocer el funcionamiento de todos los scripts cedidos por el equipo de investigación, se optó por generar una única función adaptada a las llamadas ya usadas en dichos scripts, que eligiera el conjunto de subnodos, les diese un color fijo y aplicara el mediador aleatorio (funcionalidad básica).
- ***Alfa 0.2*:** Una vez comprendido y comprobado el funcionamiento de la primera revisión, se prefirió escindir la funcionalidad en las funciones del anterior apartado, usando argumentos fijos.
- ***Alfa 0.3*:** Para finalizar esta versión, debido al gran número de parámetros que recibían algunas funciones, se optó por usar el argumento "***kwargs*" de Python ya que proporciona utilidades para pasar argumentos con un nombre determinado y argumentos asociados a una llave o *key*. Además de ello, se corrigieron bugs a la hora de generar colores aleatorios, usando el método "*random.randrange()*".

Puesto que solo se había implementado la técnica de *Referencia Aleatoria* (cada punto de acceso escoge un canal de manera aleatoria), se generó una salida de resultados adaptada a esta técnica. En el cuadro de texto de la derecha, se puede observar la parte de la salida que hace referencia a la lectura del fichero de datos, se usó para comprobar el **grafo** usado, el número de repeticiones empleado… En versiones posteriores se ocultó esta salida para evitar sobrecargar la salida de la terminal.

```
Utilizando grafo:
./grafosEPS/EPS-1-3-1
ccollections
OrderedDict
p0
((lp1
(lp2
S'Graph'
p3
aS'EPS-1-3-1'
p4
aa(lp5
S'Div_Meth_t '
p6
aF0.1128349304199218
aa(lp7
S'Run_t'
p8
aF0.007044541835784912
aatp9
Rp20
```

Cuadro de Texto 11 – Salida 1 versión Alfa 0.3.

Por otro lado, también se imprimieron algunos parámetros necesarios para el posterior análisis, en futuras versiones el número de parámetros devueltos se verá reducido, por ejemplo, se dejará de emplear la utilidad no normalizada:

Average time per run(s): 0.00188397169113

Total Time per experiment(s): 0.0376794338226

Orden del grafo: 1324

Media Utilidades: 199.998604328

Utilidad Normalizada: 0.15105634768

Desviación Estándar: 0.0310479136199

Fujita Fairness Avg: 0.0686323274546

Fujita Fairness STD: 0.0132795190133

Total Experiment Time-Lapse(s): 0.37195110321

Cuadro de Texto 12 – Salida 2 versión Alfa 0.3.

3.4.2.2. Versión *Beta*

En este nivel, el prototipo de la aplicación ya puede ser considerado como una versión funcional de la aplicación final. Por lo general, esta etapa ha de atravesar una serie de revisiones o testados que permitan detectar fallos, ampliar funcionalidad y llevar a cabo las modificaciones necesarias para alcanzar la versión final deseada. El objetivo para esta versión estuvo orientado a la **optimización** de los procesos llevados a cabo, implementación de la funcionalidad necesaria para operar con negociadores distintos de la técnica Random, generar tablas de resultados y automatizar este último proceso. Para alcanzar la *Beta* final, se realizaron 6 revisiones de la misma, las cuales, son expuestas a continuación.

3.4.2.2.1. Beta 0.1

Corregido bug al obtener el número de puntos de acceso que conforman el conjunto de subnodos, el método anterior hacía una conversión explícita de la variable de tipo *Float* a *Int*, ahora se permite truncar o redondear el valor de la variable *Float* de manera que si se tienen 28 puntos de acceso y se requiere que el 10% de estos tengan un canal fijo, se contemple que dos de ellos conformen el subconjunto (truncado) o tres de ellos lo conformen (redondeo). Por otro lado, se implementaron tres **centralidades** (*Closenness, Betweenness y Degree)* para la elección de los nodos y dos métodos de ordenación (descendente y ascendente). Asimismo, se generó el primer prototipo de tabla de resultados en Excel[16].

Como se puede observar se procuró distinguir las posibles variaciones del escenario de un **grafo** en una misma línea y se agruparon los posibles casos en función de la técnica de negociación empleada, el método de asignación de color, etc.

[16] Anexo: **¡Error! No se encuentra el origen de la referencia.**

3.4.2.2.2. Beta 0.2

Esta revisión estuvo orientada a la implementación y **optimización** de los métodos de ordenación del conjunto de subnodos en caso de emplearse una **centralidad** para seleccionarlos. Así pues, se añadió el nuevo script "*ChosenFollowingNormal.py*", que permitió la implementación del último método de ordenación contemplado en este trabajo, el ordenamiento mediante una *Distribución Normal*; y se optimizaron los métodos de ordenación ascendente y descendente mediante expresiones *Lambda*[17], reduciendo la diferencia de ambos métodos a la siguiente sentencia:

```
#Ascendente
auxapList.sort(key=lambda x: x[1])
#Descendente
auxapList.sort(key=lambda x: x[1], reverse=True)
```

Cuadro de Texto 13 – Optimización de los métodos de ordenación mediante expresiones Lambda.

3.4.2.2.3. Beta 0.3

Se modificó la llamada a la función "*colorList_Generator*" de manera que permitiese el paso por parámetro de todos los modos y métodos hasta ahora implementados (**centralidad**, ordenación, condición de color, color…). Esto evitó que el usuario de la aplicación se siguiese viendo obligado a editar los scripts correspondientes para generar el experimento deseado, estableciendo la edición de un único script, "*launch_experiment.py*", desde el cual se podían establecer todas las características deseadas para el experimento.

```
"""COLORING AND NEGOTIATORS PARAMETERS"""
#Node Selection Mode --> [0:Random, 1:Degree Centrality, 2:Closeness Cent., 3:Betweeness Cent.]
nodeSelMode = 4
#Node percentage to have a fixed color, set to 0 if not needed
nodePercentage = 0.3
#Color Selection Mode --> [0:Fixed Color, 1:Random Color, 2:TSC_DSATUR(NI)]
colorSelMode = 0
#Fixed Color Value (use in case of Fixed Color mode selected, contemplated values [1-11])
fixedColor_Val = 11
#Negotiator Selection Mode --> [0:Random, 1:Simmulated_Annealing(NI), 2:Hill_Climber(NI),
3:LCCS(NI)]
negotiatorSelMode = 0
#Sorting Method (not available/usable for random selection node mode) --> [0:More Central
Nodes(Descending Order), 1:Less central(Ascending), 2:Normal Dist.]
nodeSortMethod = 2
```

Cuadro de Texto 14 – Sección de código en launch_experiment.py para la configuración de los experimentos.

[17] Para más información consulte: http://www.secnetix.de/olli/Python/lambda_functions.hawk

Por otro lado, se detectó que en el script "*analizar_resultados.py*" se había estado usando, en algunos casos, un valor erróneo para el número de repeticiones (10 en vez de 20). Se corrigió dicho fallo y se relanzaron todos los experimentos que habían sido afectados por dicho error.

3.4.2.2.4. Beta 0.4

Esta es una de las revisiones clave en el desarrollo de la aplicación, puesto que permitió optimizar los tiempos de ejecución de esta y corregir un error de cálculo en los experimentos lanzados hasta el momento. Para el primero de estos objetivos, se eliminó el cálculo de la **centralidad** para el **grafo** dado en cada iteración del proceso y se extrajo su cálculo a una sentencia previa del código, antes de comenzar las iteraciones, esto permitió reducir tanto el coste computacional, como el tiempo total de ejecución de cada experimento en una medida apreciable. Para acabar, hasta la presente revisión, se generaba un conjunto de subnodos diferente en cada iteración del proceso (20 iteraciones en total), sin embargo, se estableció que, para poder comparar de manera fidedigna los resultados obtenidos entre sí, era necesario establecer que el conjunto de subnodos fuese el mismo en todas las iteraciones.

Una vez corregidos estos errores, se tuvieron que volver a lanzar los experimentos lanzados hasta la fecha de manera que se ajustasen a las nuevas características del código.

3.4.2.2.5. Beta 0.5

Junto a la revisión anterior, se trata de la revisión más esencial en el desarrollo de los experimentos puesto que permitió comenzar a probar experimentos con todas las técnicas de negociación citadas en el apartado "*Negociadores Automáticos*". El primer candidato para implementar la nueva funcionalidad fue "*Hill Climber*" ya que era el mecanismo de negociación con menor volumen de código, más adelante se implementó "*Simulated Annealing*" debido a la similitud entre el código del primero y el segundo y, finalmente, se operó sobre el código de "*LCCS*". Gracias a las facilidades que aporta un lenguaje como Python, las modificaciones en el script "*problem_solvers.py*" que hubieron de realizarse, fueron mínimas, llegando a necesitarse tan solo entre 3-4 sentencias de código nuevas para implementar toda la funcionalidad requerida (no operar sobre los nodos pertenecientes al subconjunto). v.g.: para detectar si el nodo puede cambiar su frecuencia, tan solo fue necesario lo siguiente:

```
if nodeToMutate in sub_apList:
        ##print "Node %d in sub_apList"%nodeToMutate
        continue
```

Cuadro de Texto 15 – Ejemplo de simplicidad en Python.

Con dichas sentencias lo que se consigue es detectar si el nodo a mutar (aquel que va a cambiar la frecuencia/canal/color) pertenece al subconjunto de nodos, en caso afirmativo, se pasa a la siguiente iteración del bucle, evitando las operaciones que se realizarían en caso negativo.

Además de este avance, esta revisión también se centró en la revisión los cálculos de las medidas obtenidas para analizar los resultados. Lamentablemente, se detectaron errores en el cálculo de la desviación estándar de la utilidad asociados a un parámetro de la función de

cálculo de la desviación conocido como "*Delta Degrees of Freedom*". Dichos errores fueron subsanados y se pudo dar paso a la siguiente revisión.

3.4.2.2.6. Beta 0.6

Esta última revisión de la fase de desarrollo *Beta* procuró asegurar que todos los cálculos y operaciones se realizasen de manera correcta y óptima para evitar tener que repetir experimentos a partir de este punto.

Aparte, se implementó una nueva **centralidad**, la **centralidad** *EigenVector* y se modificó tanto la salida del script "*analizar_resultados.py*" como el formato de las tablas de resultados para tener en cuenta los tiempos medios de ejecución de cada experimento y el tiempo total de ejecución del conjunto de experimentos lanzados, así como todas las medidas y ajustes llevados a cabo en anteriores revisiones.

Norm.Utility: 0.15105634768

Utility STD: 0.0310479136199

Fujita Fairness Avg: 0.0686323274546

Fujita Fairness STD: 0.0132795190133

Average time per run(s): 0.00188397169113

Total Experiment Time (s): 0.390965461731

Cuadro de Texto 16 – Salida versión Beta 0.6.

	1				
	U_Norm	STD	Fairness Fujita		Time(s)
	-	-	Avg	STD	Avg
EPS-02-1	0.640042	0.029907	0.118349	0.010232	68.400551
EPS-0.25-1	0.421001	0.031647	0.138310	0.009337	250.775586
EPS-0.25-2	0.404025	0.031931	0.129788	0.007530	252.227605
EPS-0.25-3	0.412972	0.035363	0.134299	0.006495	334.156354
EPS-0.5-1	0.317885	0.023803	0.111750	0.003670	787.295286
EPS-0.5-2	0.390082	0.040720	0.123612	0.007358	774.477352
EPS-0.5-3	0.396455	0.022447	0.138474	0.004920	975.728575
EPS-0.75-1	0.331712	0.027323	0.119864	0.006488	1867.117460
EPS-0.75-2	0.392665	0.023801	0.129767	0.005056	1971.704977
EPS-0.75-3	0.356893	0.024701	0.133839	0.007345	1420.168501
EPS-1-1	0.359216	0.035140	0.118242	0.006293	3003.793817
EPS-1-2	0.368713	0.031546	0.119043	0.004196	2449.483497
EPS-1-3	0.345734	0.030424	0.120758	0.006538	1811.421914
Total Time (s)	-	-	-		319335.029485

Imagen 10 – Porción de la tabla de resultados para la versión Beta 0.6.

Como se puede observar, este nuevo formato de tabla ya contempla todas las medidas requeridas para analizar los experimentos incluyendo el tiempo. Se ha añadido tan solo una parte de tabla puesto que tiene una extensión suficientemente grande como para dificultar la lectura en este documento. Para construir estas tablas se ha seguido el siguiente procedimiento: se genera un libro de Excel para cada método de elección de nodos (uno por cada **centralidad** implementada, más el método *Random*), dentro del libro existe una hoja por cada porcentaje del total de subnodos estudiado y dentro de cada hoja, existe una tabla

por cada mecanismo de negociación implementado, la primera fila indica esto mismo, la segunda fila indica el método de ordenación, la tercera la limitación de color (color fijo o aleatorio), la cuarta el color elegido y la quinta refleja el nombre de cada medida.

3.4.2.3. Versión *RTM*

Para culminar el proceso de versiones de una aplicación existe la versión de *disponibilidad general o RTM*, la cual, suele ser la versión final. Viene caracterizada por una alta estabilidad y un porcentaje muy bajo de errores. Para esta versión tan solo existe la revisión principal.

Se ha incluido una última **centralidad** al conjunto ya implementado, esta es la **centralidad** de *comunicabilidad o "communicability"*. Quedando un total de cinco posibles **centralidades** a emplear por el usuario. Además, se ha añadido una nueva medida al conjunto de medidas hasta ahora contempladas: los intervalos de confianza, al 95%, asociados a la media de las utilidades y su desviación estándar. Debido a esto, el formato definitivo de las tablas es el siguiente:

	1					
	Utility			Fairness Fujita		Time(s)
	U_Norm	STD	IC	Avg	STD	Avg
EPS-02-1	0.677434	0.006792	0.003179	0.114370	0.002511	51.948200
EPS-0.25-1	0.430813	0.009718	0.004548	0.152556	0.003181	174.999426
EPS-0.25-2	0.407537	0.011931	0.005584	0.139041	0.004674	171.572797
EPS-0.25-3	0.317159	0.007241	0.003389	0.139936	0.003318	190.262006
EPS-0.5-1	0.234050	0.008225	0.003849	0.101159	0.004314	524.346444
EPS-0.5-2	0.318136	0.012140	0.005682	0.134550	0.005342	508.410534
EPS-0.5-3	0.372669	0.010961	0.005130	0.143492	0.004459	495.543573
EPS-0.75-1	0.246109	0.014622	0.006843	0.113481	0.006064	1005.386630
EPS-0.75-2	0.352009	0.016361	0.007657	0.131500	0.003863	992.322795
EPS-0.75-3	0.320541	0.012447	0.005825	0.133959	0.003792	930.154177
EPS-1-1	0.307665	0.011111	0.005200	0.121098	0.004275	1646.074888
EPS-1-2	0.315787	0.014446	0.006761	0.124135	0.004671	1725.475376
EPS-1-3	0.316902	0.014754	0.006905	0.126761	0.003831	1675.992930
Total Time (s)	-	-	-	-		201849.795517

Imagen 11 – Porción de la tabla de resultados para la versión RTM.

Por otro lado, se ha implementado el módulo "*XlsWriter*" de Python permitiendo así automatizar la tarea de pasar los resultados de la salida del experimento a la hoja de resultados, para ello se han introducido las sentencias necesarias para generar en cada conjunto de experimentos la siguiente tabla automáticamente:

	Utility			Fairness Fujita		Time(s)
	U_Norm	STD	CI	Avg	STD	Avg
EPS-0.2-1-1	0.690553	0.0089	0.004165	0.101457	0.003338	55.100895
EPS-0.25-1-1	0.520808	0.022629	0.010591	0.126594	0.0048	200.983336
EPS-0.25-2-1	0.504215	0.022665	0.010608	0.11439	0.005411	200.086342
EPS-0.25-3-1	0.550303	0.012868	0.006023	0.108625	0.004616	217.444023
EPS-0.5-1-1	0.437505	0.022922	0.010728	0.106435	0.003842	538.627904
EPS-0.5-2-1	0.460316	0.022144	0.010364	0.119226	0.004706	491.848235
EPS-0.5-3-1	0.478254	0.01468	0.00687	0.130114	0.004824	464.043893
EPS-0.75-1-1	0.433102	0.026294	0.012306	0.116803	0.003125	1088.221044
EPS-0.75-2-1	0.470208	0.023267	0.010889	0.116048	0.006582	927.159291
EPS-0.75-3-1	0.448897	0.023745	0.011113	0.122926	0.004411	894.327125
EPS-1-1-1	0.397129	0.02824	0.013217	0.115136	0.005212	1558.858969
EPS-1-2-1	0.419731	0.022186	0.010384	0.118284	0.003165	1597.62599
EPS-1-3-1	0.383731	0.028036	0.013121	0.118877	0.004341	1579.400141
Total Time(s)	-	-	-	-		196274.543743

Imagen 12 – Tabla generada por el módulo XlsWriter.

Como se puede observar, la tabla generada tiene el mismo formato numérico y de celdas que las porciones de tabla de la versión RTM, con esto, la operación de "pasar" resultados se convierte en una simple operación de copiado y pegado ya que el formato de colores se obtiene al pegar sobre la tabla de resultados respectiva.

Para terminar, se ha implementado control de errores y manejo de excepciones en el script "*launch_experiement.py*", contemplando si el usuario introduce algún método de **coloración**, ordenación o selección no permitido (fuera de rango), interrupciones de teclado, "*system-exit*" y errores inesperados. En el caso de obtenerse un error inesperado, se genera una traza con la fuente del error (aquella función o método donde se ha producido) y otra, con los parámetros usados para dicho experimento, así como la última iteración/**grafo** realizada/empleado.

3.4.3. Optimización GPU

A esta versión de la aplicación se le ha querido dar un apartado propio ya que no ha llegado a implementarse de manera definitiva ni en la fase *Beta* de desarrollo ni en la versión *RTM.* Esto es así debido a que no se consiguió generar una aplicación equivalente o, al menos, con toda la funcionalidad requerida como para llegar a incluirla en la versión definitiva, sin embargo, el tiempo dedicado a intentar generar dicha funcionalidad sobre *GPU* hace que sea mencionable.

3.4.3.1. Objetivo

Principalmente, lo que se pretendía a la hora de generar esta **optimización**/versión era reducir en gran medida los tiempos de ejecución totales de los experimentos lanzados ya que, la mayoría de ellos, no bajaban de los dos días y medio. Antes de comenzar a generar código, se llevó a cabo un estudio para permitir averiguar las posibles fuentes de tiempos tan grandes, es decir, un estudio que reflejara en qué puntos del programa y qué funciones eran las que empleaban mayor tiempo de ejecución. Tras dos días de estudio, se concluyó que el problema de los tiempos residía en la llamada a la función del cálculo de la utilidad de los negociadores automáticos, en concreto, en la función *"util_comp"* del script *"wifi_utility.py"*. Esta función es llamada en cada iteración del negociador en cuestión y lleva a cabo una serie de cálculos para cada nodo del conjunto de nodos que compongan el **grafo** de interferencias del escenario utilizado **en un tiempo total, por nodo, de unos 0.5s** , así pues, esto no supone un gran problema para aquellos **grafos** cuyo conjunto de nodos sea reducido (inferior a 300 nodos), sin embargo, en **grafos** cuyo orden supera los 1000 nodos (en el caso de este TFG, escenarios superiores a *EPS-0.75-1-1)* el tiempo resultante final es excesivo.

3.4.3.2. Problemas hallados

Partiendo de estas conclusiones, se trató de generar el código necesario para poder realizar todas las operaciones de esta función sobre *GPU* de manera paralela, lo que permitiría ejecutar todos los cálculos, sobre todos los nodos, en un **tiempo máximo de 1.5s** (tiempo estimado en base a pruebas llevadas a cabo). No obstante, no se llegó a culminar este objetivo ya que se encontraron varios problemas en el camino:

- **Los *kernels* que emplea el lenguaje o herramienta *PyOpenCL* permiten el paso de ciertos tipos de variables**, limitados por aquellos definidos en el estándar de *C++11* o los definidos en el módulo *Numpy* de Python. En consecuencia, esto implica no poder pasar directamente algunos tipos de variables de Python 2.7 y lo que es más importante, las estructuras ya mencionadas de *"dict-of-dict-of-dict"* empleadas por *NetworkX* para el procesado de sus **grafos**. Consecuentemente, se intentó generar estructuras equivalentes para los *kernels* pero con escasos resultados positivos, puesto que no se tenía toda la información sobre la estructura *"dict-of-dict-of-dict"* básica en *NetworkX,* ni la información acerca de los atributos añadidos a algunos nodos en los *"dict-of-dict-of-dict"* empleados por el equipo de investigación a la hora de generar los **grafos** empleados.
- El lenguaje empleado en los *kernels* de *PyOpenCL* no es otro que *C++* con algunos añadidos, esto implica que **la sintaxis usada para llamar a algunos métodos dentro de Python no es válida**. En principio, esto no supuso un problema grave ya que la solución era adaptar dichas llamadas dentro del propio *Kernel.*
- **El código contenido dentro de un *Kernel* es independiente del código del script de Python** (hecho razonable ya que se ejecutan en dispositivos distintos), por

tanto, ninguno de los módulos importados en el script de Python, ni su funcionalidad o métodos, están disponibles dentro del propio *Kernel*. En sí y de por si esto no es un problema, sino un "agravante" puesto que se en la función *"util_comp"* se hacían algunos cálculos a partir de métodos importados de un módulo de Python, en consecuencia, hubo de buscarse módulos empleados tenían una licencia de código abierto, buscar el código del método empleado y reproducirlo para poder ser usado en un *Kernel*... Esto solo se consiguió para el método "*log10(x)"* del módulo "math".

En resumen, debido a este conjunto de problemas, la falta de conocimiento en algunas áreas de esta herramienta por parte del autor, no formar parte de los objetivos principales del proyecto, así como el tiempo que hubiera requerido desarrollar este sub-proyecto, se decidió paralizar el diseño de esta versión y relegarla a un plano secundario.

3.4.4. Comportamiento Esperado

Antes de pasar a explicar los resultados obtenidos y las conclusiones extraídas de ellos y del conjunto de este trabajo, se ha querido crear un apartado en el que se detalle el comportamiento esperado de los distintos experimentos, permitiendo establecer una comparativa entre esto y los resultados reales, que ayude a extraer conclusiones y facilite su deducción. Sin más dilación, se pasa a exponer cada una de estas directivas:

- **Utilidad - %Subnodos:** De esta relación se espera que según aumente el número de nodos/puntos de acceso pertenecientes al conjunto de subnodos con frecuencia/canal fijo, disminuya la utilidad obtenida en el **grafo**. Se espera este comportamiento puesto que, al limitar el número de nodos sobre los que puede actuar el mecanismo de negociación, la **optimización** que este puede llevar a cabo se reduce.
- **%Subnodos - Tiempo Ejecución:** De manera similar a la anterior relación, se espera de esta un comportamiento inversamente proporcional, es decir, a medida que aumenta el porcentaje de nodos que conforman el subconjunto, el tiempo de ejecución de los experimentos se verá reducido en un cierto valor. Esta potencial conclusión se extrae directamente del código implementado ya que, al llamar a los negociadores, si el nodo en cuestión pertenece al subconjunto de nodos la iteración se ve interrumpida, pasando a la siguiente sin necesidad de realizar todas las operaciones que un nodo no perteneciente tendría.
- **Utilidad – Frecuencias/Canales Fijos:** Al haber operado en los experimentos con el protocolo 802.11n del IEEE se hablará preferiblemente de canales, aunque, en cualquier caso, podría hacerse referencia a estos como "frecuencias". Como ya se ha mencionado anteriormente, en los experimentos se ha operado con tres posibles canales fijos (1,6,11), correspondientes a los extremos de la banda de frecuencia de este protocolo y su centro, y con canales, de los 11 disponibles, asignados de manera aleatoria. Teniendo en cuenta la matriz de interferencias citada en el apartado *"Teoría Introductoria→Modelado",* se espera que, la utilidad obtenida para los canales de los extremos, es decir, los canales *"1" y "11",* tengan resultados parecidos, sino iguales; para el canal 6, sin embargo, se esperan resultados con una utilidad inferior para cualquier escenario ya que esta matriz indica un mayor ratio de interferencia para este canal que para los citados.
- **Utilidad – Canales Fijos - Aleatorios:** Para este caso, se espera que la utilidad para aquellos escenarios en los que el **coloreado** se lleve a cabo de manera aleatoria sea

mayor que la de aquellos escenarios en los que el **coloreado** se ha realizado mediante un color fijo. Esto se puede extraer de manera simple, al plantear la probabilidad de interferencias entre nodos en estos dos escenarios: en el primero de ellos, con **coloreado** aleatorio, los nodos con frecuencia fija tienen una probabilidad alta de no tener el mismo color o, al menos, una probabilidad suficientemente alta como para pensar que si dos de los nodos escogidos están lo suficientemente cerca como para crear interferencias entre sí, dichas interferencias serán menores que el caso en que ambos nodos tienen el mismo color.

- **Utilidad – Número de Nodos:** Se espera un mayor valor de utilidad para aquellos escenarios o **grafos** en los que el orden del **grafo** sea menor, es decir, cuanto menor sea el número de **grafos**, mayor será la utilidad. Cabe tener en cuenta esta potencial conclusión, debido a que, en principio, siempre es más fácil optimizar la interferencia generada entre tres nodos que entre 500.
- **Tiempo – Número de Nodos:** De manera similar al apartado anterior, cabe esperar que el tiempo de ejecución tenga una relación directamente proporcional con respecto al orden del **grafo**, por el simple hecho de que, cuanto mayor sea el orden, mayor cantidad de operaciones tendrán que llevarse a cabo.
- **Utilidad – Centralidades:** Debido a la propia definición de las **centralidades**, se espera de la **centralidad** de *Vector Propio o EigenVector* los mejores resultados. Con respecto al resto de **centralidades** no se ha extraído, a priori, ninguna otra conclusión.
- **Utilidad – Centralidades – Random:** Para una amplia mayoría de los resultados, se espera que la utilidad de aquellos escenarios cuyo método de selección del conjunto de subnodos haya sido *"Random"* sea mayor a aquellos en los que el método empleado sea una **centralidad**.
- **Utilidad – Negociadores:** En referencia a lo expuesto en el apartado *"Negociadores Automáticos"* se tiene la expectativa de que el mecanismo de negociación que mejores resultados, referentes al valor de la utilidad, obtenga sea el citado "*Annealer*" ya que corrige los defectos (escapar de máximos locales) de "*Hill Climber*". En último lugar, con peores resultados se espera que figure la técnica "*referencia Random*".
- **Utilidad – Método de ordenación:** Es de esperar que tengan menor utilidad aquellos escenarios ordenados mediante los nodos más centrales, puesto que el hecho de no poder actuar sobre ellos para optimizarlos recaerá de manera directa en la utilidad, disminuyendo su potencial valor. Al contrario, para los métodos de ordenación mediante *Distribución Normal* y *nodos con menor* ***centralidad*** se espera tengan una utilidad mayor con respecto al anterior método. De estos dos últimos métodos no se ha podido distinguir, a priori, cuál de los dos devolverá mejores resultados.

4. Resultados

En este apartado se procede a explicar el conjunto de experimentos desarrollados por el autor de este TFG. Se han lanzado, sin tener en cuenta aquellas ejecuciones de prueba o testado, un total de **368** experimentos (alrededor de un tercio del conjunto total posible de experimentos), cada uno de ellos con **20 repeticiones** y **3000 iteraciones** para los negociadores automático. Cada uno de estos experimentos estaba compuesto por los **13 grafos** generados para simular el entorno de la *Escuela Politécnica Superior de Alcalá de Henares* y tardaba en ejecutarse alrededor de dos días y medio, por tanto, el factor del tiempo y los equipos empleados han sido una de las mayores limitaciones a la hora de poder haber llevado a cabo una mayor cantidad de experimentos en estos meses.

Antes de comenzar a exponer los resultados, se adjunta una imagen[18] en el anexo para que el lector tenga una referencia gráfica de todos los posibles escenarios que podrían generarse a partir de las condiciones implementadas. Como se puede apreciar en esta, las posibilidades son inmensas, solo en la rama de los escenarios sin **centralidad** como método de elección del conjunto de subnodos (la rama *Random)*, ya es posible generar **64** tipos distintos de escenario. En consecuencia, para todos los métodos de ordenación de cada una de las **centralidades**, se pueden generar **192** escenarios diferentes y, por lo tanto, en total, se consigue un conjunto de **1024** posibles escenarios con, al menos, una característica que los distingue del resto de escenarios. Cabe destacar que en la imagen no se han representado todas las posibilidades ya que hubiera resultado imposible adjuntar la imagen resultante en este documento.

[18] Anexo: Imagen 27 – Esquema gráfico con todas las posibilidades para desarrollar un escenario.

4.1. Utilidad

En este subapartado se contemplarán y analizarán una serie de resultados y gráficas relativos a el parámetro *Utilidad.* Gracias a ello se podrán extraer una serie de conclusiones que confirmen o rebatan las teorías contempladas en el apartado *Comportamiento Esperado.*

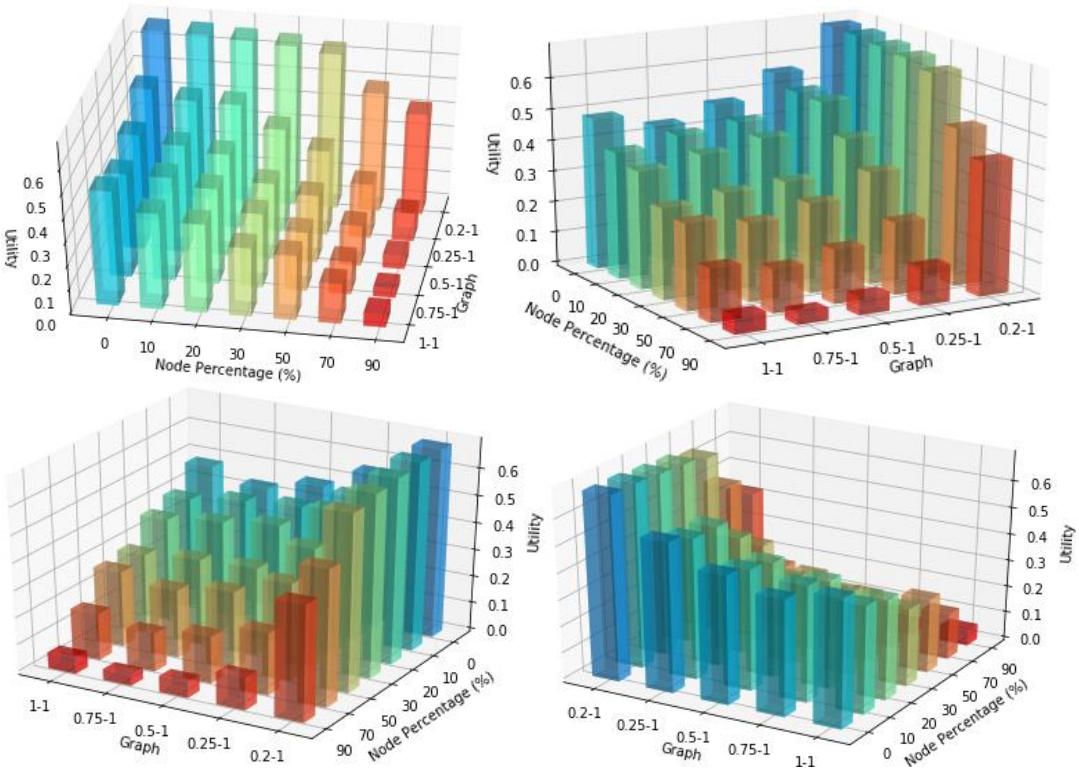


Imagen 13 – Utilidad de SA para distintos grafos (EPS) y porcentajes de subnodos.

En estas primeras gráficas realizadas para la técnica de negociación ***Simulated Annealing*** con **centralidad *EigenVector*, canal/color fijo *1*** y ***ordenación*** mediante ***Distribución Normal,*** *es* posible observar cómo la utilidad decae en base a dos factores: el primero de ellos es el n**úmero de nodos que componen el conjunto de subnodos**, esto es, como se había supuesto en el apartado de *Comportamiento Esperado,* cuanto mayor es el número de nodos que componen este conjunto, menor es la utilidad obtenida; por otro lado, cuanto mayor es **el orden del grafo**, menor es la utilidad obtenida, es decir, escenarios con un menor número total de nodos siempre tendrán mayor utilidad. A continuación, se adjuntan las tablas correspondientes a cada una de las técnicas de negociación empleadas para observar este mismo comportamiento en todas ellas y concluir que se trata de una norma genérica y no solo aplicable a las condiciones de las gráficas anteriormente descritas.

%subnodos/ Escenario	0%	10%	20%	30%
0.2-1	0.642	0.629	0.612	0.594
0.25-1	0.470	0.471	0.446	0.351
0.5-1	0.402	0.384	0.362	0.338
0.75-1	0.369	0.379	0.353	0.306
1-1	0.358	0.341	0.305	0.293

Tabla 7– Utilidad de HC para distintos grafos (EPS) y porcentajes de subnodos.

%subnodos/ Escenario	0%	10%	20%	30%
0.2-1	0.642	0.629	0.612	0.594
0.25-1	0.470	0.471	0.446	0.351
0.5-1	0.402	0.384	0.362	0.338
0.75-1	0.369	0.379	0.353	0.306
1-1	0.358	0.341	0.305	0.293

Tabla 8– Utilidad de LCCS para distintos grafos (EPS) y porcentajes de subnodos.

%subnodos/ Escenario	0%	10%	20%	30%
0.2-1	0.642	0.629	0.612	0.594
0.25-1	0.470	0.471	0.446	0.351
0.5-1	0.402	0.384	0.362	0.338
0.75-1	0.369	0.379	0.353	0.306
1-1	0.358	0.341	0.305	0.293

Tabla 9– Utilidad de Rnd para distintos grafos (EPS) y porcentajes de subnodos.

Nota: Tan solo se han realizado experimentos para porcentajes del conjunto de subnodos superiores al 30% para el negociador *Simulated Annealing* puesto que es el más representativo en términos de utilidad.

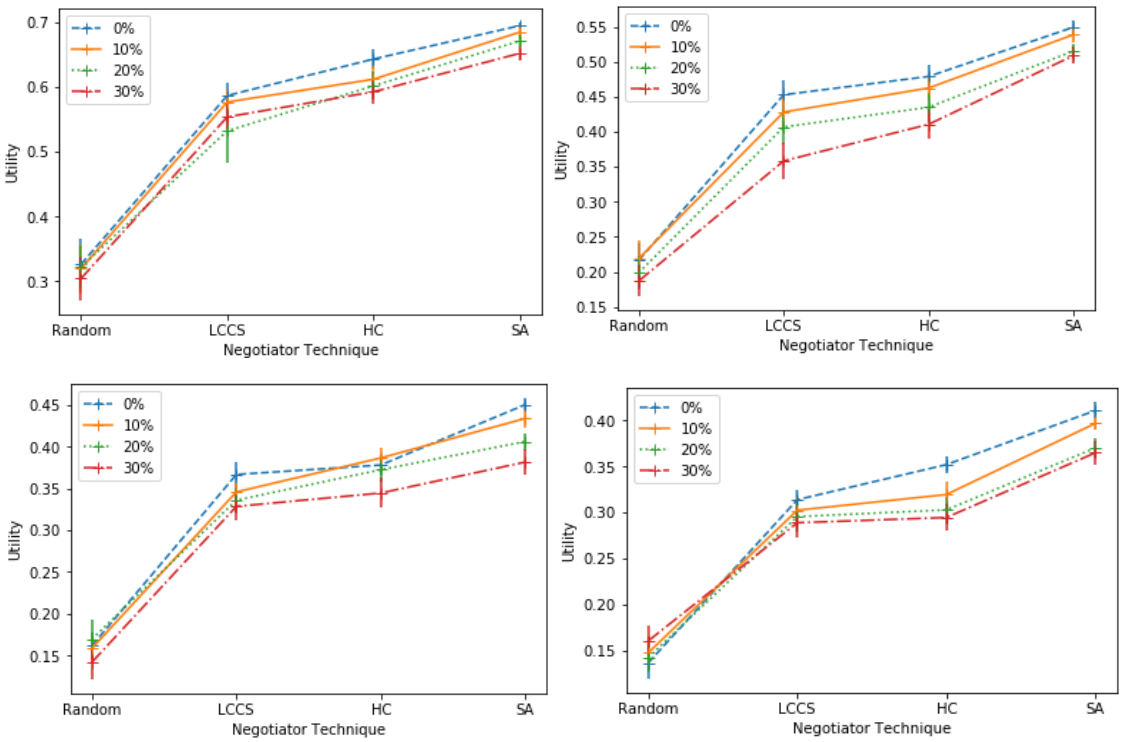


Imagen 14 - Utilidad frente a distintos tipos de negociadores y grafos. Figura superior izquierda: EPS-0.2-1; superior derecha: EPS-0.25-3; inferior izquierda: EPS-0.75-3; inferior derecha: EPS-1-3.

En este caso, se ha reflejado el comportamiento de distintos **grafos** EPS para la **centralidad *EigenVector***, **canal/color aleatorio** y ***ordenación*** mediante ***Distribución Normal***. Es obvio, en vista de las gráficas, que las mejor de las técnicas de negociación es, como se había previsto teóricamente, *Simulated Anneling*. A la par, se mantienen las conclusiones sacadas a partir de las anteriores gráficas relativas a la caída de utilidad con respecto al porcentaje de nodos que conforma el conjunto de subnodos (en la gráfica inferior izquierda, para el método *Hill Climber*, es posible observar una excepción a esto, pero se justifica atribuyendo el valor obtenido a la asignación aleatoria de canal dada en el ejemplo), y también se aprecia dicha caída con respecto al número total de nodos (el eje "*y*" de las gráficas se ha ajustado de tal manera que esto sea visible).

En estas gráficas también se ha incluido, para cada valor reflejado, el valor del intervalo de confianza al 95%. Dicho valor se encuentra representado mediante las barras verticales ubicadas en cada punto de la gráfica, se ha procurado que dichos valores no se superpongan, pero, en algunos casos debido al escalado de la propia gráfica ha sido imposible. Es importante destacar que, en ningún caso, estos valores han superado las tres centésimas, en otras palabras, el mayor "*error*" asociado ha sido de $x \pm 0.03$.

Cabe destacar que el comportamiento reflejado en estas gráficas se mantiene en el resto de experimentos abordados en este trabajo[19], confirmando así la teoría abordada en el apartado *Comportamiento Esperado* acerca de la relación entre la utilidad y las distintas técnicas de negociación empleadas.

[19] Véase en el Anexo: Tabla 14, Tabla 15, Tabla 16, Tabla 17 y Tabla 18.

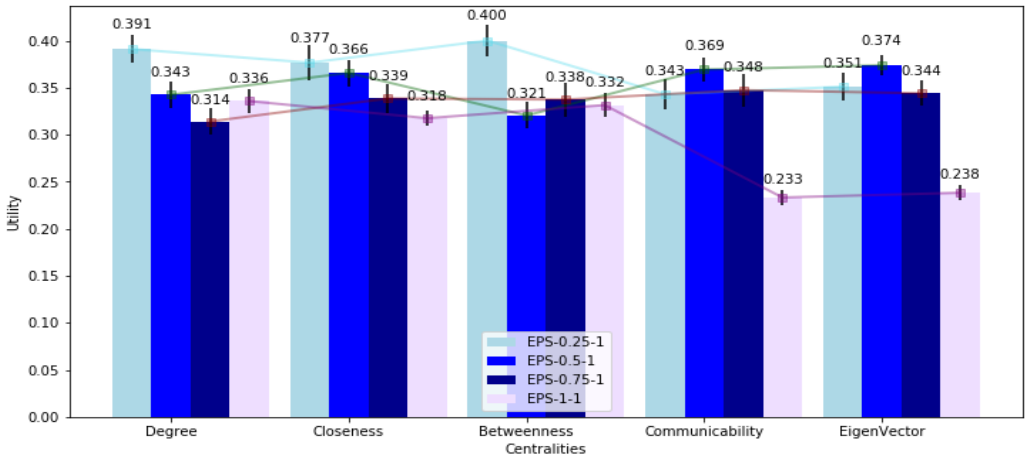


Imagen 15 - Utilidad frente a distintas centralidades. 30% de subnodos con ordenación menos central, color/canal fijo 6 y negociador HC. Escenarios EPS-X-1-1.

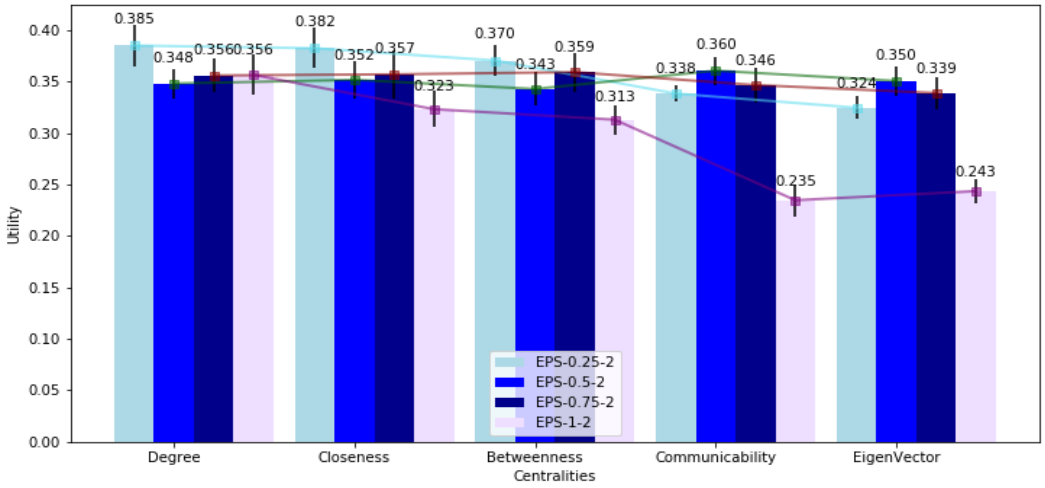


Imagen 16 - Utilidad frente a distintas centralidades. 30% de subnodos con ordenación menos central, color/canal fijo 6 y negociador HC. Escenarios EPS-X-2-1.

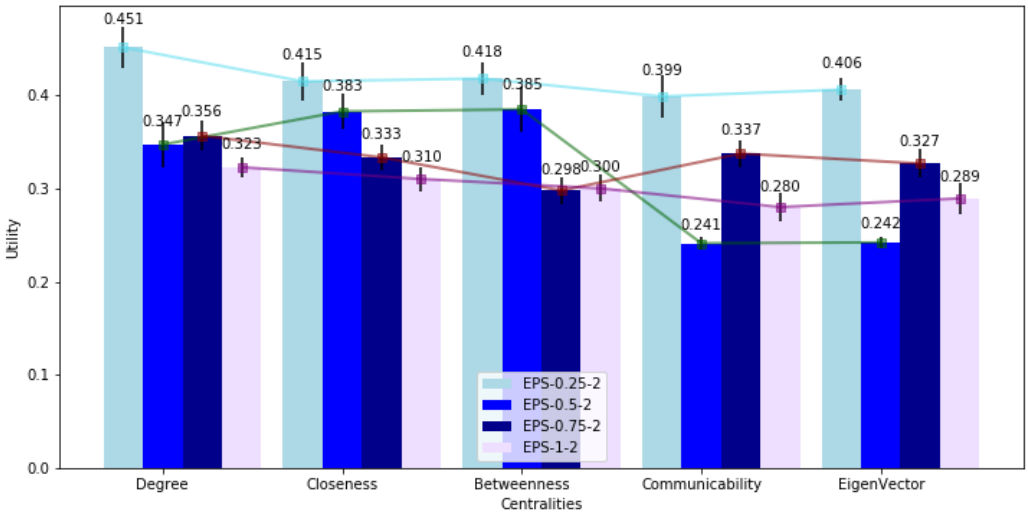


Imagen 17 - Utilidad frente a distintas centralidades. 30% de subnodos con ordenación menos central, color/canal fijo 6 y negociador HC. Escenarios EPS-X-3-1.

El volumen de experimentos realizados que permitan distinguir o establecer qué **centralidad** es más eficiente, en términos de utilidad, de manera genérica, ha sido muy limitado puesto que el autor de este trabajo dio preferencia a la **centralidad** *EigenVector* siguiendo el consejo del equipo de investigación y debido a que esta era la **centralidad** que mejores resultados les había dado. Además, el tiempo requerido para estos "nuevos" experimentos habría excedido la fecha final de este TFG.

Así pues y **en base a los resultados representados en las tres gráficas anteriores se puede extraer una conclusión parcial**, no extensible a todos los posibles experimentos, que cubra y justifique dichos valores para estos experimentos concretos (**grafos** EPS-X-X-1, HC, color fijo 6, ordenación menos central), esta es: en los tres casos reflejados, se puede observar cómo, para aquellos escenarios con **menor número de nodos** (*EPS-0.25-x-1*), es decir, de menor orden, la **centralidad** que más consigue "explotar" la utilidad es la conocida como ***Centralidad de Grado***. En el caso contrario, para aquellos **grafos de mayor orden** (*EPS-1-X-1*), este puesto se lo llevan las **centralidades *Closeness* y, especialmente, *Betweenness;*** las **centralidad**es *EigenVector y Communicability*, para este caso, son las que peores resultados obtienen. Para terminar, en los **escenarios con un número de nodos "estándar"** (por debajo de los de *EPS-1-x-1* y por encima de los de *EPS-0.25-x-1*), es decir, *EPS-0.5-x-1 y EPS-0.75-x-1*, se obtienen los mejores valores para las **centralidades** que en el caso de **grafos** con mayor orden obtuvieron los peores resultados, esto es: ***Communicability y, especialmente, EigenVector.***

Como ya se ha mencionado, el número de experimentos ejecutados para extraer conclusiones respectivas a la relación "utilidad-tipo de **centralidad**" ha sido insuficiente como para poder elevar estas a un grado más genérico y, por tanto, se delega e incita a futuros investigadores que lleven a cabo más experimentos como para poder satisfacer la norma de esta relación y concluir lo necesario.

Finalmente, en el *Anexo* se han incluido tres tablas que reflejan el comportamiento de varios escenarios en los que se ha empleado la **centralidad** *Closeness* en vez de la **centralidad** *EigenVector*. Referencia cruzada a las tablas:

- Tabla 23 - Comportamiento experimento 10% subnodos, HC, centralidad Closeness, distribución normal y canal/color fijo 1.

-Tabla 24- Comportamiento experimento 10% subnodos, LCCS, centralidad Closeness, ordenación por nodos de mayor centralidad y canal/color aleatorio.

-Tabla 25- Comportamiento experimento 20% subnodos, SA, centralidad Closeness, ordenación por nodos de menor centralidad y canal/color fijo 1.

-Tabla 26- Comportamiento experimento 30% subnodos, HC, centralidad Closeness, ordenación por nodos de mayor centralidad y canal/color fijo 11.

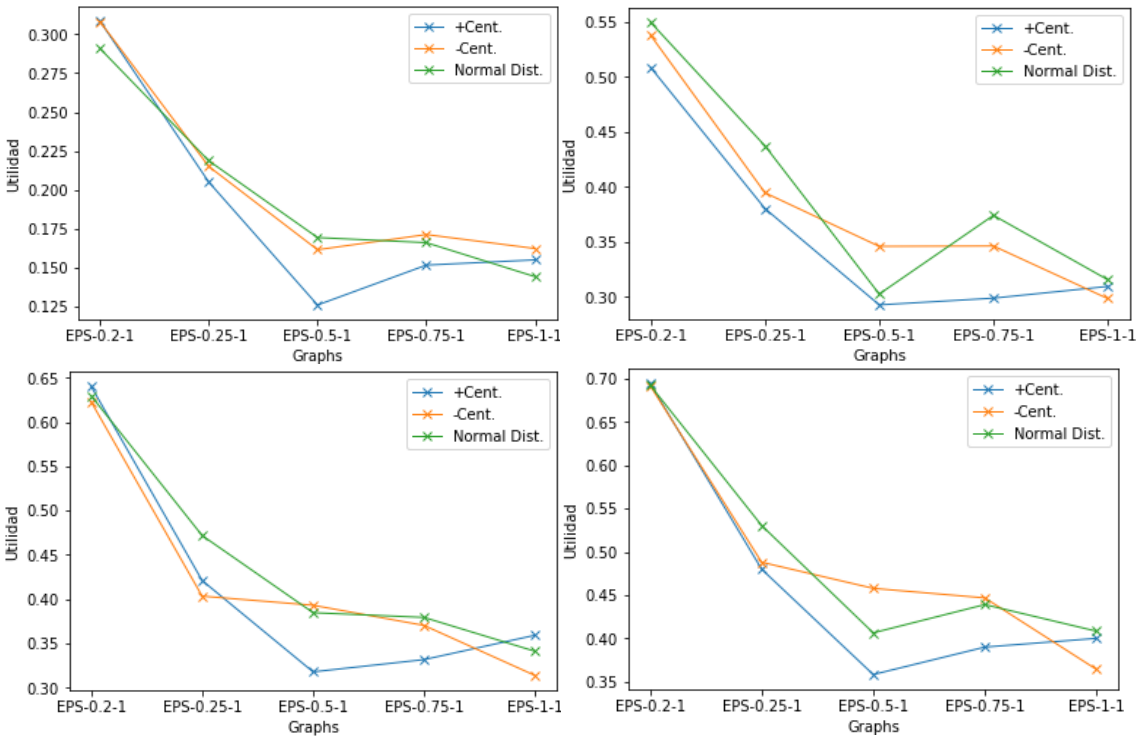


Imagen 18 - Utilidad frente a métodos de ordenación en distintos negociadores. Figura superior izquierda: Rnd; superior derecha: LCCS; inferior izquierda: HC; inferior derecha: SA.

Una vez más, gracias a los resultados obtenidos, podemos confirmar una de las teorías establecidas en el apartado de *Comportamiento Esperado*, en este caso, la relativa a la relación entre la utilidad y los distintos métodos de ordenación. En ella se esperaba que los escenarios en los que el método de ordenación fuera en base a ***los nodos más centrales*** la **utilidad fuese menor** puesto que los nodos escogidos en dicho caso tendrían, en cierto modo, un mayor peso en el **grafo** dado y, por tanto, el hecho de que los negociadores no pudieran actuar sobre estos nodos empeoraría los resultados reduciendo la potencial **optimización** de utilidad de estos. Por el contrario, también se mantenía que los otros dos métodos de ordenación existentes, *Distribución Normal y nodos con menor* ***centralidad,*** alcanzarían unos mayores valores de utilidad en los distintos escenarios de cada experimento.

En las gráficas mostradas, extraídas de los resultados más representativos para este caso: *10% de subnodos, varios negociadores,* ***centralidad*** *EigenVector, varios métodos de ordenación y color/canal fijo 1*; se evidencia de manera clara, en todos los casos, que **el peor método de ordenación** de los métodos estudiados es el **método de ordenación para nodos con mayor centralidad.** A la par, se pone de manifiesto que no existe un "ganador" claro entre los otros dos métodos, en otras palabras, **no existe un método de ordenación que consiga los mejores resultados de manera genérica y para todos los escenarios.** Sin embargo, resulta prudente intentar establecer cuál de ambos métodos tiene un mejor comportamiento, en media. La siguiente tabla pretende analizar el comportamiento de estos métodos, en media, para los escenarios expuestos en las anteriores gráficas:

M.Ordenación /Negociador	Distribución Normal (Utilidad)		Nodos con menor centralidad (Utilidad)	
-	Media	Desv.Estándar	Media	Desv.Estándar
Random	0.2096	0.0588	0.2005	0.0625
LCCS	0.3957	0.1010	0.3845	0.0919
HC	0.4412	0.1155	0.4206	0.1181
SA	0.4953	0.1209	0.4898	0.1218

Tabla 10 - Comparativa entre métodos de ordenación.

En media, para todos los negociadores, tiene un mayor valor el método de ordenación mediante ***Distribución Normal***, empero, no hay que olvidar que se trata de un parámetro estadístico y que, según lo mostrado en las gráficas, para algunos casos particulares su comportamiento, respecto a la utilidad, es peor que el método de ordenación para nodos con menor **centralidad**. En definitiva, convendría realizar un estudio con mayor profundidad en futuras investigaciones/proyectos si se quiere esclarecer cuál de estos dos métodos es más conveniente en la mayoría de los casos, aunque se advierte, de que es muy posible que tampoco se pueda extraer una conclusión genérica si no, conclusiones locales a determinados escenarios o casos.

A continuación se procede a analizar la relación entre la utilidad y los distintos modos de selección de color, para ello se mostrarán una serie de gráficos de barras horizontales para cada negociador en un mismo escenario:

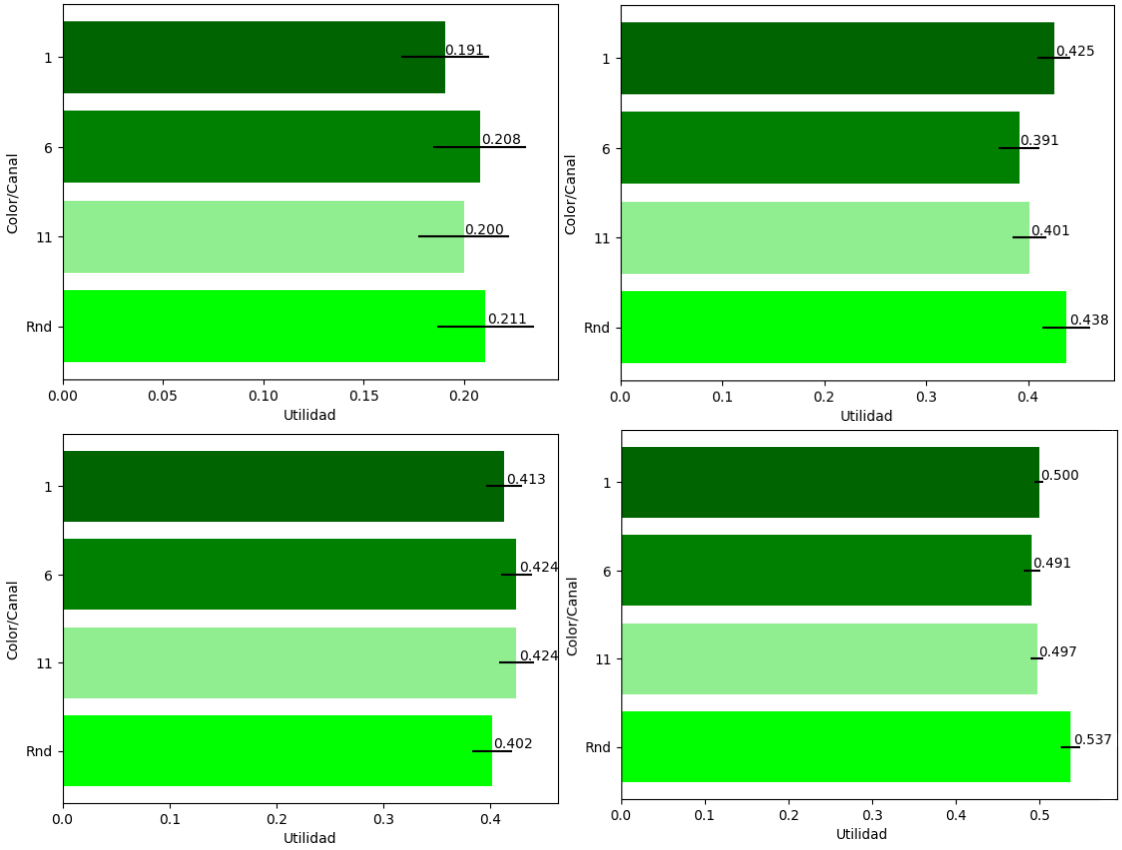


Imagen 19 - Utilidad frente a color, escenario EPS-0.25-3-1, 10%, EigenVector y nodos de mayor centralidad. Figura superior izquierda: Rnd; superior derecha: LCCS; inferior izquierda: HC; inferior derecha: SA.

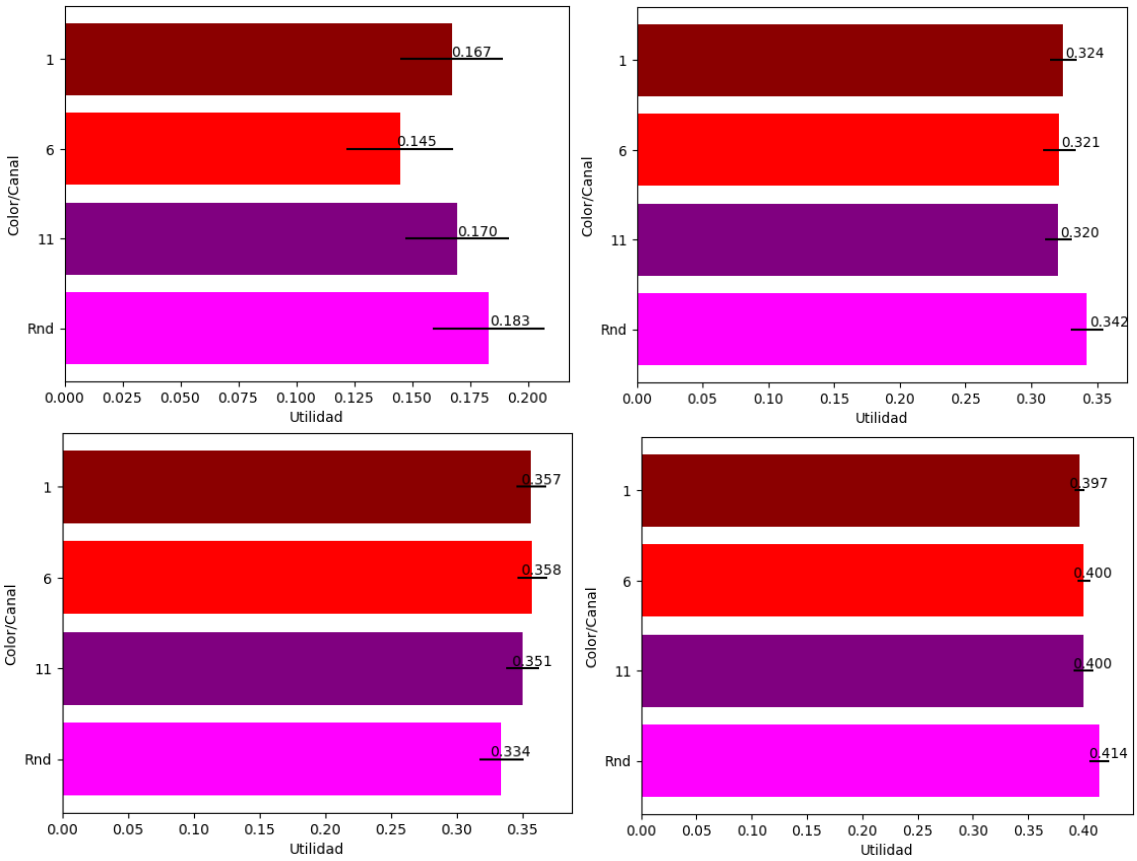


Imagen 20 - Utilidad frente a color, escenario EPS-0.75-3-1, 10%, EigenVector y nodos de mayor centralidad. Figura superior izquierda: Rnd; superior derecha: LCCS; inferior izquierda: HC; inferior derecha: SA.

Bien, quizá se podría afirmar que estos resultados[20] son los que más han sorprendido al autor, así como al equipo de investigación puesto que, como se había comentado en anteriores apartados, se esperaba que l**a utilidad para los canales 1 y 11 fuera muy parecida** en todos los casos, que e**l canal 6 tuviera una utilidad menor con respecto a estos dos canales** (se trata de un canal con más interferencias al no estar en un extremo del ancho de banda empleado) y, finalmente, que la asignación *Random* de color tuviera, en general, mayores valores de utilidad que cualquiera de los anteriores canales (las posibilidades de **optimización** por parte del negociador son más altas porque la probabilidad de escoger, para varios puntos de acceso, el mismo canal es menor). Sin embargo, el comportamiento esperado con respecto a la asignación de colores fijos solo ha sido posible verlo en casos aislados y sin una pauta aparente, en las gráficas esto queda reflejado de manera concisa: los canales 1 y 11, a pesar de tener utilidades similares, no son tan similares como cabría esperar siendo los canales límite del ancho de banda y, por otro lado, no es posible apreciar que el canal 6 siempre o, en la mayoría de casos tenga una utilidad menor que los canales 1 y 11, siendo, muy a menudo, su valor equivalente al de estos canales.

Se han llevado a cabo, para intentar dar explicación a este comportamiento, algunos experimentos de prueba que fijaban el valor inicial de la lista de canales asociados a cada punto de acceso (en los experimentos normales estos valores se generan de manera aleatoria) para observar si la causa radicaba en que al generar dichos valores de manera aleatoria normalmente, en algunos casos la lista inicial tuviera una mayor probabilidad de ser optimizada para el canal 6 y, por ello, se obtuvieran estos valores tan altos para el canal 6.

[20] Véase el *Anexo, en concreto las tablas: 23-26 ; para ampliar los resultados relativos a este aspecto.*

Empero, los resultados de estos experimentos **no fueron concluyentes**, una vez más el comportamiento esperado no llegó a observarse y por ello, se descartó esta fuente o teoría. Finalmente, se decidió investigar la matriz de interferencias empleada y, como ya se ha comentado anteriormente, se observó que esta **no es una matriz simétrica**, esto implica que las interferencias esperadas "iguales" para los canales 1 y 11, no existen como tal y esto justificaría los resultados para estos canales. No se ha alcanzado una conclusión definitiva para el comportamiento observado en el canal 6.

Por otra parte, sí que se manifiesta el comportamiento esperado en relación a la utilidad para los escenarios con asignación de color Random con respecto a aquellos con asignación fija. La primera, en la mayoría de los casos, obtiene unos mayores valores para la utilidad; aquellos casos en que esto no se cumple, podrían justificarse debido a que la asignación aleatoria resultante para dicho escenario ha sido peor, en términos de **optimización** del **grafo**, que la asignación de un canal fijo.

4.2. Tiempo de ejecución

En este apartado se analizarán los resultados extraídos referentes al tiempo de ejecución necesario para los escenarios empleados, bajo distintas condiciones.

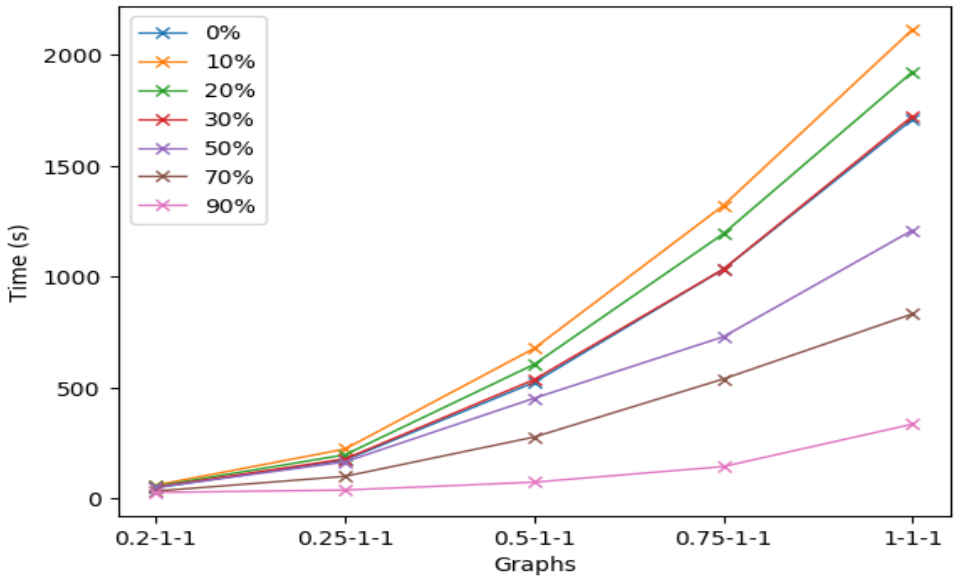


Imagen 21 - Relación tiempo ejecución - %subnodos - escenario.

De esta gráfica se pueden extraer dos grandes conclusiones: la primera de ellas permite relacionar el tiempo de ejecución con respecto al orden del **grafo** empleado, así pues, esta relación sigue una proporción cuasi exponencial (especialmente para porcentajes del conjunto de subnodos bajos), es decir, cuanto mayor sea el orden del **grafo**, mayor será el tiempo requerido para la ejecución del experimento, lo cual tiene sentido puesto que existe un mayor número de datos para procesar. La segunda gran conclusión relaciona el número de subnodos existentes en el subconjunto con el tiempo de ejecución, en este caso, la relación es inversamente proporcional ya que, cuanto mayor sea el número de nodos que tiene este subconjunto menor será el tiempo empleado para la ejecución del experimento (mantiene la lógica puesto que cada nodo dentro del subconjunto requiere menos operaciones de procesado por parte del negociador que un nodo normal). Para esta última

conclusión, existe una excepción, el caso en que no existe un subconjunto de nodos con un canal fijo; como se puede observar, el tiempo de ejecución de este es muy similar al de un porcentaje del 30% de subnodos. La explicación de esto radica en las llamadas a funciones que es necesario realizar en el caso de querer crear un subconjunto de nodos con canales fijos, estas llamadas y, en aquellos casos en los que se emplee una **centralidad**, el cálculo de la **centralidad** pertinente generan un intervalo de procesado o tiempo de ejecución que, en el caso de no tener dicho subconjunto, no es necesario.

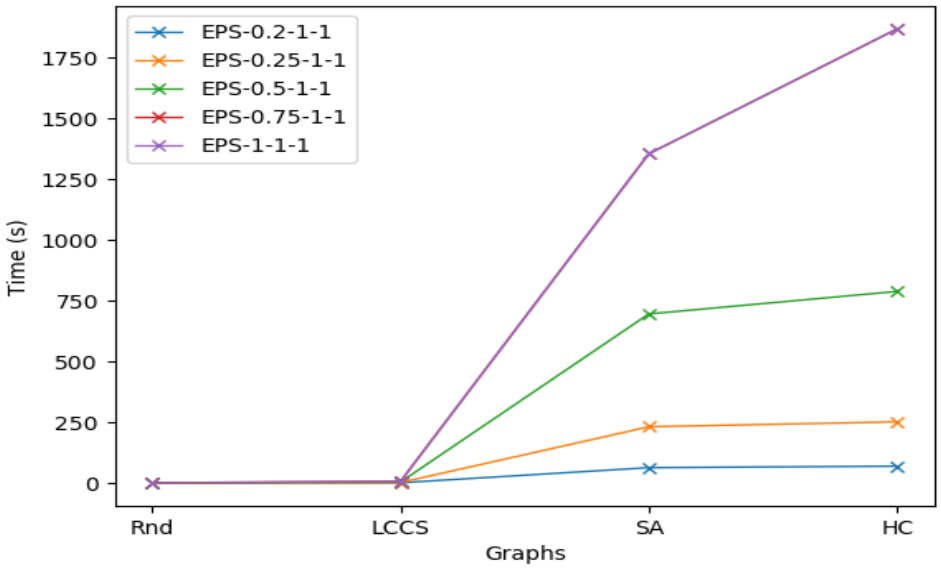


Imagen 22 - Relación tiempo ejecución - técnicas de negociación.

En esta gráfica se han querido comparar los tiempos de ejecución para las distintas técnicas de negociación automática y, como se hace evidente, la **técnica más rápida** ha sido la técnica ***Referencia Random*** puesto que no requiere ningún tipo de procesado. Por otra parte, la **técnica más lenta ha sido la de *Hill Climber*.**

Estos resultados, unidos a los extraídos del anterior apartado acerca de la utilidad permiten establecer una afirmación clara acerca de los negociadores más eficientes en lo que a utilidad y tiempo de ejecución respecta, estos son:

- ***LCCS:*** Esta técnica, a pesar de conseguir utilidades inferiores a *Hill Climber* y *Simulated Annealing*, compensa esta pérdida con una ganancia enorme en tiempo de ejecución[21].
- ***Simulated Annealing:*** Además de ser la técnica con utilidades más altas, es más veloz que *Hill Climber* que es la técnica con utilidades más próximas.

En definitiva, **la técnica más eficiente en relación tiempo – utilidad es *LCCS*,** seguida por ***Simulated Annealing*** con mayores valores de utilidad pero a costa de un gran incremento en el tiempo de ejecución.

[21] Véase la "Tabla 31 - Relación tiempo de ejecución - Técnica de negociación." en el Anexo.

Para completar este apartado se ha querido añadir una relación de tiempos de ejecución de las distintas **centralidades** implementadas, es decir, el tiempo que tarde en calcularse la **centralidad** para cada uno de los nodos de un **grafo**, para ello se ha elegido el escenario más representativo, es decir, "*EPS-1-2-1*" puesto que es el **grafo** con mayor orden de los **grafos** empleados . Todo ello, se refleja en la siguiente tabla:

Centralidad	Tiempo de Procesado (s)
Degree	0.000999927520752
Closeness	39.1500000954
Betweennes	110.493000031
EigenVector	15.6209998131
Communicability	1.00400018692

Claramente la **centralidad** que más tiempo de procesado necesita es la **centralidad** *Betweenness.*

Uniendo estos resultados a los obtenidos en el anterior apartado "*Utilidad*", se puede afirmar que la **centralidad** más eficiente, respecto a la relación utilidad - tiempo, en **grafos** con un orden pequeño es la **centralidad** de *Grado*, para el resto de casos, la **centralidad** *EigenVector* suple este puesto.

4.3. Fairness

Para terminar de exponer los resultados, este apartado se va a dedicar a mostrar aquellos que estén relacionados con el parámetro *Fairness*, intentando establecer una relación con aquellos **grafos** o negociadores que mejores valores consigan para este parámetro. Con ello se conseguirá poder valorar qué escenarios son más *justos* en base a esta medida. A continuación, se adjuntan varias tablas, una por negociador, con los resultados más representativos (10% subnodos, **centralidad** *EigenVector*, ordenación por nodos más centrales):

Escenario (EPS)	***1***		***6***		***11***		***Rnd***	
-	***Avg***	***STD***	***Avg***	***STD***	***Avg***	***STD***	***Avg***	***STD***
0.2-1-1	0.141	0.024	0.131	0.022	0.122	0.020	0.124	0.022
0.25-1-1	0.087	0.022	0.077	0.017	0.093	0.018	0.089	0.020
0.25-2-1	0.072	0.019	0.074	0.016	0.080	0.017	0.083	0.016
0.25-3-1	0.070	0.017	0.072	0.013	0.065	0.014	0.090	0.014
0.5-1-	0.048	0.013	0.045	0.009	0.042	0.011	0.070	0.014
0.5-2-1	0.054	0.022	0.054	0.015	0.061	0.017	0.076	0.015
0.5-3-1	0.079	0.020	0.069	0.016	0.071	0.019	0.082	0.014
0.75-1-1	0.056	0.014	0.056	0.011	0.059	0.014	0.073	0.017
0.75-2-1	0.066	0.015	0.065	0.019	0.067	0.014	0.077	0.016
0.75-3-1	0.068	0.019	0.070	0.016	0.066	0.018	0.079	0.015
1-1-1	0.056	0.014	0.061	0.017	0.064	0.014	0.070	0.014
1-2-1	0.067	0.015	0.065	0.015	0.060	0.019	0.072	0.013
1-3-1	0.068	0.021	0.065	0.015	0.069	0.015	0.066	0.020

Tabla 11 - Fairness para Referencia Random.

Escenario (EPS)	***1***		***6***		***11***		***Rnd***	
-	***Avg***	***STD***	***Avg***	***STD***	***Avg***	***STD***	***Avg***	***STD***
0.2-1-1	0.146	0.008	0.144	0.011	0.147	0.012	0.136	0.010
0.25-1-1	0.136	0.008	0.136	0.009	0.136	0.010	0.126	0.008
0.25-2-1	0.120	0.007	0.123	0.011	0.120	0.009	0.115	0.008
0.25-3-1	0.118	0.007	0.120	0.008	0.120	0.006	0.119	0.009
0.5-1-	0.076	0.008	0.080	0.012	0.078	0.009	0.098	0.009
0.5-2-1	0.106	0.008	0.109	0.010	0.111	0.007	0.104	0.009
0.5-3-1	0.122	0.013	0.126	0.012	0.120	0.012	0.120	0.008
0.75-1-1	0.098	0.008	0.099	0.007	0.097	0.007	0.105	0.008
0.75-2-1	0.113	0.009	0.113	0.012	0.110	0.012	0.107	0.009
0.75-3-1	0.121	0.005	0.120	0.008	0.121	0.006	0.114	0.007
1-1-1	0.100	0.007	0.100	0.007	0.103	0.005	0.099	0.008
1-2-1	0.109	0.006	0.107	0.007	0.105	0.008	0.106	0.007
1-3-1	0.104	0.010	0.101	0.011	0.099	0.010	0.095	0.013

Tabla 12 - Fairness para LCCS.

Escenario (EPS)	***1***		***6***		***11***		***Rnd***	
-	***Avg***	***STD***	***Avg***	***STD***	***Avg***	***STD***	***Avg***	***STD***
0.2-1-1	0.141	0.015	0.134	0.014	0.141	0.016	0.128	0.016
0.25-1-1	0.144	0.008	0.149	0.007	0.148	0.007	0.136	0.008
0.25-2-1	0.135	0.009	0.131	0.013	0.134	0.013	0.125	0.008
0.25-3-1	0.130	0.007	0.130	0.009	0.130	0.009	0.126	0.008
0.5-1-	0.087	0.006	0.090	0.011	0.087	0.008	0.102	0.008
0.5-2-1	0.120	0.009	0.125	0.008	0.124	0.008	0.120	0.008
0.5-3-1	0.133	0.007	0.135	0.007	0.137	0.007	0.131	0.006
0.75-1-1	0.106	0.008	0.106	0.008	0.109	0.004	0.117	0.007
0.75-2-1	0.125	0.007	0.125	0.007	0.125	0.007	0.115	0.008
0.75-3-1	0.131	0.006	0.126	0.007	0.128	0.009	0.124	0.007
1-1-1	0.118	0.006	0.115	0.009	0.116	0.005	0.112	0.007
1-2-1	0.119	0.005	0.118	0.009	0.120	0.008	0.116	0.007
1-3-1	0.120	0.006	0.119	0.008	0.122	0.005	0.117	0.007

Tabla 13 - Fairness para Hill Climber.

Escenario (EPS)	***1***		***6***		***11***		***Rnd***	
-	***Avg***	***STD***	***Avg***	***STD***	***Avg***	***STD***	***Avg***	***STD***
0.2-1-1	0.114	0.003	0.114	0.003	0.116	0.004	0.109	0.007
0.25-1-1	0.153	0.003	0.153	0.005	0.152	0.005	0.140	0.008
0.25-2-1	0.139	0.005	0.139	0.006	0.138	0.004	0.128	0.005
0.25-3-1	0.140	0.003	0.142	0.005	0.141	0.004	0.132	0.005
0.5-1-	0.101	0.004	0.101	0.004	0.101	0.005	0.109	0.007
0.5-2-1	0.135	0.005	0.137	0.004	0.135	0.006	0.129	0.007
0.5-3-1	0.143	0.004	0.144	0.003	0.145	0.004	0.137	0.006
0.75-1-1	0.113	0.006	0.115	0.005	0.115	0.004	0.119	0.006
0.75-2-1	0.132	0.004	0.128	0.005	0.133	0.005	0.121	0.005
0.75-3-1	0.134	0.004	0.135	0.005	0.136	0.003	0.126	0.007
1-1-1	0.121	0.004	0.122	0.004	0.122	0.004	0.120	0.005
1-2-1	0.124	0.005	0.125	0.005	0.127	0.005	0.119	0.005
1-3-1	0.127	0.004	0.126	0.004	0.126	0.004	0.121	0.006

Tabla 14 - Fairness para Simulated Annealing.

Aunque el conjunto de datos expuesto sea grande (a su vez este tan solo es una porción del conjunto total de resultados), si se analizan detenidamente las tablas se puede concluir que, en general, para la mayoría de experimentos **el *Fairness* de un grafo decrece cuanto mayor sea su orden.** Se puede observar este comportamiento en todas las tablas expuestas, sin embargo, **existen dos excepciones**:

- ***Referencia Random:*** En este caso, el escenario *EPS-0.2-1-1* rompe la regla puesto que obtiene, en todas las posibles variaciones del experimento, un valor muy superior al del resto de **grafos**. La justificación de esto radica en el valor de la utilidad para este escenario ya que, el cálculo del *Fairness* se lleva a cabo a partir de las muestras obtenidas para la utilidad[22].
- ***Simulated Annealing:*** Al contrario que en el caso anterior, para el mismo escenario EPS, aquí el valor es menor incluso al del **grafo** con mayor orden. En este caso, la justificación de este comportamiento en base a los resultados obtenidos no ha sido concluyente.

Puesto que el análisis de resultados estaba especialmente orientado a los parámetros de *Utilidad y Tiempo de Ejecución*, las conclusiones de este último parámetro son ínfimas ya que cualquier relación establecida entre los dos parámetros anteriores ya ha sido establecida en los anteriores apartados.

[22] Véase la "Tabla 27 - Utilidad experimento 10% subnodos, Rnd, centralidad EigenVector, ordenación por nodos de mayor centralidad." Del Anexo para comprobar dichos valores de la utilidad.

5. Conclusiones

Desde un inicio este proyecto supuso un reto tanto por la cantidad de nuevos contenidos que el autor habría de encontrar como por el tiempo y el volumen de datos que sería necesario emplear y recopilar. En los primeros días, la toma de contacto se hizo un poco "cuesta arriba" puesto que se desconocían parte de las herramientas empleadas, no se tenía una base teórica sólida de algunos conceptos empleados y, en general, no se tenían aún definidos unos objetivos claros. Sin embargo, con algo de tiempo y esfuerzo, esta cuesta resultó en un proyecto claramente enriquecedor e interesante que ha permitido al autor expandir sus conocimientos y disfrutar gratamente de él, y con él, aún en los momentos más críticos.

Este TFG ha pretendido presentar un problema que resulta una extensión de investigaciones relacionadas con el comportamiento de redes complejas inalámbricas, concretamente, redes WiFi, ahondando en la perspectiva de la **optimización** mediante negociadores y con la adición de ciertas condiciones limitantes permitiendo así, abstraer escenarios reales a los modelos usados en las **simulaciones**. Gracias a ello, los experimentos llevados a cabo han permitido extraer una gran cantidad de conclusiones en varios aspectos referentes a los parámetros analizados, empero, han quedado bastantes posibilidades sin explorar y se cree que este podría ser un buen objeto de estudio para futuras investigaciones que quieran mejorar el comportamiento de este tipo de redes bajo las condiciones contempladas.

A pesar de que los datos obtenidos han lanzado resultados muy esperanzadores, de cara a los resultados esperados, cabe hacer cierto ahínco en aquellos puntos de digresión o resultados no esperados ya que podrían suponer grandes cambios en los resultados. Cabe destacar de ellos, el comportamiento general para el canal/color 6 en los distintos experimentos, como ya se ha mencionado este podría deberse a la matriz de interferencias empleada y, por ello, se cree que un análisis más profundo de este aspecto sería necesario para esclarecer definitivamente el porqué de lo observado. Finalmente, también podrían ampliarse los resultados añadiendo nuevos tipos de **centralidades** o negociadores, pero esto abriría un nuevo, y muy amplio, abanico de posibilidades.

Por otro lado, sería una ventaja sustancial ahondar la investigación relacionada con la **optimización** GPU de esta misma aplicación, puesto que permitiría acelerar todos los procesos envueltos en este TFG, desde la ejecución de los experimentos hasta el propio análisis de estos. Pese a ello y como ya se ha comentado, este proceso requerirá una gran cantidad de tiempo.

Para acabar esta sección se quiere recalcar el hecho de la oportunidad que ha supuesto para el autor este proyecto: en el ámbito académico, junto con el equipo de investigación, le ha permitido atisbar el mundo de la investigación y, a su vez, le ha permitido aplicar algunos de los conocimientos adquiridos a lo largo del Grado y su vida académica; por otro lado, en el ámbito personal, le ha permitido aprender y enfrentarse a algunos retos que, de no hacerlo, nunca habría podido llegar a enfrentar.

6. Pliego de Condiciones

6.1. Pliego de Condiciones Generales

El presente *Pliego de Condiciones* es el resumen de las características que se deberán cumplir en la instalación, ejecución y análisis del proyecto descrito en esta *Memoria*, así como también los dispositivos utilizados en el desarrollo y diseño de esta.

Para cualquier especificación fuera de las aquí contempladas, el autor no se responsabiliza de la no obtención del comportamiento esperado de las aplicaciones en dicho caso.

Para la obtención de un guion detallado de cómo instalar, reunir los requisitos software necesarios, ejecutar y analizar los resultados de los experimentos lanzados, véase el apartado *"Manual de Usuario".*

6.2. Pliego de Condiciones Técnicas

En esta subsección se exponen los requisitos técnicos (software y dispositivos) requeridos para llevar a cabo este proyecto de manera exacta al propio autor. A su vez, esta subsección estará escindida en dos apartados, cada uno referente al subsecuente requisito técnico.

6.2.1. Hardware

Debido a que el consumo computacional de los experimentos ha sido, en su mayoría, alto, se han utilizado tres dispositivos distintos para llevar a cabo el proyecto en su plenitud. Por un lado, para el desarrollo, testeo y **optimización** de código se ha empleado un dispositivo portátil, facilitando así la movilidad de este; por otro lado, las **simulaciones** de mayor peso, aquellas con una mayor carga computacional, se han llevado a cabo en el servidor remoto facilitado por el equipo de investigación ligado a este proyecto, ya que sus características se adaptaban perfectamente a las condiciones requeridas por cada experimento. Finalmente, se ha empleado un tercer dispositivo, una *Workstation* con inferiores características a las del servidor remoto, pero con mejores características a las del dispositivo portátil. Este último dispositivo se ha empleado, principalmente, para lanzar aquellos experimentos de menor carga computacional, evitando así la sobrecarga del servidor remoto, para testar partes del código y, especialmente, para probar todo lo relacionado con **optimización** GPU y automatizado de scripts.

Dispositivo	Información
Workstation HP Genérico 	***SO:*** *Windows 10 Pro.* ***CPU:*** *Intel® Core™ i7-2600 (3.40GHz).* ***GPU:*** *NVIDIA GForce GT 530 (2GB RAM).* ***RAM:*** *2x 4GB DDR3 (1600MHz).* ***DD:*** *Seagate Barracuda HDD (1TB-SATA3).* ***Pantalla 1:*** *Samsung HD 22" (1920x1080).* ***Pantalla 2:*** *HP LP2465 HD 24" (1920x1200).*
Dell Precission M6800 18" 	***SO:*** *Debian Jessie (8.4).* ***CPU:*** *Intel® Core™ i7-2720QM (2.20GHz).* ***GPU:*** *NVIDIA Quadro 1000M (2GB RAM).* ***RAM:*** *2x 4GB DDR3 (1600MHz).* ***DD:*** *Kingston HyperX Savage SSD (480GB-SATA3).* ***Pantalla 1:*** *Samsung HD 15" (1366x768).*
Servidor Remoto 	***SO:*** *Ubuntu (16.04).* ***CPU:*** *Intel® Core™ i7-7700 (3.60GHz).* ***GPU:*** *IntelHD Graphics 630 (3GB RAM).* ***RAM:*** *2x 16GB DDR4 (2400MHz).* ***DD:*** *SSD (240GB-SATA3).*

Tabla 15 – Resumen HW empleado.

6.2.2. Software

El software con el código ha venido dado por los scripts ya desarrollados por el equipo de investigación, por tanto, se ha mantenido para este trabajo. El resto de software, los sistemas operativos de los dispositivos personales y los procesadores de texto han sido elegidos como opción personal, aquella persona que desee reproducir o continuar estos experimentos es libre de usar otro software alternativo puesto que no afectará a la reproducción de dichos experimentos. A continuación, se expone un esquemático con el software empleado:

Software	Información
Windows 10 Pro	Sistema operativo instalado en el ordenador *Workstation***.** En él se ha trabajado con casi todas las fases del proyecto, desde documentación hasta testado y pruebas.
Debian Jessie (8.4)	Sistema operativo instalado en el dispositivo portátil, elegido por su alta disponibilidad de paquetes y la predisposición del autor por sistemas GNU/Linux.
Ubuntu 16.04	Sistema operativo instalado en el servidor remoto.

Anaconda 	Plataforma "OpenSource" para Python 2.7 y 3.0 que permite generar un entorno con todas las dependencias necesarias para llevar a cabo este TFG y más.
Spyder 	Entorno de desarrollo integrado (IDE) para Python que implementa un editor gráfico, consolas interactivas, explorador de variables… Muy útil para desarrollar código en este lenguaje.
Networkx	Módulo de aplicación o Framework de Python que permite crear, manipular y estudiar diferentes estructuras de red. Componente esencial, utilizado para la manipulación de los **grafos** de este TFG.
Numpy	Módulo de aplicación o Framework de Python que facilita realizar operaciones matemáticas de alta complejidad,
PyOpenCL	Módulo de aplicación o Framework de Python para la ejecución de código de Python sobre GPU.
XlsWriter	Módulo de aplicación o Framework de Python para generar hojas de Excel con formato. Usado para obtener las tablas de resultados de manera automática.
Office 365 	Conjunto de software Office que permite tener acceso a tres de las aplicaciones usadas en este TFG para documentación: Word, Excel y OneNote.

Tabla 16 – Resumen SW empleado.

7. Presupuesto

A continuación, se expone el presupuesto estimado del coste de ejecución de este proyecto. En él se incluyen precios que hacen referencia a productos semi-nuevos (todo aquel producto que tiene más de un año, desde su compra, y cuya amortización no ha expirado) y, a su vez, se incluyen los precios de las licencias necesarias para aquellos programas cuyo software requería del uso de estas.

Concepto	Descripción	Cantidad	Precio	Subtotal
Salario Ingeniero	Becario Ingeniería. Precio por mes bruto.	**6**	**333.334 €**	**2,000.00€**
Escritorio Genérico	PC + Escritorio HD 24''. Precio Ud. Seminuevo.	**1**	**1,000.00€**	**1,000.00**
Portátil	Dell Precision m6800. Precio Ud. Seminuevo.	**1**	**450.00€**	**450.00€**
Servidor Remoto	-	**1**	**2,000.00€**	**2,000.00€**
Office 365	Conjunto de programas Office. Precio licencia anual.	**1**	**149.00€**	**149.00€**
Windows 10 Pro	Sistema operativo. Software preinstalado.	**1**	**0.00€**	**0.00€**
Debian Jessie 8.4	Sistema operativo. Software instalado.	**1**	**0.00€**	**0.00€**
Ubuntu 16.04	Sistema operativo. Software instalado.	**1**	**0.00€**	**0.00€**
Networkx	Framework Python, estudio de **grafos**. Licencia BSD.	**1**	**0.00€**	**0.00€**
Numpy	Framework Python, matemática. Licencia BSD.	**1**	**0.00€**	**0.00€**
XlsWriter	Framework Python, conversión a Excel. Licencia BSD.	**1**	**0.00€**	**0.00€**
Anaconda	Entorno de Desarrollo Integrado Python. Licencia BSD.	**1**	**0.00€**	**0.00€**
Spyder	Entorno de Desarrollo Integrado Python. Licencia BSD.	**1**	**0.00€**	**0.00€**
PyOpenCL	Framework Python, GPU. Licencia BSD.	**1**	**0.00€**	**0.00€**
			Dto.: (2.67%)	**149.00€**
			Base Imponible	**5,450.01€**
			IVA: (21%)	**1,144.50€**
			Total	**6,594.51€**

NOTA: El descuento aplicado se debe a que la licencia para Office 365 ha sido obtenida a través del programa de alumnos de la UAH, por tanto, su valor ha sido costeado por dicha entidad.

8. Manual de Usuario

En este apartado se pretende dar las explicaciones necesarias para que cualquier usuario que desee utilizar la aplicación desarrollada para lanzar sus propios experimentos, pueda hacerlo sin problemas. Para ello, se describen una serie de instrucciones que permiten llevar a cabo la correcta instalación, el correcto uso, la configuración de distintos escenarios y el análisis de los resultados obtenidos.

8.1. Instalación

No existe un proceso de instalación como tal ya que no es necesario emplear un ejecutable (en caso de usar Windows) que instale las correspondientes DLL en nuestro sistema o un paquete del repositorio de paquetes (en caso de usar sistemas basados en GNU/LINUX). El único requisito es obtener la carpeta raíz *WiFi-IJCAI* con todos los scripts de Python necesarios y copiarla en el directorio deseado, con este objetivo y, dado que el proyecto se está realizando en un repositorio privado de *GitHub,* se recomienda hablar con alguno de los miembros que componen el equipo de investigación de este proyecto para obtener los permisos necesarios para obtener el conjunto de scripts.

Cabe destacar que la aplicación generada puede ser ejecutada en cualquier equipo con un sistema operativo que soporte Python 2.7, por este motivo, las instrucciones que vienen a continuación, así como las dependencias de la aplicación se han clasificado por sistemas con interfaz gráfico y sistemas sin interfaz gráfico de usuario, presuponiendo que el usuario tiene los conocimientos suficientes como para instalar las dependencias requeridas en su sistema para cada uno de ambos casos.

8.1.1. Dependencias

La descripción de cada una de las dependencias reflejadas en este subapartado ha sido realizada en el apartado *"Pliego de Condiciones→ Pliego de Condiciones Técnicas→Software"* de este trabajo, por tanto, en la siguiente tabla tan solo se refleja el nombre de la dependencia y para qué tipo de sistema es necesaria:

Dependencia	*Sistemas con IG*	*Sistemas sin IG*
Spyder	X	-
Anaconda	X	-
Networkx	*X*	*X*
Numpy	*X*	*X*
XlsWriter	*X*	*X*
Scripts Python	*X*	*X*

Tabla 17 – Dependencias para la instalación.

8.2. Directorios

Será necesario especificar o editar las rutas relativas a la carpeta con los **grafos** a emplear y la carpeta con los resultados generados al lanzar un experimento en los scripts *"launch_experiment.py" y "analizar_resultados.py".* Se recomienda usar las rutas indicadas por defecto, ya que permiten trabajar con todas las carpetas desde un mismo directorio

A continuación, se muestra la configuración por defecto de carpetas, así como las líneas a modificar en cada script para poder indicar la ruta de cada una de estas carpetas en caso de no querer usar la configuración por defecto:

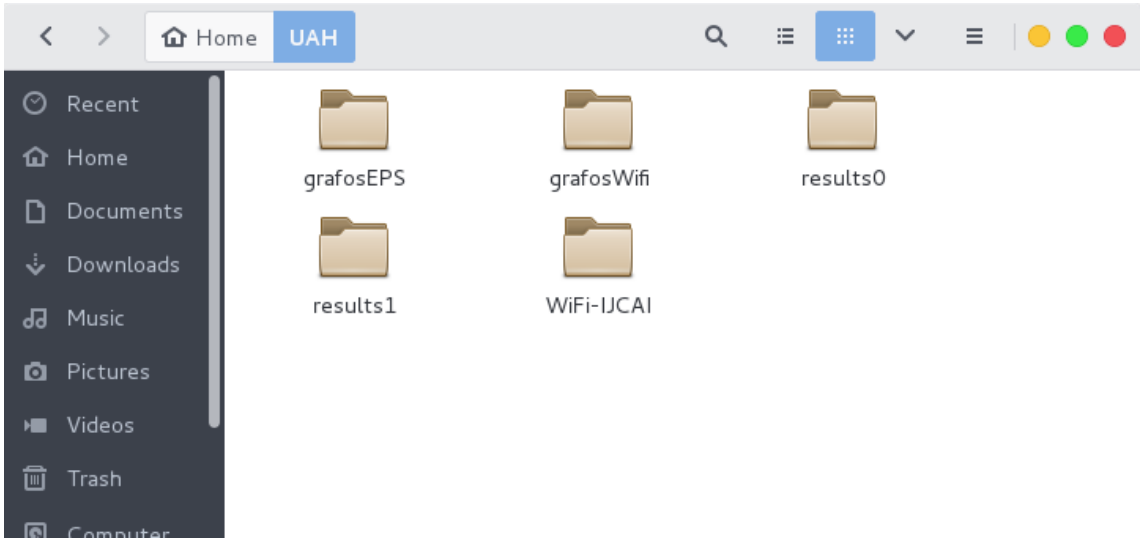


Imagen 23 – Directorio con la configuración de carpetas por defecto.

En la captura se pueden observar cinco carpetas, las dos primeras, "***grafos**EPS*" y "***grafos**Wifi*", conforman las carpetas contenedoras de los **grafos**; las carpetas "*results**" contienen resultados de algunos experimentos ya lanzados y, para acabar, la carpeta "*WiFi-IJCAI*" contiene todos los scripts ya citados.

```
EXPERIMENT_NAME='20170528ref'
graph_path='../grafosEPS/'
results_path='../results/'+EXPERIMENT_NAME+'/'
time_path= '../results/' +'TimeRes'+'/'
```

Cuadro de Texto 17 – Sentencias por defecto para configurar directorios en "launch_experiment.py".

```
graph_path='../grafosEPS/'
results_path='../results21/'+fecha+'ref/'
excel_workbook = xls.Workbook('../results/analise_results.xlsx')
time_path= '../results/' +'TimeRes'+'/'
```

Cuadro de Texto 18 – Sentencias por defecto para configurar directorios en "launch_experiment.py".

8.3. Sistemas con *GUI*

Para este tipo de sistemas es altamente recomendable instalar las dependencias de *Anaconda y Spyder* (este último viene integrado en la primera dependencia, aunque puede ser instalado de manera independiente), puesto que facilitaran de sobremanera la edición de los archivos/scripts que configuran los distintos escenarios, evitaran instalar los módulos de Python requeridos de forma manual… En caso de no querer instalar estas herramientas, se ruega al usuario que siga las instrucciones del siguiente apartado "*Sistemas sin GUI*".

8.4. Sistemas sin *GUI*

En este tipo de sistemas, editores como *Spyder* no tienen cabida, por tanto, se recomienda al usuario instalar las dependencias necesarias para que el sistema soporte *Python 2.7*, así como los módulos ya mencionados en el apartado *"Dependencias"* de manera manual. Una vez realizada esta tarea, la edición de los archivos/scripts será llevada a cabo con el editor que el usuario tenga implementando o acostumbre a usar en su sistema[23].

8.5. Configuración del escenario

Para obtener las condiciones o limitaciones requeridas, dentro del abanico ya descrito anteriormente, es necesario configurar varios parámetros para su correcto funcionamiento dentro del script *"launch_experiment.py"*. Aparte de los directorios ya citados, es posible configurar:

- **Número de Experimentos:** Permite elegir la cantidad de experimentos deseada para obtener resultados de un mismo escenario, de manera que se puedan obtener parámetros como la media, la desviación típica o los intervalos de confianza de algunos parámetros (utilidad, Fairness, tiempo ejecución...). Por defecto, el valor es 20, ha de ser un entero.
- **Número de Iteraciones:** Se trata de la cantidad de iteraciones o "vueltas" que ha de dar el mecanismo de negociación seleccionado dentro del bucle de su algoritmo. Por defecto el valor es 3000, ha de ser un entero.
- **Temperatura:** Parámetro de configuración para el algoritmo *"Simmulated Annealing"*, permite regular la probabilidad con la que un contrato es aceptado en caso de ser peor que el anterior[24]. Por defecto su valor es igual a la unidad, puede ser de tipo *Float.*
- **Número de proveedores:** Por defecto su valor es cuatro, ha de ser un entero.
- **Número de colores:** Por defecto su valor es 11, ha de ser un entero.
- **Modo de selección de subnodos:** Permite elegir entre distintas **centralidades** y un modo aleatorio, su valor por defecto es cero y ha de ser un entero contemplado dentro del rango de modos disponibles [0-5].
- **Porcentaje de Subnodos:** Permite elegir qué porcentaje del total de puntos de acceso formarán parte de las **restricciones** de color. Su valor por defecto es "0.1", tipo *Float.*
- **Modo de Selección de color:** Permite elegir cómo se van a asignar colores al conjunto de subnodos. Su valor por defecto es cero, ha de ser un entero contemplado dentro del rango de modos disponibles [0,1].
- **Color para modo Fijo:** En caso de seleccionar en el anterior parámetro el modo de *color fijo*, este parámetro establece el valor de dicho color. Acepta valores de tipo entero, siempre que estén dentro del rango definido por el parámetro *número de colores.*
- **Modo de selección de negociador:** Permite seleccionar la técnica de negociación automática a ser empleada. De nuevo, ha de ser un entero contemplado en el rango [0-3] y su valor por defecto es cero.
- **Modo de Ordenación:** En caso de elegir una **centralidad** como modo de selección de nodos, es posible ordenarlos con distintos métodos. Su valor por defecto es cero, ha de ser un entero.

[23] El autor ha empleado el editor "*Vi*".

[24] Para más información, véase el apartado "*Negociadores Automáticos →Annealer".*

- **Lista de grafos:** Permite definir la lista de **grafos** que van a ser empleados, se trata de una lista de Python.

Como se puede observar, hay una gran variedad de posibilidades dado el conjunto de parámetros a configurar, sin embargo, la modificación de estos parámetros es tremendamente trivial. A continuación, se exponen capturas con las sentencias para la modificación de dichos parámetros, así como ejemplos de la ejecución de distintos experimentos en sistemas con GUI y sin GUI:

```
39
40 EXPERIMENT_NAME='20170528ref'
41 graph_path='../grafosEPS/'
42 results_path='../results4/'+EXPERIMENT_NAME+'/'
43 num_runs =20
44 iterations=3000
45 temperature= 1
46 nP=4
47 num_colors=11
48
49 """COLORING AND NEGOTIATORS PARAMETERS"""
50 #Node Selection Mode --> [0:Random, 1:Degree Centrality, 2:Closeness Cent., 3:Betweeness Cent., 4:EigenVector Cent., 5:Communicability Cent.]
51 nodeSelMode = 4
52 #Node percentage to have a fixed color, set to 0 if not needed
53 nodePercentage = 0.3
54 #Color Selection Mode --> [0:Fixed Color, 1:Random Color, 2:TSC_DSATUR(not yet implemented)]
55 colorSelMode = 0
56 #Fixed Color Value (use in case of Fixed Color mode selected, contemplated values [1-11])
57 fixedColor_Val = 11
58 #Negotiator Selection Mode --> [0:Random, 1:Simmulated_Annealing, 2:Hill_Climber, 3:LCCS]
59 negotiatorSelMode = 0
60 #Sorting Method (not available/usable for random selection node mode) --> [0:More Central Nodes(Descending Order), 1:Less central(Ascending), 2:Normal Dist.]
61 nodeSortMethod = 2
62
63 #grafosList=[(0.2,1,1),(0.25,1,1),(0.25,2,1),(0.25,3,1),(0.5,1,1),(0.5,2,1),(0.5,3,1),(0.75,1,1),(0.75,2,1),(0.75,3,1),(1,1,1),(1,2,1),(1,3,1)]
64 grafosList=[(0.25,1,1),(0.25,2,1),(0.25,3,1),(0.5,1,1),(0.5,2,1),(0.5,3,1)]
65 #grafosList=[(0.25,1,1),(0.25,2,1),(0.25,3,1)]
66 time_path= '../results4/' +'TimeRes'+'/'
67
```

Imagen 24 – Parámetros para la configuración del escenario.

8.6. Ejecución

Para ejecutar la aplicación con la configuración establecida por el usuario en el script "*launch_experiment.py*", existen dos modos:

- **Usando *Spyder:*** Para ejecutar scripts desde este entorno no es necesario más que pulsar la tecla *F5*.
- **Terminal:** El script se ejecuta como cualquier otro script de Python, a priori, no es necesario pasarle parámetros (existe la opción *verbose*, en caso de querer usarla ha de pasarse por parámetro al script). Así pues, bastaría con ejecutar la siguiente sentencia:

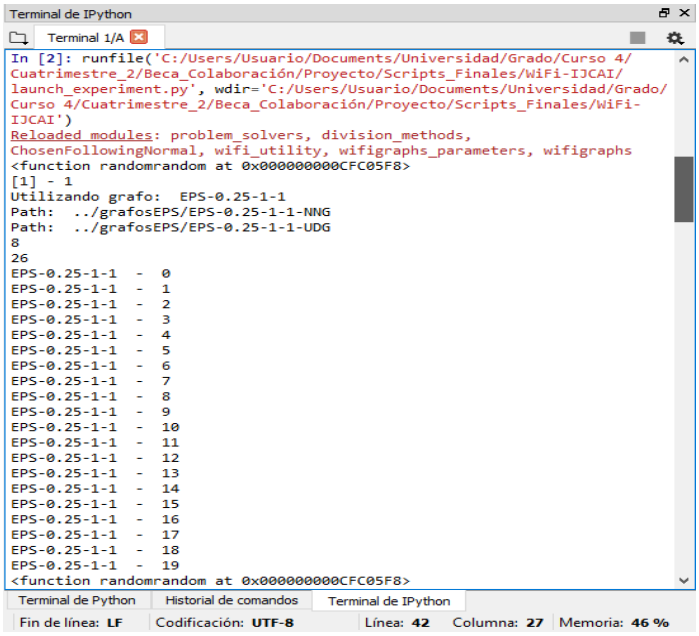
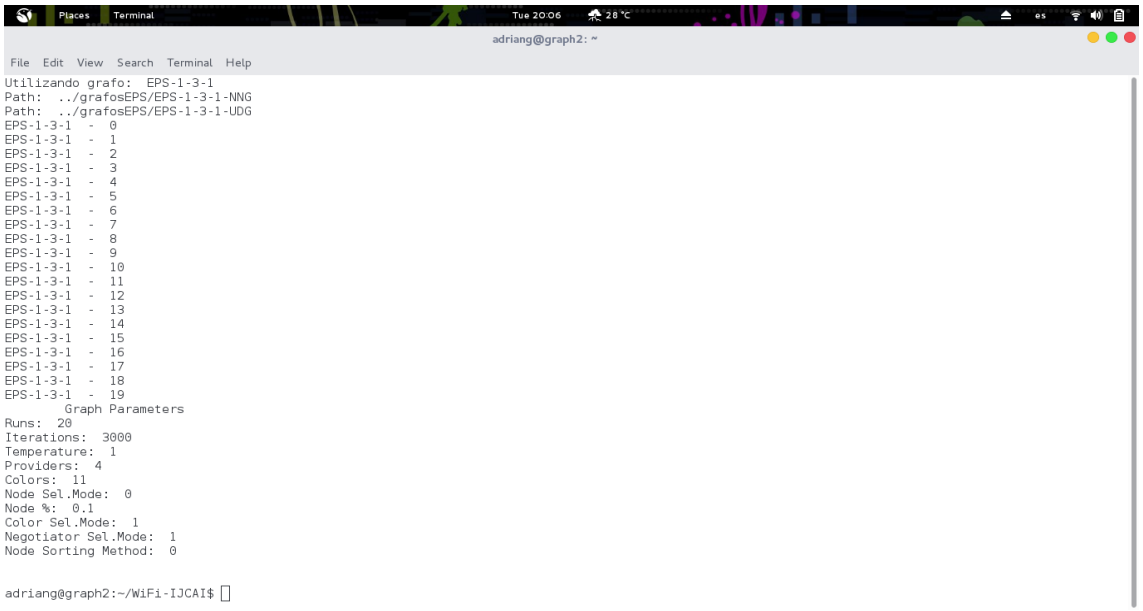


Imagen 25 – Inicio de la ejecución del script "launch_experiment.py" en un sistema con GUI desde Spyder.

```
python launch_experiment.py
```

Cuadro de Texto 19 - Ejecución de scripts en terminal.

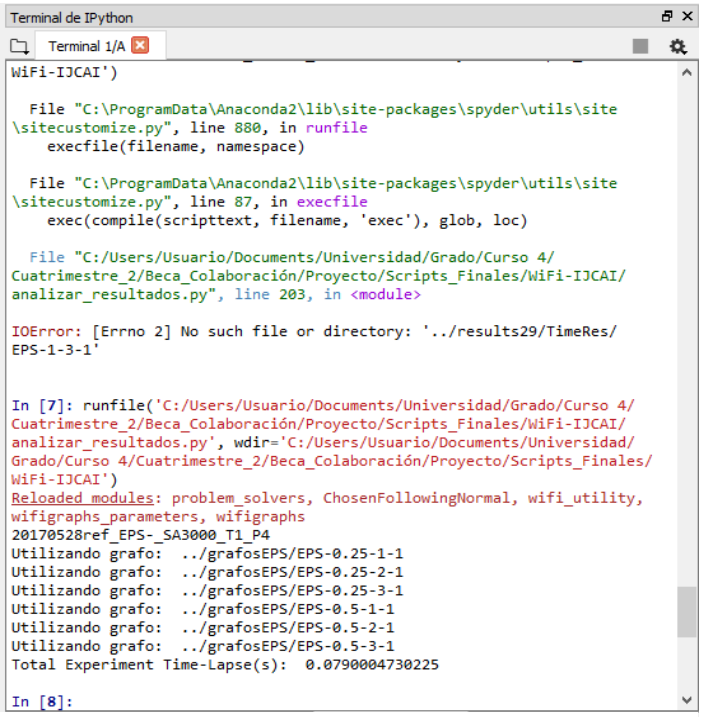


Cuadro de Texto 20 – Fin de la ejecución del script "launch_experiment.py" en un sistema sin GUI desde el terminal.

8.7. Obtención de resultados

Una vez terminada la ejecución de los distintos experimentos, en el directorio raíz se podrán observar las carpetas correspondientes a los resultados de cada uno de ellos (por defecto estas carpetas tendrán el nombre *results* más un número). Dentro de cada una de estas carpetas se podrán observar otras dos carpetas, la primera de ellas contendrá en su nombre una fecha, esta será la carpeta con los resultados relativos a los propios experimentos, de ella se podrá extraer la información necesaria para llevar a cabo cálculos como la utilidad o el Fairness. La segunda carpeta tendrá el nombre, por defecto, "*timeRes*" y de ella se podrán extraer los datos necesarios para reflejar los tiempos de ejecución de cada uno de los experimentos.

Este proceso puede parecer difícil, sin embargo, el script "*analizar_resultados.py*" permite llevar a cabo todos los cálculos requeridos de manera automática y los representa en una tabla de Excel. La única configuración necesaria en este script es la referida en el apartado "*Directorios*", es decir, tan solo es necesario modificar los paths a las carpetas para obtener los datos de los parámetros por defecto de este script, esto son: media de la utilidad, desviación estándar de la utilidad, intervalos de confianza de la utilidad, media del Fairness, desviación estándar del Fairness, tiempo medio de cada experimento y tiempo total



Cuadro de Texto 21 – Salida del script "Analizar_resultados.py" desde Spyder.

de todos los experimentos. En caso de querer realizar cálculos diferentes a los citados, el propio usuario será el encargado de implementarlos. Al terminar la ejecución de este script, se obtendrá en el path indicado, un libro de Excel con los resultados obtenidos. El formato de dicho libro de Excel puede ser modificado y adaptado a las necesidades requeridas.

Para finalizar, se recomienda a todo usuario que quiera almacenar los resultados que siga el convenio de nombres establecido por el autor para este mismo propósito en el archivo "*README*", este es el siguiente:

<node%>.<node_selection_mode>.[info1].<color_sel_mode>.[info2].<negotiator_technique>

Siendo los campos "<>" de carácter obligatorio, los campos "*[]*" de carácter opcional y los posibles valores de cada campo: **Porcentaje de nodos:** *10,20 o 30.*

- **Modo de selección de nodos:** *"Rnd" (Random), "DC" (Degree Cent.), "BC" (Betweenness Cent.) , "CC" (Closeness Cent.), "EC" (EigenVector Cent.) y "CMC" (Communicability Cent.).*
- **Info1:** Campo para la información referente a el método de ordenación, posibles valores: *"+D" (mayor centralidad), "-D" (menor centralidad) y "ND" (distribución normal).*
- **Modo de selección de color:** *"FC" (Color Fijo) y "Rnd" (Color aleatorio).*
- **Info2:** Este campo indica el valor del canal/frecuencia asignada en caso de haber elegido el modo *FC.*
- **Negociador Automático: "***Rnd" (Random), "SA" (Simulated Annealing), "HC" (Hill Climber) y "LCCS" (Least Congested Channel).*

9. Referencias

Esta sección tiene como único objetivo facilitar el código de las funciones desarrolladas por el autor para todo aquel lector que quiera utilizarlo para continuar este proyecto o emprender proyectos independientes de este y también como muestra o guiño hacia la filosofía de la *Free Software Foundation*. El resto de código desarrollado no se incluye puesto que incluiría parte del código implementado por el equipo de investigación, código del cual el autor de este trabajo no tiene autoría o derechos como para difundirlo.

9.1. Negotiator selection

```
def negotiator_selection(**kwargs):
    I= kwargs['I']
    G= kwargs['G']
    apList= kwargs['apList']
    pList = kwargs['pList']
    nColors = kwargs['nColors']
    colorList = kwargs['colorList']
    sub_apList = kwargs['sub_apList']
    negotiatorSel_Mode = kwargs['negotiatorSel_Mode']
    iterations = kwargs['iterations']
    temperature = kwargs['temperature']
    overlap = kwargs['overlap']

    #Random
    if  negotiatorSel_Mode == 0:
        colorear_a_mano(I,G,apList,colorList)
        return colorList

    #Simulated_Annealing
    elif negotiatorSel_Mode == 1:
      colorList =
      ps.annealer_mediator(colorList=colorList,sub_apList=sub_apList,I
      =I,G=G,apList=apList,pList=pList, nColors=nColors,
      temperature=temperature, iterations=iterations)
        return colorList

    #Hill_Climber
    elif negotiatorSel_Mode == 2:
      colorList =
      ps.hill_climber_mediator(colorList=colorList,sub_apList=sub_apLi
      st,I=I,G=G,apList=apList,pList=pList, nColors=nColors,
      iterations=iterations)
        return colorList

    #LCCS
    elif negotiatorSel_Mode == 3:
      colorList =
      ps.reference_LCCS(colorList=colorList,sub_apList=sub_apList,I=I,
      G=G,apList=apList,nColors=nColors,iterations=iterations,overlap=
      overlap)          #I,G,apList,nColors,iterations,overlap
        return colorList
    else:
      print "Node ordering mode(%d) not available" %
      negotiatorSel_Mode
      print "Please introduce a valid mode number[0-2], programm will
      exit now"
      sys.exit()
```

9.2. Colorlist Generator

```
def
colorList_Generator(iterations=None,temperature=None,overlap=None,**kw
args):

    I= kwargs['I']
    G= kwargs['G']
    apList= kwargs['apList']
    nColors = kwargs['nColors']
    sub_apList = kwargs['sub_apList']
    colorSel_Mode = kwargs['colorSel_Mode']
    negotiatorSel_Mode = kwargs['negotiatorSel_Mode']
    fixedColor_val = kwargs['fixedColorVal']
    #Negotiator variable assignation
    pList = kwargs['pList']

    nNodes=len(apList)

    ##print "Number of Nodes: ",nNodes
    ##print "Fixed Color Value: ",fixedColor_val

    #Random Color list
    colorList=np.random.randint(nColors, size=nNodes)+1
    ##print "Lista Colores1: ",colorList

    #Fixed Color
    if colorSel_Mode == 0:
        for i in range(len(apList)):
                  if apList[i] in sub_apList:
    #In case of being part of the subnode list
                colorList[i] = fixedColor_val

    #Random Color
    elif colorSel_Mode == 1:
        for i in range(len(apList)):
            if apList[i] in sub_apList:
                 color=random.randrange(1,12)
                 colorList[i] = color
                #print color

    elif colorSel_Mode == 2:
        print "TODO DSATUR_modificado.greedy_color_wicN()"
        print "Mode not yet implemented, programm will exit now"
        exit()

    else:
      print "Color selection mode(%d) not available" % colorSel_Mode
      print "Please introduce a valid mode number [0-2], programm will
      exit now"
      sys.exit()

    colorList =
    negotiator_selection(I=I,G=G,apList=apList,pList=pList,nColors=nCo
    lors,colorList=colorList,sub_apList=sub_apList,negotiatorSel_Mode=
    negotiatorSel_Mode,temperature=temperature,iterations=iterations,o
    verlap=overlap)
    return colorList
```

9.3. Node Selection

```
def node_selection(I,apList,node_sel_mode,percentage,nodeSortMethod):

    nap=round(len(apList)*percentage)
    ##print "N°Subnodos: ",nap
    node_count = 0
    sub_apList = []

    #Random
    if node_sel_mode == 0:
        for i in
np.random.randint(0,high=len(apList),size=len(apList)):
            ##print i
            if apList[i] in sub_apList:
                ##print "Element already exists"
                continue
            else:
                ##print "element append"
                sub_apList.append(apList[i])
                node_count += 1
                if node_count >= nap:
                    ##print "Node Count Break"
                    break
                continue
        return sub_apList

    #Degree Centrality
    elif node_sel_mode == 1:
        Nodes = nx.degree_centrality(I)
        ##print "Degree Centrality"

    #Closeness Centrality
    elif node_sel_mode == 2:
        Nodes = nx.closeness_centrality(I)
        ##print "Closeness Centrality"

    #Betweeness Centrality
    elif node_sel_mode == 3:
        Nodes = nx.betweenness_centrality(I)
        ##print "Betweeness Centrality"

    #EigenVector Centrality
    elif node_sel_mode == 4:
        Nodes = nx.eigenvector_centrality(I)
        ##print "EigenVector Centrality"

    #Communicability Centrality
    elif node_sel_mode == 5:
        Nodes = nx.communicability_centrality(I)
        ##print "Communicability Centrality"

    else:
      print "Node selection mode(%d) not available" % node_sel_mode
       print "Please introduce a valid mode number[0-4], programm
      will exit now"
      sys.exit()

    sub_apList = order_nodes(Nodes,apList,nap,nodeSortMethod)
    return sub_apList
```

9.4. Order Nodes

```
def order_nodes(Nodes,apList,nap,nodeSortMethod):
    nodeList = Nodes.items()
    auxapList = []
    sub_apList = []
    inap = int(nap)
    #print inap
    #print len(apList)

    for i in range(len(nodeList)):
        if nodeList[i][0] in apList:
            auxapList.append(nodeList[i])

    #Descendente
    if nodeSortMethod == 0:
        auxapList.sort(key=lambda x: x[1], reverse=True)
        ##print "Lista ordenada",auxapList
        for i in range(len(auxapList)-inap):
            auxapList.pop()
        ##print "Lista con tupla de subnodos/centralities
ordenados",auxapList
        for i in range(len(auxapList)):
            sub_apList.append(auxapList[i][0])
        ##print "Lista subnodos: ",sub_apList
        return sub_apList

    #Ascendente
    elif nodeSortMethod == 1:
        auxapList.sort(key=lambda x: x[1])
        #print "Lista ordenada",auxapList
            for i in range(len(auxapList)-inap):
            auxapList.pop()
        ##print "Lista con tupla de subnodos/centralities
ordenados",auxapList
        for i in range(len(auxapList)):
            sub_apList.append(auxapList[i][0])
        ##print "Lista subnodos: ",sub_apList
        return sub_apList

    #Distribución Normal
    elif nodeSortMethod == 2:
        valueList = []
        aux = []
        for i in range(len(auxapList)):
            valueList.append(auxapList[i][1])
        ##print "VALUELIST: ",valueList
        aux = cfn.normal_choice(valueList,inap)
        ##print "AUX: ",aux
        for i in range(len(auxapList)):
            if auxapList[i][1] in aux:
                sub_apList.append(auxapList[i][0])

        ##print "Lista subnodos: ‟,sub_apList
        ##print "Normal Distribution"
        return sub_apList

    else:
        print "Node ordering mode(%d) not available" % nodeSortMethod
        print "Please introduce a valid mode number[0-2], programm
will exit now"
        sys.exit()
```

10. Bibliografía

[1] Marsá Maestre, I.; Hoz de la Hoz, E.; Gimenez Guzmán, J.M.; Orden Martín, D.; Klein , M. "Nonlinear negotiation approaches for complex-network optimization: a study inspired by Wi-Fi channel assignment". In Proceedings of the 2nd International Congress, "Conflict Resolution and Decision Making (COREDEMA), Deen Haag, Congreso. Internacional. "2nd Worksop on Conflict Resolution and Decision Making (COREDEMA 2016)". Den Haag, Holland, Netherlands, Europe, 29.30 August 2016; pp. 1-6.

[2] De La Hoz, E., Marsa-Maestre, I., Gimenez-Guzman, J. M., Orden, D., & Klein, M. (2017, May). *Multi-Agent Nonlinear Negotiation for Wi-Fi Channel Assignment. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (pp. 1035-1043)*. International Foundation for Autonomous Agents and Multiagent Systems.

[3] de la Hoz, E., Gimenez-Guzman, J. M., Marsa-Maestre, I., & Orden, D. (2015). *Automated negotiation for resource assignment in wireless surveillance sensor networks. Sensors, 15(11), 29547-29568.*

[4] Garey, M. R.; Johnson D. S. *"The Complexity of Near-Optimal Graph Coloring," Proc. JACM vol.23, pp. 43-49. January, 1976* (New York, NY, USA).

[5] D. Brelaz, "New methods to color vertices of a graph," *Communications of ACM*, vol. 22, pp. 251–256, 1979 (New York, NY, USA).

[6] Orden Martín, D.; Marsá Maestre, I.; Gimenez Guzmán, J.M.; Hoz de la Hoz, E. *Spectrum graph coloring and applications to WiFi channel assignment [online] Cornell University Library, pp.1-22.* Disponible en: https://arxiv.org/abs/1602.05038

[7] Ackley, D.A. *Connectionist Machine for Genetic Hill-Climbing. Kluwer Academic Publishers,Bostons,1987.*

[8] David, P.; Kuipers, B. Learning hill-climbing functions as a strategy for generating behaviors in mobile robots. Technical report AI90-137, University of Texas at Austin, 1990.

[9] Gibbons, P.; Mathon R. *The use of hill-climbing to construct orthogonal Steiner triple systems. Technical report 263/92,* University of Toronto, 1992.

[10] Winston, P.H. *Artificial Intelligence. Addison Wesley Publishing Company, Reading MA Third Edition, 1992.*

[11] Achanta, M. "Method and Apparatus for Least Congested Channel Scan for Wireless Access Points," US Patent No. 20060072602, Apr. 2006.

[12] Mi, P., & Wang, X. (2012). "Improved channel assignment for WLANs by exploiting partially overlapped channels with novel CIR-based user number estimation". In IEEE, international conference on communications (ICC) (pp. 6591–6595).

[13] *K. K. Leung and B.-J. Kim, "Frequency Assignment for IEEE 802.11 Wireless Networks," in Proc. IEEE Vehicular Technology Conference, vol. 3, pp. 1422–1426, Oct. 2003.*

[14] C. Hua and R. Zheng, "Starvation Modeling and Identification in Dense 802.11 Wireless Community Networks," Proc. IEEE INFOCOM, 2008.

[15] A.Mishra and V. Shrivastava, D. Agrawal, S. Banerjee, and S. Ganguly, "Distributed Channel Management in Uncoordinated Wireless Environments," in Proc. International Conference on Mobile Computing and Networking, pp. 170–181, 2006.

[16] S. S. Pablo, "A new DSATUR-based algorithm for exact vertex coloring" [online], Centro de Automática y Robótica (CAR), José Abascal 2, 28006, Madrid, España. Disponible en: https://pdfs.semanticscholar.org/c018/e239f80389879af5eb91c83c793078da33a4.pdf

[17] D., Brèlaz, "new Methods to color three vertices of a graph", Communications of the Assoc. of Comput. Machinery 22 (1979), 251-256.

[18] Beauchamp, M. A. (1965). «An improved index of centrality». Systems Research and Behavioral Science 10 (2): 161-163.

[19] Brandes, U. *On Variants of Shortest-Path Betweenness Centrality and their Generic Computation*. Social Networks 30(2), pp. 136-145 (2008).

[20] Estrada E.; Rodriguez-Velazquez J. A. "Subgraph centrality in complex networks", Physical Review E 71, 056103 (2005).

[21] S. W. K. Ng and T. H. Szymanski. Interference measurements in an 802.11n Wireless Mesh Network testbed. In Electrical Computer Engineering (CCECE), 2012. 25th IEEE Canadian Conference on, pages 1–6, April 2012.

11. Anexo

	Random															
	Fixed Color												Random			
	1				6				11				-			
	U_Norm	STD	Fairness Fujita		U_Norm	STD	Fairness Fujita		U_Norm	STD	Fairness Fujita		U_Norm	STD	Fairness Fujita	
	-	-	Avg	STD	-	-	Avg	STD	-	-	Avg	STD	-	-	Avg	STD
EPS-02-1	0.331281	0.086463	0.136420	0.031923	0.364470	0.080020	0.126506	0.017110	0.303453	0.047233	0.134253	0.017105	0.351646	0.351646	0.137036	0.021053
EPS-0.25-1	0.205073	0.032890	0.088384	0.016130	0.226997	0.046585	0.092179	0.016178	0.212293	0.043838	0.086614	0.017140	0.207917	0.041239	0.086186	0.019219
EPS-0.25-2	0.198526	0.030253	0.083064	0.008927	0.179790	0.050620	0.078836	0.019964	0.192825	0.053894	0.082136	0.016421	0.183648	0.049636	0.078089	0.015148
EPS-0.25-3	0.199780	0.025057	0.085752	0.007767	0.231216	0.048432	0.097887	0.010153	0.244054	0.041202	0.100391	0.011927	0.214061	0.044913	0.093529	0.018641
EPS-0.5-1	0.159075	0.042898	0.069727	0.019119	0.174386	0.039415	0.072638	0.014133	0.154708	0.038569	0.066936	0.015080	0.139447	0.043224	0.077918	0.015793
EPS-0.5-2	0.176621	0.050162	0.075435	0.020683	0.167668	0.047319	0.072132	0.021708	0.156983	0.022337	0.067251	0.013450	0.163239	0.039895	0.070220	0.016005
EPS-0.5-3	0.170140	0.053837	0.074927	0.017700	0.179774	0.049036	0.083269	0.021637	0.191181	0.051330	0.083855	0.018044	0.171377	0.046954	0.079262	0.018211
EPS-0.75-1	0.172305	0.048995	0.077603	0.018822	0.164360	0.043428	0.071058	0.016571	0.182634	0.029564	0.079689	0.011037	0.179720	0.039962	0.078229	0.015370
EPS-0.75-2	0.186876	0.049440	0.081949	0.021883	0.182439	0.034177	0.076603	0.012111	0.189638	0.049809	0.078806	0.015832	0.219299	0.031277	0.090107	0.010582
EPS-0.75-3	0.193775	0.037333	0.086134	0.013092	0.183672	0.063134	0.079313	0.022721	0.177925	0.047115	0.079068	0.017571	0.180114	0.047357	0.081616	0.018761
EPS-1-1	0.136414	0.031574	0.063169	0.013085	0.144277	0.054213	0.064911	0.021687	0.120160	0.019185	0.055915	0.007815	0.143744	0.039442	0.067047	0.015761
EPS-1-2	0.144124	0.046849	0.082093	0.017507	0.160606	0.033454	0.071510	0.012115	0.146664	0.037212	0.067766	0.016893	0.134174	0.030054	0.068681	0.011789
EPS-1-3	0.169380	0.030735	0.075560	0.012959	0.149321	0.048708	0.068566	0.020735	0.142912	0.035653	0.063125	0.013085	0.149310	0.041593	0.062618	0.015795

Imagen 26 - Tabla de resultados para versión Beta 0.1.

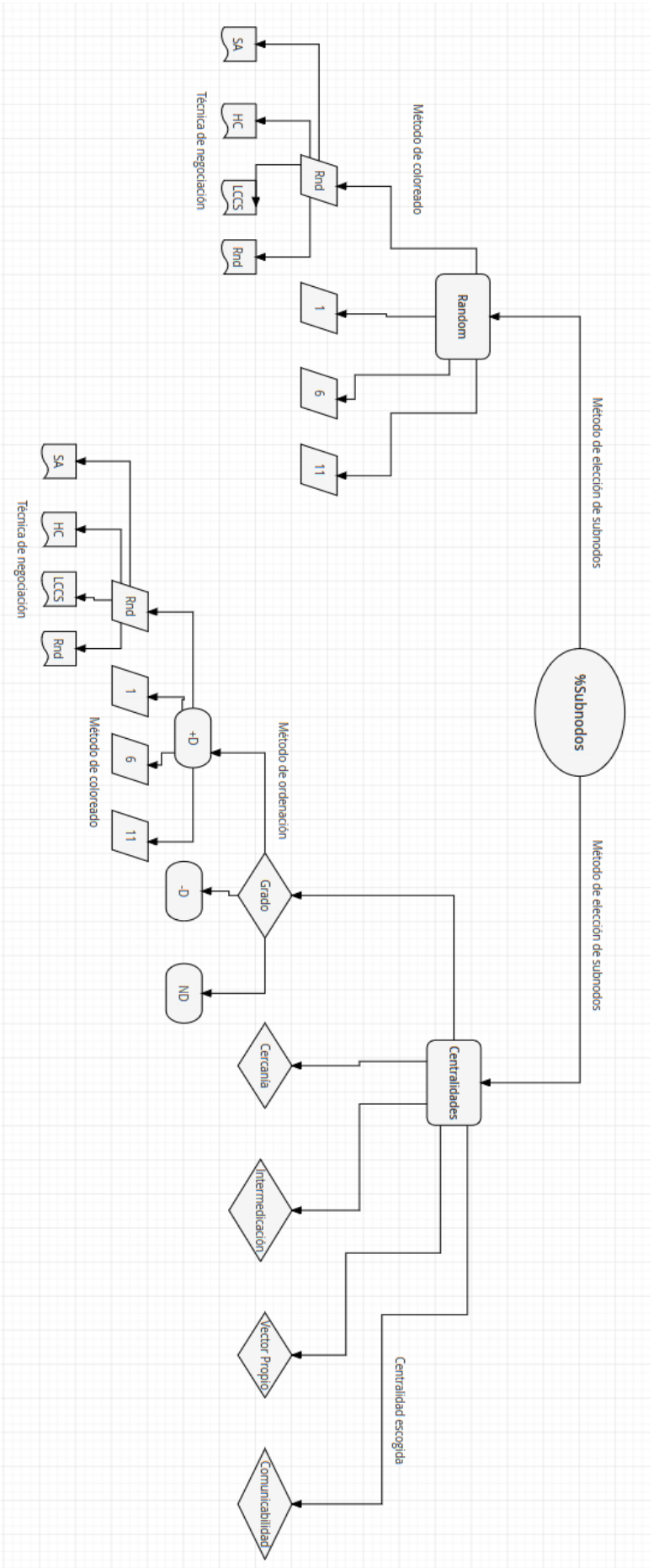


Imagen 27 – Esquema gráfico con todas las posibilidades para desarrollar un escenario.

%subnodos/ Negociador	0%		10%		20%		30%	
-	Utilidad	IC	Utilidad	IC	Utilidad	IC	Utilidad	IC
Random	0.147	0.014	0.143	0.016	0.156	0.016	0.151	0.019
LCCS	0.324	0.015	0.315	0.015	0.331	0.011	0.260	0.008
HC	0.358	0.014	0.341	0.015	0.305	0.009	0.293	0.013
SA	0.485	0.008	0.408	0.008	0.379	0.008	0.294	0.005

Tabla 18 - Utilidad para distintos Negociadores sobre EPS-1-1 con color fijo 1, centralidad EigenVector y ordenación mediante distribución normal.

%subnodos/ Negociador	0%		10%		20%		30%	
-	Utilidad	IC	Utilidad	IC	Utilidad	IC	Utilidad	IC
Random	0.166	0.019	0.156	0.020	0.136	0.022	0.131	0.017
LCCS	0.366	0.02	0.334	0.018	0.270	0.015	0.280	0.019
HC	0.420	0.018	0.391	0.016	0.331	0.012	0.318	0.013
SA	0.482	0.014	0.439	0.006	0.374	0.005	0.349	0.007

Tabla 19- Utilidad para distintos Negociadores sobre EPS-0.75-2 con color fijo 6, centralidad EigenVector y ordenación mediante nodos más centrales.

%subnodos/ Negociador	0%		10%		20%		30%	
-	Utilidad	IC	Utilidad	IC	Utilidad	IC	Utilidad	IC
Random	0.186	0.025	0.181	0.020	0.130	0.015	0.117	0.013
LCCS	0.371	0.018	0.326	0.023	0.270	0.015	0.226	0.006
HC	0.428	0.015	0.380	0.016	0.288	0.019	0.241	0.009
SA	0.485	0.006	0.445	0.006	0.330	0.006	0.271	0.004

Tabla 20 - Utilidad para distintos Negociadores sobre EPS-0.5-3 con color fijo 11, centralidad EigenVector y ordenación mediante nodos menos centrales.

%subnodos/ Negociador	0%		10%		20%		30%	
-	Utilidad	IC	Utilidad	IC	Utilidad	IC	Utilidad	IC
Random	0.216	0.024	0.210	0.024	0.221	0.025	0.210	0.021
LCCS	0.452	0.020	0.437	0.023	0.371	0.028	0.342	0.021
HC	0.478	0.016	0.401	0.018	0.401	0.018	0.373	0.022
SA	0.549	0.009	0.536	0.011	0.447	0.017	0.397	0.018

Tabla 21 - Utilidad para distintos Negociadores sobre EPS-0.25-3 con color aleatorio, centralidad EigenVector y ordenación mediante nodos más centrales.

%subnodos/ Negociador	0%		10%		20%		30%	
-	Utilidad	IC	Utilidad	IC	Utilidad	IC	Utilidad	IC
Random	0.149	0.011	0.160	0.014	0.158	0.020	0.140	0.018
LCCS	0.328	0.018	0.322	0.014	0.320	0.015	0.252	0.013
HC	0.378	0.014	0.339	0.013	0.350	0.014	0.311	0.010
SA	0.433	0.008	0.409	0.008	0.387	0.010	0.346	0.007

Tabla 22- Utilidad para distintos Negociadores sobre EPS-1-2 con color fijo 11, centralidad EigenVector y ordenación mediante distribución normal.

Escenario	U_Norm	STD	IC	Fairness Fujita		Time(s)
-	-	-	-	Avg	STD	Avg
EPS-02-1	0.693	0.012	0.005	0.100	0.003	71.695
EPS-0.25-1	0.522	0.020	0.009	0.123	0.007	264.940
EPS-0.25-2	0.525	0.012	0.005	0.110	0.004	256.716
EPS-0.25-3	0.504	0.014	0.006	0.126	0.003	288.757
EPS-0.5-1	0.453	0.022	0.010	0.106	0.005	804.314
EPS-0.5-2	0.465	0.020	0.009	0.118	0.004	795.634
EPS-0.5-3	0.477	0.014	0.006	0.128	0.003	768.082
EPS-0.75-1	0.418	0.014	0.006	0.117	0.002	1576.281
EPS-0.75-2	0.463	0.022	0.010	0.126	0.003	1525.414
EPS-0.75-3	0.378	0.013	0.006	0.123	0.004	1461.479
EPS-1-1	0.393	0.019	0.009	0.116	0.004	2556.564
EPS-1-2	0.416	0.018	0.008	0.117	0.003	2567.246
EPS-1-3	0.3748	0.014	0.006	0.120	0.003	1923.973
Total Time(s)	-	-	-	-		297222.018

Tabla 23 - Comportamiento experimento 10% subnodos, HC, centralidad Closeness, distribución normal y canal/color fijo 1.

Escenario	U_Norm	STD	Fairness Fujita		Time(s)
-	-	-	Avg	STD	Avg
EPS-02-1	0.572	0.044	0.123	0.014	0.139
EPS-0.25-1	0.408	0.040	0.121	0.007	0.889
EPS-0.25-2	0.398	0.041	0.116	0.004	0.905
EPS-0.25-3	0.447	0.039	0.116	0.008	0.906
EPS-0.5-1	0.340	0.026	0.105	0.004	3.817
EPS-0.5-2	0.366	0.053	0.112	0.011	2.403
EPS-0.5-3	0.368	0.040	0.120	0.007	2.891
EPS-0.75-1	0.337	0.028	0.108	0.005	5.660
EPS-0.75-2	0.356	0.052	0.106	0.006	5.628
EPS-0.75-3	0.344	0.035	0.113	0.007	5.435
EPS-1-1	0.312	0.023	0.109	0.006	9.838
EPS-1-2	0.336	0.034	0.107	0.005	9.742
EPS-1-3	0.319	0.027	0.106	0.007	10.065
Total Time(s)	-	-	-	-	1164.590

Tabla 24- Comportamiento experimento 10% subnodos, LCCS, centralidad Closeness, ordenación por nodos de mayor centralidad y canal/color aleatorio.

Escenario	U_Norm	STD	IC	Fairness Fujita		Time(s)
-	-	-	-	Avg	STD	Avg
EPS-02-1	0.603	0.049	0.023	0.132	0.015	70.844
EPS-0.25-1	0.494	0.015	0.007	0.130	0.006	249.000
EPS-0.25-2	0.493	0.013	0.006	0.123	0.004	253.792
EPS-0.25-3	0.519	0.017	0.008	0.107	0.004	277.484
EPS-0.5-1	0.446	0.020	0.009	0.109	0.003	790.292
EPS-0.5-2	0.452	0.023	0.011	0.122	0.006	758.717
EPS-0.5-3	0.476	0.019	0.009	0.129	0.004	721.152
EPS-0.75-1	0.438	0.015	0.007	0.119	0.003	1538.260
EPS-0.75-2	0.473	0.024	0.011	0.114	0.007	1431.177
EPS-0.75-3	0.437	0.024	0.011	0.126	0.003	1339.046
EPS-1-1	0.393	0.015	0.007	0.117	0.004	2408.555
EPS-1-2	0.416	0.014	0.007	0.118	0.004	2410.688
EPS-1-3	0.402	0.018	0.008	0.121	0.004	2038.422
Total Time(s)	-	-	-	-		285748.583

Tabla 25- Comportamiento experimento 20% subnodos, SA, centralidad Closeness, ordenación por nodos de menor centralidad y canal/color fijo 1.

Escenario	U_Norm	STD	IC	Fairness Fujita		Time(s)
-	-	-	-	Avg	STD	Avg
EPS-02-1	0.690	0.013	0.006	0.101	0.004	80.880
EPS-0.25-1	0.369	0.005	0.002	0.135	0.003	213.888
EPS-0.25-2	0.344	0.007	0.003	0.147	0.004	209.846
EPS-0.25-3	0.417	0.005	0.002	0.150	0.002	234.460
EPS-0.5-1	0.315	0.006	0.003	0.123	0.003	630.456
EPS-0.5-2	0.375	0.008	0.004	0.141	0.003	619.069
EPS-0.5-3	0.372	0.005	0.003	0.149	0.002	590.307
EPS-0.75-1	0.351	0.008	0.004	0.133	0.002	1221.129
EPS-0.75-2	0.417	0.007	0.003	0.140	0.003	1193.967
EPS-0.75-3	0.394	0.008	0.004	0.141	0.002	1144.118
EPS-1-1	0.364	0.011	0.005	0.132	0.001	2008.520
EPS-1-2	0.367	0.007	0.003	0.133	0.003	2048.215
EPS-1-3	0.364	0.010	0.005	0.134	0.002	2047.820
Total Time(s)	-	-	-	-		244853.533

Tabla 26- Comportamiento experimento 30% subnodos, HC, centralidad Closeness, ordenación por nodos de mayor centralidad y canal/color fijo 11.

Escenario/ Color	1	6	11	Rnd
EPS-02-1	0.308489	0.337147	0.303762	0.338621
EPS-0.25-1	0.204904	0.191993	0.193100	0.225762
EPS-0.25-2	0.179385	0.156775	0.178088	0.193222
EPS-0.25-3	0.190691	0.207976	0.199863	0.210768
EPS-0.5-1	0.125718	0.137875	0.138120	0.171581
EPS-0.5-2	0.185223	0.178648	0.160533	0.162585
EPS-0.5-3	0.163962	0.172619	0.186373	0.176286
EPS-0.75-1	0.151397	0.145971	0.152398	0.165197
EPS-0.75-2	0.159026	0.156810	0.173593	0.183688
EPS-0.75-3	0.167105	0.144859	0.169654	0.183093
EPS-1-1	0.154773	0.142451	0.153926	0.137109
EPS-1-2	0.154196	0.154630	0.159843	0.165141
EPS-1-3	0.149382	0.144296	0.138181	0.141482

Tabla 27 - Utilidad experimento 10% subnodos, Rnd, centralidad EigenVector, ordenación por nodos de mayor centralidad.

Escenario/ Color	1	6	11	Rnd
EPS-02-1	0.508259	0.502916	0.502031	0.559711
EPS-0.25-1	0.380228	0.367449	0.368936	0.414561
EPS-0.25-2	0.345376	0.354451	0.367456	0.378517
EPS-0.25-3	0.425432	0.391412	0.401432	0.437672
EPS-0.5-1	0.292770	0.296807	0.280827	0.323338
EPS-0.5-2	0.359184	0.357233	0.351300	0.365317
EPS-0.5-3	0.355608	0.342884	0.344947	0.352966
EPS-0.75-1	0.298865	0.323555	0.302703	0.329309
EPS-0.75-2	0.340664	0.334053	0.340337	0.362755
EPS-0.75-3	0.324189	0.321293	0.320483	0.342247
EPS-1-1	0.309533	0.315251	0.309341	0.321153
EPS-1-2	0.321761	0.324929	0.326643	0.318163
EPS-1-3	0.315690	0.296170	0.304548	0.304538

Tabla 28- Utilidad experimento 10% subnodos, LCCS, centralidad EigenVector, ordenación por nodos de mayor centralidad.

Escenario/ Color	1	6	11	Rnd
EPS-02-1	0.640042	0.639956	0.604660	0.614180
EPS-0.25-1	0.421001	0.419715	0.418665	0.419140
EPS-0.25-2	0.404025	0.399860	0.404389	0.400551
EPS-0.25-3	0.412972	0.424213	0.424451	0.401625
EPS-0.5-1	0.317885	0.331465	0.322569	0.322412
EPS-0.5-2	0.390082	0.383304	0.399928	0.329899
EPS-0.5-3	0.396455	0.373597	0.375532	0.382300
EPS-0.75-1	0.331712	0.326915	0.314982	0.335070
EPS-0.75-2	0.392665	0.391155	0.392640	0.369748
EPS-0.75-3	0.356893	0.357547	0.350571	0.334241
EPS-1-1	0.359216	0.365820	0.352772	0.338402
EPS-1-2	0.368713	0.363439	0.356155	0.337013
EPS-1-3	0.345734	0.296170	0.304548	0.304538

Tabla 29- Utilidad experimento 10% subnodos, HC, centralidad EigenVector, ordenación por nodos de mayor centralidad.

Escenario/ Color	1	6	11	Rnd
EPS-02-1	0.694118	0.697543	0.696714	0.697204
EPS-0.25-1	0.479956	0.479044	0.475872	0.516342
EPS-0.25-2	0.462701	0.468283	0.463536	0.491523
EPS-0.25-3	0.499594	0.490919	0.4974	0.536937
EPS-0.5-1	0.358547	0.367674	0.369513	0.417827
EPS-0.5-2	0.451324	0.454948	0.451422	0.457422
EPS-0.5-3	0.439120	0.438231	0.443691	0.460140
EPS-0.75-1	0.390173	0.383701	0.385821	0.413848
EPS-0.75-2	0.439080	0.439742	0.439703	0.456378
EPS-0.75-3	0.396646	0.400258	0.399768	0.414353
EPS-1-1	0.400232	0.405257	0.394462	0.389016
EPS-1-2	0.408487	0.412596	0.412816	0.414029
EPS-1-3	0.397507	0.400467	0.397504	0.397453

Tabla 30 - Utilidad experimento 10% subnodos, SA, centralidad EigenVector, ordenación por nodos de mayor centralidad.

Escenario/ Negociador	Random	LCCS	HC	SA
EPS-02-1	0.0002	0.143	68.401	62.231
EPS-0.25-1	0.0004	0.944	250.776	230.494
EPS-0.25-2	0.00045	0.947	252.228	226.206
EPS-0.25-3	0.0004	0.943	334.156	252.650
EPS-0.5-1	0.0006	2.880	787.295	694.609
EPS-0.5-2	0.0006	2.510	774.477	674.029
EPS-0.5-3	0.0006	2.869	975.729	643.711
EPS-0.75-1	0.0012	5.637	1867.117	1353.446
EPS-0.75-2	0.001	5.632	1971.705	1292.240
EPS-0.75-3	0.00085	5.362	1420.169	1231.116
EPS-1-1	0.00125	9.750	3003.794	2169.603
EPS-1-2	0.00125	9.673	2449.483	2237.180
EPS-1-3	0.0012	10.021	1811.422	1692.211

Tabla 31 - Relación tiempo de ejecución - Técnica de negociación.