Curso 2023-2024

Iñaki Diez Lambies

MUIARFID

UPV

Práctica 1: Gossiping

Sistemas multiagente

Tabla de contenido

[1. Introducción 2](#_Toc156258594)

[2. Metodología 3](#_Toc156258595)

[2.1. BaseAgent 3](#_Toc156258596)

[2.1.1. BaseAgentMessage 4](#_Toc156258597)

[2.1.2. Función de comparación 4](#_Toc156258598)

[2.2. Implementación de los algoritmos 4](#_Toc156258599)

[2.2.1. Push 5](#_Toc156258600)

[2.2.2. Pull 6](#_Toc156258601)

[2.2.3. Push-pull 8](#_Toc156258602)

[2.3. Main.py 9](#_Toc156258603)

[2.3.1. Recopilación de resultados 10](#_Toc156258604)

[3. Experimentación 11](#_Toc156258605)

[4. Análisis de resultados 12](#_Toc156258606)

[4.1. Tiempos de convergencia 12](#_Toc156258607)

[4.2. Número de mensajes enviados 14](#_Toc156258608)

[4.3. Comparativa de configuraciones 16](#_Toc156258609)

[5. Conclusiones 17](#_Toc156258610)

# Introducción

En este documento, nos enfocamos en la implementación y análisis de algoritmos *Gossiping* en agentes desarrollados mediante SPADE, una plataforma para la creación de sistemas multiagente. La práctica aborda un desafío específico: la difusión eficiente de información entre agentes, utilizando las variantes *Push*, *Pull* y *Push-Pull* de los algoritmos *Gossiping*.

Cada agente, iniciando con un valor entero generado aleatoriamente, comparte esta información con un conjunto selecto de otros agentes. Nuestro objetivo es evaluar cómo estos métodos impactan en la rapidez de la difusión y en la cantidad de comunicación necesaria para alcanzar un estado de convergencia, ya sea hacia un valor máximo, mínimo, o promedio.

Este documento presenta una descripción detallada de la metodología aplicada, incluyendo la implementación técnica y el diseño experimental. Seguido a esto, se exponen los resultados obtenidos en diversas configuraciones, analizando los tiempos de convergencia y el número de mensajes intercambiados entre los agentes. Finalmente, se ofrecen conclusiones basadas en los hallazgos, proporcionando una visión comprensiva sobre la eficacia y eficiencia de los algoritmos *Gossiping* en el contexto de sistemas multiagente.

Con todo esto, el trabajo no solo busca aportar a la comprensión teórica de estos algoritmos, sino también explorar sus aplicaciones prácticas en entornos controlados.

# Metodología

La sección de metodología de este documento describe el diseño y la estructura del código implementado para evaluar los algoritmos *Gossiping* en un entorno de agentes SPADE. El código se compone de varios módulos clave, cada uno desempeñando un papel específico en la simulación y análisis de los algoritmos *Push*, *Pull* y *Push-Pull*.

En el núcleo de la implementación se encuentra *BaseAgent*, una clase fundamental que define las operaciones y características comunes a todos los agentes. Este módulo establece la base sobre la cual se construyen las variantes específicas de los agentes para cada algoritmo. Cada tipo de agente, representado por los archivos *PushAgent.py*, *PullAgent.py* y *PushPullAgent.py*, extiende y personaliza la funcionalidad del *BaseAgent* para implementar los mecanismos de comunicación y difusión de información correspondientes a su algoritmo particular.

El archivo *main.py* actúa como punto de entrada para la ejecución de las simulaciones, coordinando la creación de agentes, la inicialización de los valores y la gestión del flujo de ejecución. Este componente es crucial para configurar los experimentos y recopilar los datos necesarios para el análisis posterior.

A lo largo de esta sección, se expone cómo cada uno de estos componentes contribuye al funcionamiento general del sistema de agentes, detallando las interacciones entre ellos y el papel que desempeñan en el logro de los objetivos del experimento. Este análisis permite una comprensión clara de la lógica detrás de la implementación y sienta las bases para la interpretación de los resultados obtenidos en las pruebas realizadas. ​

## BaseAgent

Esta clase constituye la base común desde la cual se derivan las variaciones específicas para los algoritmos *Push*, *Pull* y *Push-Pull*. La clase encapsula las funcionalidades y características esenciales que son compartidas por todos los agentes en el sistema, proporcionando un marco de trabajo coherente y eficiente para la simulación. Está diseñado para manejar tareas fundamentales como la inicialización del agente, la gestión de la comunicación y la ejecución de las operaciones básicas necesarias para los algoritmos de *gossiping*.

Texto

Descripción generada automáticamente

Ilustración - Implimentación de BaseAgent

En cuanto a las funcionalidades que aporta encontramos que cada *BaseAgent* y que podemos ver en la Ilustración 1, mantiene un estado interno, que representa la información que el agente posee y debe difundir a otros agentes. Este estado es crucial para el funcionamiento de los algoritmos de *gossiping*, ya que es el valor que se comparte y se actualiza durante el proceso de comunicación. Por otro lado, la clase proporciona una estructura para la programación de los comportamientos, lo cual es esencial para la implementación de los ciclos de *gossiping*. Estas tareas pueden ser la emisión de mensajes (en el caso de *Push* y *Push-Pull*) o la solicitud de información (en el caso de *Pull*) las cuales se han de configurar a través del método abstracto *add\_all\_behavious* que toda clase derivada ha de implementar.

### BaseAgentMessage

Una parte integral de *BaseAgent* es *BaseAgentMessage*, una subsección que maneja el formato y la estructura de los mensajes intercambiados entre los agentes. Este protocolo de mensajería garantiza que la información transmitida sea consistente y comprensible para todos los agentes receptores, facilitando así una comunicación efectiva y la correcta actualización de los estados. La propia clase *BaseAgent* que podemos ver en la Ilustración 2 proporciona una serie de métodos para comunicarse con la clase *BaseAgentMessage* consiguiendo una comunicación consistente entre todos los agentes.

Texto

Descripción generada automáticamente

Ilustración - Implementación de BaseAgentMessage

### Función de comparación

Otro componente esencial del *BaseAgent* es la función de comparación, responsable de evaluar y actualizar el estado del agente basándose en la información recibida. Esta sirve de interfaz ya que todos los agentes requieren de su existencia y nos permite, por tanto, evitar código repetido.

## Implementación de los algoritmos

Esta sección se enfoca en desglosar las particularidades de cada algoritmo, destacando las similitudes y diferencias en su enfoque y funcionamiento. Aunque cada variante de agente tiene características únicas, hay elementos comunes que forman la base de su implementación y operación.

En primer lugar, todos los agentes, ya sean *Push*, *Pull* o *Push-Pull*, se derivan de la clase *BaseAgent*. Esto garantiza que mantienen una estructura común y se adhieren a un protocolo de comunicación consistente, facilitando la integración y el funcionamiento conjunto en el sistema. De esta forma, *cada* variante de agente utiliza el estado interno y el protocolo de mensajería definidos en *BaseAgent*. La forma en que cada tipo de agente actualiza su estado y se comunica con otros varía según el algoritmo, pero el mecanismo subyacente de manejo de estados y envío/recepción de mensajes es un pilar común.

Aunque los métodos de convergencia difieren entre los algoritmos, todos los agentes buscan alcanzar un estado de convergencia, ya sea hacia un valor máximo, mínimo o promedio (dependiendo de la función de comparación escogida). También podemos variar parámetros como el número de vecinos (k) y evaluar así el impacto de estos cambios en el rendimiento del algoritmo.

### Push

El agente *Push*, definido en el archivo *PushAgent.py*, representa una implementación específica del algoritmo *Gossiping*. Este agente, al extender la funcionalidad del *BaseAgent*, incorpora las características fundamentales de comunicación y gestión de estado, adaptándolas para ejecutar el algoritmo *Push* de difusión de información.

La esencia del agente *Push* radica en su método proactivo para compartir información. A diferencia de otros enfoques, donde un agente puede esperar a recibir datos o solicitarlos activamente, el agente *Push* toma la iniciativa de enviar su información a otros agentes seleccionados.

En términos de implementación, el agente *Push* utiliza un mecanismo de temporizador o un ciclo definido para determinar cuándo enviar información. En cada ciclo, el agente selecciona uno o varios agentes destino de su red y les envía un mensaje con su estado actual. Esta selección puede ser aleatoria o basarse en ciertos criterios, dependiendo de la configuración específica del experimento que en nuestro caso es aleatoria. Este enfoque asegura que la información se propague continuamente a través de la red, incluso en ausencia de solicitudes externas-

Una característica clave del agente *Push* es su capacidad para adaptarse a diferentes configuraciones y escenarios. Por ejemplo, el número de agentes a los que se envía información (k-vecinos) puede ajustarse para observar cómo afecta la velocidad y eficacia de la difusión. Además, el contenido del mensaje y la frecuencia de envío pueden configurarse para simular distintas condiciones y evaluar el rendimiento del agente bajo diversas circunstancias.

#### Comportamientos

El agente *Push* en su implementación se caracteriza por dos comportamientos principales: *PushBehaviour* y *RecvBehaviour*, ambos cruciales para el funcionamiento del algoritmo *Gossiping*.

Texto

Descripción generada automáticamente

Ilustración - Implementación de PushBehaviour y RecvBehaviour

El *PushBehaviour* es un comportamiento periódico que define cómo y cuándo el agente envía información a otros agentes. Se puede apreciar que este comportamiento es esencial para el enfoque proactivo del agente *Push*. Durante su ejecución, el agente selecciona aleatoriamente un conjunto de contactos (determinado por el valor de *k*) de su lista de contactos y les envía un mensaje que contiene su estado actual. Esta selección aleatoria y la periodicidad del comportamiento aseguran una difusión constante y extensa de la información a través de la red de agentes.

El método run de *PushBehaviour* primero verifica si el sistema está listo para operar. Si es así, procede a seleccionar los contactos y enviar los mensajes. La eficiencia y efectividad de este comportamiento radican en su capacidad para mantener activa la red, promoviendo una rápida propagación de la información sin depender de las solicitudes o acciones de otros agentes.

Por otro lado, *RecvBehaviour* es un comportamiento cíclico que maneja los mensajes entrantes. Este comportamiento es vital para la actualización del estado del agente basándose en la información recibida de otros agentes. En su método run, el agente espera recibir un mensaje dentro de un tiempo específico (en este caso, un *timeout* de 2 segundos). Si se recibe un mensaje dentro de este período, el agente procesa la información como podemos ver en la Ilustración 3.

El procesamiento de los mensajes se realiza mediante el método *add\_value*, que compara el valor actual del agente con el valor recibido en el mensaje como vemos en la Ilustración 4. Si el valor recibido resulta en un cambio del estado actual (según la función de comparación), el agente actualiza su valor y registra este cambio en su historial de valores. Si no hay cambio, simplemente registra el intento de actualización. Este registro incluye no solo el valor sino también la marca de tiempo y el remitente del mensaje, proporcionando un seguimiento detallado de la interacción y la evolución del estado del agente a lo largo del tiempo.

Texto

Descripción generada automáticamente con confianza media

Ilustración - Implementación de PushAgent

La combinación de *PushBehaviour* y *RecvBehaviour* en el agente *Push* permite una dinámica de comunicación en dos direcciones: el agente no solo envía activamente su información a otros, sino que también está preparado para recibir y procesar la información de otros agentes. Esta dualidad asegura que el agente esté constantemente involucrado en el proceso de difusión de información, ya sea como emisor o receptor, lo que es fundamental para el éxito y la eficiencia del algoritmo *Gossiping* en un entorno de sistemas multiagente.

### Pull

El agente *Pull*, encontrado en el archivo *PullAgent.py*, es una implementación detallada del algoritmo que, al igual que el agente *Push*, este agente se basa en la clase *BaseAgent* para sus capacidades fundamentales de gestión de estado y comunicación. Sin embargo, el enfoque de *Pull* es significativamente diferente, centrado en la solicitud activa de información en lugar de su difusión proactiva.

En el agente *Pull*, el énfasis está en la iniciativa del agente para obtener información de sus pares, en contraposición al enfoque del agente *Push* de enviar información sin una solicitud previa. Esto se manifiesta principalmente en sus comportamientos especializados: *PullBehaviour*, *RecvPullBehaviour* y *RecvReplyBehaviour*.

#### Comportamientos

El primer comportamiento que encontramos es *PullBehaviour*, un comportamiento periódico similar al *PushBehaviour* del agente *Push*, pero con un propósito diferente. En lugar de enviar su propio estado, el agente *Pull* envía solicitudes a un conjunto de agentes seleccionados aleatoriamente para obtener su información. Esta selección se realiza en base al valor de *k*, al igual que en el agente *Push* como se puede ver en la Ilustración 5. Sin embargo, en lugar de simplemente difundir su estado, el agente *Pull* solicita activamente información de otros agentes, lo que implica una naturaleza más inquisitiva en su interacción con la red.

Texto

Descripción generada automáticamente

Ilustración - Implementación de PullBehaviour

Por otro lado, el *RecvPullBehaviour* es un comportamiento cíclico que se encarga de procesar las solicitudes entrantes de otros agentes. Cuando un agente *Pull* recibe un mensaje con la solicitud de información, el comportamiento *add\_value* es invocado. Este método no solo actualiza el estado del agente *Pull* si el valor recibido es diferente, sino que también envía una respuesta con su propio estado actual si el valor recibido no coincide con el suyo. Esta dinámica de solicitud y respuesta fomenta una interacción bidireccional, donde el agente no solo recibe información, sino que también contribuye con su propio estado a la red. Esto se puede apreciar en la Ilustración 6.

Texto

Descripción generada automáticamente

Ilustración - Implementación de RecvPullBehaviour

Por último, el *RecvReplyBehaviour* gestiona las respuestas a las solicitudes de información enviadas por el agente *Pull* como vemos en la Ilustración 7. Al recibir una respuesta, este comportamiento evalúa si el valor recibido debería provocar una actualización del estado del agente. Si es así, actualiza el estado y registra este cambio en el historial de valores. Esto completa el ciclo de comunicación del agente *Pull*, donde solicita información, recibe respuestas y actualiza su estado en consecuencia.

Imagen que contiene Texto

Descripción generada automáticamente

Ilustración - Implementación de RecvReplyBehaviour

Comparado con el agente *Push*, el agente *Pull* adopta un enfoque más reactivo y dependiente de la información proporcionada por otros agentes. Mientras que el agente *Push* se enfoca en la difusión activa de su estado, el agente *Pull* busca activamente información de otros, lo que resulta en un estilo de comunicación más interrogativo y dependiente de la participación de otros agentes en la red.

## Push-pull

El agente *PushPull*, cuya implementación se encuentra en *PushPullAgent.py*, es un híbrido de los enfoques *Push* y *Pull*. Combina las estrategias de ambos algoritmos para crear un modelo de difusión de información más versátil y potente en el contexto de los sistemas multiagente SPADE.

En el agente *PushPull*, se integran los comportamientos de emisión proactiva de información (característico del agente *Push*) y de solicitud y procesamiento de información de otros agentes (propio del agente *Pull*). Este enfoque dual permite al agente *PushPull* no solo difundir su propio estado, sino también solicitar y recibir información de otros.

#### Comportamientos

El *PushPullBehaviour* de la Ilustración 8 se encarga de enviar activamente el estado del agente a una selección aleatoria de contactos, similar al comportamiento del agente *Push*. Esta acción asegura que el agente no solo esté pasivamente esperando información, sino que también esté contribuyendo activamente a la distribución de datos en la red.

Texto

Descripción generada automáticamente

Ilustración – Implementación de PushPullBehaviour

Por otro lado, el *RecvPushPullBehaviour* gestiona los mensajes entrantes. Este comportamiento es similar al *RecvPullBehaviour* del agente *Pull*, donde el agente evalúa la información recibida y decide si actualizar su estado como vemos en la Ilustración 9. Además, si el valor recibido es diferente del propio, el agente *PushPull* responde con su propio estado actual, promoviendo así una interacción bidireccional.

Pantalla de computadora

Descripción generada automáticamente

Ilustración – Implementación de RecvPushPullBehaviour

Finalmente, *RecvReplyBehaviour* de la Ilustración 10 maneja las respuestas a las solicitudes de información enviadas, completando el ciclo de comunicación bidireccional. Este comportamiento asegura que el agente no solo reciba información nueva, sino que también proporcione retroalimentación, contribuyendo así a una distribución más equilibrada de la información en toda la red.

Imagen que contiene Texto

Descripción generada automáticamente

Ilustración - Implementación de RecvReplyBehaviour

Este agente representa un equilibrio entre la iniciativa proactiva del agente *Push* y la naturaleza inquisitiva del agente *Pull*. Mientras que el agente *Push* se enfoca en la difusión constante de su propio estado y el agente *Pull* en la búsqueda activa de información, el agente *PushPull* combina ambos enfoques, lo que le permite adaptarse a diferentes situaciones y responder de manera más efectiva a las dinámicas de la red. En resumen, el agente *PushPull* destaca por su capacidad para armonizar las ventajas de los enfoques *Push* y *Pull*.

## Main.py

El script *main.py* realiza la ejecución y evaluación del sistema de agentes *SPADE* que implementan los algoritmos *Gossiping*. Este script actúa como el punto de entrada para la simulación, coordinando la creación, inicialización y monitoreo de los agentes, ya sean del tipo *Push*, *Pull* o *PushPull*.

Inicialmente, el script define una función *main*, que es la responsable de orquestar la simulación. Dentro de esta función, se crean los agentes según los parámetros proporcionados (tipo de agente, número de agentes, valor de *k* y función de comparación) como podemos ver en la Ilustración 11. Esta flexibilidad permite experimentar con diferentes configuraciones y observar cómo cada una afecta el rendimiento del sistema. Una vez creados, los agentes se agregan a una lista y se les asignan sus contactos. Posteriormente, se inicia la ejecución de cada agente y se espera hasta que todos estén en funcionamiento.

Texto

Descripción generada automáticamente

Ilustración - Implementación de main.py

### Recopilación de resultados

Una característica destacada del script es su capacidad para recopilar y registrar datos sobre el proceso de difusión de la información. A medida que los agentes interactúan, sus historiales de valores se van acumulando, capturando detalles del valor en cada instante, la marca de tiempo y el agente emisor del mensaje.

Imagen que contiene monitor, computadora, computer

Descripción generada automáticamente

Ilustración - Recolección de datos de main.py

Al finalizar la simulación, estos datos se consolidan en un *DataFrame* de *Pandas*, proporcionando una representación estructurada y detallada de la evolución de la información en la red. Este *DataFrame* luego se exporta a un archivo CSV, creando un registro persistente de la simulación que puede ser analizado posteriormente como vemos en la Ilustración 12. La recopilación y registro de datos en este formato ofrece varias ventajas. Esto permite un análisis detallado y retrospectivo de la simulación, facilitando la identificación de patrones, tendencias y anomalías en la difusión de la información.

En resumen, *main.py* no solo inicia y gestiona la simulación de los agentes SPADE, sino que también proporciona herramientas esenciales para la recopilación, registro y análisis de datos, elementos clave para la evaluación y comprensión profunda del sistema de agentes y los algoritmos implementados.

# Experimentación

El script de experimentación es una herramienta fundamental para realizar pruebas sistemáticas y exhaustivas con diferentes configuraciones de los agentes *SPADE*. Este script automatiza la ejecución de múltiples simulaciones, variando parámetros clave como el tipo de agente (*Push*, *Pull*, *PushPull*), el número de agentes, el valor de *k* y la función de comparación (mínimo, máximo, media).

El script itera a través de una serie de combinaciones predefinidas de estos parámetros, ejecutando una simulación para cada configuración. Utiliza el comando *subprocess.run* para invocar el script *main.py* con los argumentos correspondientes, lo que permite iniciar cada simulación en su propio proceso. Esto facilita la realización de múltiples pruebas de forma automatizada y secuencial como vemos en la Ilustración 13.

Las configuraciones incluyen:

* Tipos de Agentes: *Push*, *Pull*, *PushPull*.
* Número de Agentes: 10, 20, 100, 200, 500.
* Valores de *k*: 1, 2, 5.
* Funciones de Comparación: Mínimo, Máximo, Media.

Gráfico

Descripción generada automáticamente con confianza media

Ilustración - Implementación de experiments.py

Cada ejecución de la simulación genera un archivo *CSV* con los datos recopilados, como se detalla en el script *main.py*. Estos archivos contienen información valiosa sobre la evolución de los estados de los agentes a lo largo del tiempo y bajo las diferentes configuraciones de prueba.

Es importante mencionar que, debido a limitaciones de recursos, solo se han podido completar las primeras seis pruebas con 200 agentes. Esto significa que, aunque se planificaron pruebas con hasta 500 agentes, las limitaciones prácticas han restringido el alcance de los experimentos. A pesar de esto, los datos recopilados en las pruebas completadas aún proporcionan información significativa sobre el comportamiento y la eficiencia de los diferentes tipos de agentes y configuraciones.

La recopilación de datos de estas simulaciones proporciona una base sólida para el análisis posterior. Al examinar los archivos *CSV* generados, se pueden obtener *insights* sobre aspectos como la velocidad de convergencia, la eficiencia en el uso de mensajes y la efectividad de diferentes estrategias de difusión de información. Este análisis es crucial para entender cómo las variables del sistema afectan el rendimiento y pueden guiar mejoras y ajustes en futuras implementaciones o experimentos.

# Análisis de resultados

En esta sección se abordarán aspectos cruciales derivados de las pruebas realizadas centrándonos en la interpretación y comprensión de los datos recogidos durante las simulaciones, proporcionando una visión detallada del comportamiento y eficiencia de los diferentes tipos de agentes y configuraciones en el sistema.

## Tiempos de convergencia

Aquí se puede encontrar un análisis fundamental para evaluar la eficiencia de los algoritmos *Gossiping* en el sistema de agentes. En este segmento, se centra la atención en medir y comparar la rapidez con la que diferentes configuraciones de agentes alcanzan un estado de convergencia. La convergencia se define como el punto en el que todos los agentes en la red han llegado a un acuerdo o han estabilizado su información, ya sea al valor más alto, más bajo, o la media, dependiendo de la función de comparación utilizada.

Se observarán y compararán estos tiempos bajo diferentes parámetros como el tipo de agente (*Push*, *Pull*, *PushPull*), el número de agentes en la red, y el valor de *k*, que determina el número de agentes con los que cada uno interactúa. El objetivo es identificar qué configuraciones son más eficientes para una rápida diseminación y actualización de la información.

Para el análisis de los tiempos de convergencia, se han proporcionado tres gráficos que representan cómo el número de agentes y el valor de *k* influyen en el tiempo necesario para alcanzar la convergencia bajo las funciones de comparación de máximo, media y mínimo.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ilustración - Tiempos de convergencia para max

En la Ilustración 14 se muestra la relación entre el tiempo de convergencia y el valor de *k* para las configuraciones con 10, 20 y 100 agentes. Se observa una tendencia clara en la que el tiempo de convergencia disminuye a medida que aumenta el valor de k para los grupos más pequeños de agentes (10 y 20). Sin embargo, para el grupo de 100 agentes, el tiempo de convergencia se mantiene relativamente constante, lo que sugiere que, en redes más grandes, incrementar el número de contactos a los que un agente envía información no tiene un impacto significativo en la rapidez de la convergencia.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ilustración - Tiempos de convergencia para mean

Por otro lado, en la Ilustración 15 nos enfocamos en la convergencia hacia la media de los valores. Similar al primer caso, hay una reducción notable en los tiempos de convergencia para los grupos de 10 y 20 agentes a medida que el valor de k aumenta. Para el grupo de 100 agentes, se percibe una disminución leve al principio, pero luego se estabiliza, de nuevo indicando lo mismo que en el anterior.

Gráfico

Descripción generada automáticamente

Ilustración - Tiempos de convergencia para min

Para la Ilustración 16 se muestra los tiempos de convergencia cuando la función de comparación se centra en el valor mínimo. Aquí, los resultados son consistentes con los anteriores: los grupos más pequeños muestran una mejora en los tiempos de convergencia al aumentar k, mientras que el grupo de 100 agentes muestra un tiempo de convergencia bastante estable independientemente del valor de *k*. Esto refuerza la idea de que, en redes más grandes, la influencia de k en la rapidez de convergencia disminuye.

De estos tres gráficos se pueden extraer conclusiones importantes. En primer lugar, que el tamaño de la red (número de agentes) tiene un impacto significativo en los tiempos de convergencia. Las redes más pequeñas son más sensibles a cambios en el valor de *k*, mostrando mejoras en los tiempos de convergencia al aumentar *k*. En contraste, las redes más grandes parecen ser menos afectadas por el incremento en k, posiblemente debido a que la información ya se difunde eficientemente a través de una red extensa de comunicaciones.

Seguidamente, sabemos ahora que aumentar el valor de *k* tiende a mejorar los tiempos de convergencia en grupos más pequeños de agentes, lo que indica que tener más contactos a los que un agente puede enviar o solicitar información acelera el proceso de convergencia. Sin embargo, existe un punto de saturación en redes más grandes donde aumentar k no produce mejoras significativas.

A pesar de que de si los agentes buscan el valor máximo, mínimo o la media, las tendencias en los tiempos de convergencia son consistentes a través de las diferentes funciones de comparación, lo que sugiere que el mecanismo subyacente de los algoritmos *Gossiping* es robusto frente a la meta específica de convergencia.

## Número de mensajes enviados

El análisis del número de mensajes enviados es un indicador de la eficiencia comunicativa dentro de un sistema de agentes que implementa algoritmos *Gossiping*. Este apartado examinará cómo la cantidad de comunicaciones requeridas para alcanzar la convergencia varía en función de los diferentes tipos de agentes y configuraciones. Este análisis nos permite comprender mejor cómo la estructura de la red y los parámetros operativos influyen en la carga de la red y, por extensión, en el uso de recursos.

Dentro de este marco, se considerará el número total de mensajes intercambiados hasta que todos los agentes hayan alcanzado un estado de conocimiento común o hayan llegado a un acuerdo sobre un valor específico. Evaluar el volumen de mensajes es fundamental, ya que un número excesivo puede indicar una sobrecarga de la red, lo que podría llevar a ineficiencias y retrasos en sistemas más amplios o en aplicaciones en tiempo real.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ilustración - Número de mensajes enviados para max

Para el caso de convergencia hacia el valor máximo, observamos en la Ilustración 17 que, para redes más pequeñas, el número de mensajes enviados disminuye a medida que aumenta *k*. Esto podría indicar que una mayor conectividad entre agentes reduce la necesidad de mensajes redundantes para alcanzar un valor máximo común. Sin embargo, para una red de 100 agentes, el número de mensajes baja significativamente más, lo que implica que un valor mayor de *k* consigue una conectividad mayor y, por tanto, una mayor eficiencia en el número de mensajes.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ilustración - Número de mensajes enviados para mean

En la búsqueda de la media, la tendencia es más marcada para la red de 100 agentes, donde el número de mensajes se mantiene elevado sin importar el valor de k. Esto que podemos ver en la Ilustración 18 podría reflejar la complejidad de llegar a un consenso cuando se trata de un promedio, ya que cada nuevo mensaje puede ajustar ligeramente el valor medio que se busca alcanzar, generando así una mayor cantidad de comunicación.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ilustración - Número de mensajes enviados para min

En cuanto a los mínimos, el patrón es similar al de los máximos para las redes más pequeñas, con una disminución en el número de mensajes a medida que k aumenta. Para la red de 100 agentes, el conteo de mensajes se incrementa ligeramente con mayores valores de k. Esto podría sugerir que, en redes más grandes, la comunicación se vuelve más compleja al intentar converger a un valor mínimo como vemos en la Ilustración 19.

De este análisis podemos suponer que la eficiencia comunicativa en sistemas multiagente no solo depende del número de agentes y de la densidad de la red (valor de k), sino también del tipo de convergencia que se busca.

También vemos que redes más pequeñas se benefician de un mayor valor de k en términos de reducción del número de mensajes necesarios para alcanzar la convergencia, pero existe un punto de saturación en redes más grandes. De hecho, la convergencia hacia la media es la que parece requerir una mayor carga comunicativa, lo cual es un dato importante que considerar en la optimización de este tipo de sistemas.

## Comparativa de configuraciones

Si observamos los historiales de mensajes de los diferentes experimentos nos podemos dar cuenta de lo eficaces que son estos algoritmos para conseguir su objetivo aunque, a primera vista, parezca que pudieran no ser la mejor opción para transmitir información entre diferentes sistemas. En la Ilustración 20 se refleja como los diferentes agentes consiguen llegar a un consenso fruto de las comunicaciones entre ellos y sus comportamientos enfocados a este objetivo.

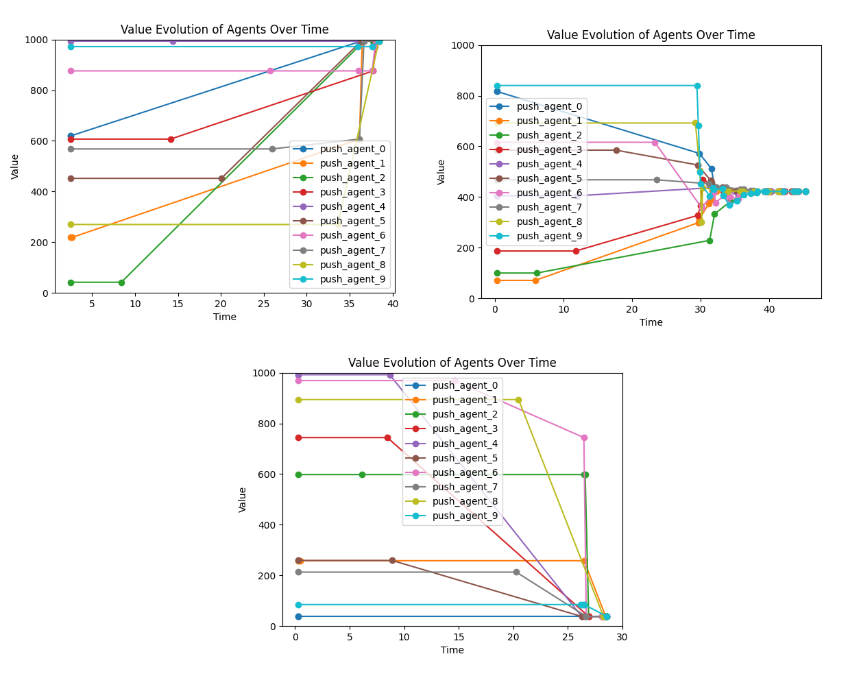


Ilustración - Comparativa de los historiales de diferentes configuraciones

# Conclusiones

A lo largo de este trabajo, se ha realizado una exploración exhaustiva de los algoritmos *Gossiping* aplicados a un entorno de agentes *SPADE*, revelando una serie de conocimientos y perspectivas que son esenciales en el campo de los sistemas multiagente. La implementación y el análisis detallado de los algoritmos *Push*, *Pull* y *PushPull* han demostrado su eficacia en la diseminación y convergencia de información, cada uno con sus particularidades y ventajas según la configuración y el contexto de aplicación.

El estudio ha destacado la importancia de la configuración de parámetros como el número de agentes y el valor de *k* en la eficiencia del proceso de convergencia. Se ha observado que mientras las redes más pequeñas se benefician de un aumento en el valor de *k*, mostrando una disminución en los tiempos de convergencia, las redes más grandes no siempre experimentan mejoras significativas con un k más alto. Esto subraya un punto en el diseño de sistemas de agentes: la necesidad de un equilibrio entre la conectividad y la sobrecarga de la comunicación.

Los resultados también han abierto discusiones sobre la carga comunicativa, evidenciando cómo las estrategias de difusión influyen en el número total de mensajes enviados y, por tanto, en la carga de la red. Se ha visto que la convergencia hacia un promedio puede requerir una mayor cantidad de comunicación, lo cual es una consideración valiosa para el diseño de sistemas que buscan eficiencia tanto en la convergencia como en la utilización de recursos.

En conclusión, este trabajo no solo ha confirmado la viabilidad de los algoritmos *Gossiping* en sistemas multiagente, sino que también ha delineado una serie de consideraciones críticas sobre este. La investigación ofrece una comprensión más profunda de cómo las estrategias de comunicación y los parámetros del sistema influyen en la eficiencia global de la red, proporcionando una guía valiosa de los sistemas multiagente.