Departamento de Matemáticas

TRABAJO FIN DE GRADO

Modelos de formación de redes socioeconómicas

Autor: Pablo Lozano Rodríguez

Director: Ángel Sánchez Sánchez
Codirector: Alberto Antonioni

Leganés, Junio 2016

Título: Modelos de formación de redes socioeconómicas
Autor: Pablo Lozano Rodríguez
Director: Ángel Sánchez Sánchez
Codirector: Alberto Antonioni

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día ..... de ..... de ..... en ....., en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

A mi familia.

Resumen

El objetivo del presente trabajo es la modelización matemática de la formación de redes sociales y económicas y su posterior evolución. En una sociedad tan conectada como la actual, la reputación es una herramienta indispensable para guiar las decisiones económicas y sociales con individuos desconocidos; por ello, es importante conocer cómo crecen y se desarrollan las redes que forman las personas en función de la reputación de las mismas.

En este proyecto se ha realizado un modelo matemático que intenta simular el comportamiento observado en los experimentos llevados a cabo en la Universidad de Lausanne y en el École Polytechnique Fédérale of Lausanne, y obtener a partir de dicho modelo unas leyes que los expliquen.

Se ha conseguido un modelo que, con solo dos parámetros, es capaz de simular el comportamiento observado. Se obtiene además una interpretación cuantitativa de la *reputación*. Esta se construye, en el modelo, como resultado de las interacciones de los agentes, que juegan al Dilema del Prisionero en redes dinámicas.

Palabras clave: redes, teoría de juegos, dilema del prisionero, modelo basado en agentes, reputación.

Abstract

The aim of this document is the mathematical modeling of the formation of social and economic networks and their evolution. In a networked society as today's one, reputation is an indispensable tool for guiding the economic and social decisions with unknown individuals; therefore, it is important to know how networks that people develop grow and form as influenced by people's reputation.

In this project a mathematical model attempts to simulate the behavior observed in experiments conducted at the University of Lausanne and the École Polytechnique Fédérale of Lausanne, and from its design laws to explain them are derived.

The agent-based model is capable of simulating the observed behaviour, using only two parameters. We also obtain a quantitative interpretation of reputation. In this case, reputation is the output of cooperative interactions, which is simulated with a Prisoner's Dilemma game played in dynamical networks.

Keywords: networks, agent-based model, game theory, prisoner's dilemma, reputation.

Índice general

Índice de figuras

Introducción

En este capítulo se expondrá el marco teórico de los modelos matemáticos, la motivación del proyecto y una explicación de la estructura del documento.

Estado del arte

Un modelo matemático es, desde un punto de vista, una relación entre ciertos objetos matemáticos y, también, un fenómeno de naturaleza no matemática [3]. La modelización matemática ha sido empleada, convencionalmente, como un proceso dinámico que ayudaba a entender problemas en distintas áreas, como la Física, Química o Biología, utilizando conceptos y técnicas matemáticas para el análisis de situaciones reales [4]. Durante las tres últimas décadas, se han desarrollado nuevas ramas de ciencias con conocimientos interdisciplinarios basándose en el marco teórico de la modelización matemática en física teórica [5], ya que existen "problemas complejos"que implican más de una especialidad para ser resueltos [6].

En el campo de la modelización matemática, sobre todo en la modelización de eventos discretos, se ha avanzado mucho en los últimos años. Se han desarrollado varios tipos de herramientas de análisis, como por ejemplo las redes continuas de Petri [7] o técnicas que emplean la Teoría de Juegos. La Teoría de Juegos es un área que emplea modelos para estudiar interacciones en estructuras formalizadas con incentivos [8], es decir, no analiza el azar de los elementos aleatorios que intervienen, sino que se centra en la estrategia de los jugadores.

Creada en 1944 por John von Neumann y Oskar Morgenstern [8], en su libro "*Theory of Games and Economics Behaviour*", y después popularizada por el ganador del premio Nobel John Nash, al aplicarla al estudio de situaciones económicas y procesos de negociación, la teoría de juegos es un campo que está siendo estudiado actualmente por numerosos especialistas en todo el mundo. Dentro de los problemas que trata, se pueden distinguir dos grandes grupos: *cooperativos* y *no cooperativos* [9]. En los primeros, los jugadores pueden intercambiar información entre ellos, es decir, pueden comunicarse, y su estudio se basa en analizar las coaliciones que podrán surgir. En la segunda, los jugadores no puede llegar a un acuerdo previo.

Dentro de los juegos sin transferencia de información, existe una gran clasificación. La mayoría son bipersonales, aunque son posibles juegos de más personas. Se llaman de suma cero cuando el aumento en las ganancias de un jugador provoca una disminución equivalente en las ganancias del otro, y de suma no nula si las ganancias de los agentes

involucrados pueden aumentar o disminuir en función de sus decisiones [9]. Si los jugadores tienen acceso solo a dos estrategias, se denominan juegos biestratégicos. En el caso de los juegos con repetición, los que se juegan varias veces seguidas por los mismos jugadores, las estrategias pueden ser también simples o reactivas, si la decisión depende del comportamiento que haya manifestado el contrincante en jugadas anteriores.

Dentro de los problemas sin transferencia de información de la teoría de juegos, existe un juego fundamental: el **dilema del prisionero**. Este modela una situación de cooperación entre dos individuos, que puede resultar en la no cooperación de ambos, aun siendo esto perjudicial para los dos [10]. Tal y como fue descrito por primera vez, era un juego simétrico y bipersonal, pero con el paso del tiempo se han elaborado multitud de variaciones, incluso con repetición de juegos. Como el dilema del prisionero es un problema fácil de modelar, en los últimos años se han realizado numerosos experimentos, involucrando agentes humanos [1] e incluso no humanos [11], para crear modelos que permitan explicar el comportamiento observado tanto a nivel individual como en grupo.

En concreto, para la realización del modelo de este trabajo, se ha empleado como texto básico el artículo [2] de Alberto Antonioni, Ángel Sánchez y Marco Tomassini; que trata un dilema del prisionero con repetición de juegos y simétrico. Y los resultados obtenidos se compararán con dicho artículo y con otra publicación de José A. Cuesta, Carlos Gracia-Lázaro, Alfredo Ferrer, Yamir Moreno y Angel Sánchez [1].

Motivación personal

La motivación principal de este proyecto era realizar un primer acercamiento a la investigación en el campo de las matemáticas. Al haber cursado Ingeniería en Tecnologías Industriales, el plan de estudios toca campos muy variados, pero no profundiza especialmente en ninguno de ellos. Por ello, deja bastante libertad en la elección del camino a seguir al finalizar el grado; y en mi caso es hacia el campo de la matemática aplicada.

Otra motivación básica fue la programación. Bajo mi punto de vista, aunque en la carrera se ven asignaturas de programación, en ninguna se desarrolla un proyecto de un tamaño considerable como para adquirir la soltura necesaria en un lenguaje de programación y poder emplear dichos conocimientos en otras asignaturas.

La elección del lenguaje no fue al azar, ya que Python es un lenguaje sencillo de aprender y aplicar, con una sintaxis clara y además es de licencia libre. Además, en los últimos años ha experimentado un enorme crecimiento, y se han desarrollado paquetes y herramientas que facilitan en gran medida la implementación de casi cualquier idea en este lenguaje.

También creo que en el futuro me serán muy necesarios y útiles los conocimientos en programación, tanto en el ámbito de la investigación como en el entorno laboral, por lo que la necesidad de aprender en profundidad un lenguaje de programación estaba ya presente.

Estructura del documento

A continuación, y para facilitar la lectura del documento, se detalla el contenido de cada capítulo.

- En el capítulo 1 se realiza una introducción a los experimentos realizados en el artículo base.
- En el capítulo 2 se explica el modelo creado.
- En el capítulo 3 se presentan los resultados obtenidos y se comparan con los experimentos realizados.

Organización del trabajo

A continuación, en la figura 1, se muestra un diagrama de Gantt con la planificación semanal del proyecto, las tareas correspondientes y los hitos conseguidos. El diagrama se ha estructurado en meses divididos en bloques de dos semanas.

Para una mejor gestión del tiempo y un trabajo más continuado, se llevó a cabo un sistema de hitos, mediante el cual, cada una o dos semanas se presentaban los resultados al director y al director del trabajo, quien los valoraba, corregía y guiaba en los siguientes pasos.

En la fase de *comienzo*, la idea era buscar toda la información relativa al lenguaje de programación, la elaboración de modelos matemáticos, pensar una estructura general y formas alternativas, así como estructurar el trabajo con los objetivos y los plazos. En la fase de *análisis*, se realizó un primer modelo del programa, basándose en otros trabajos de la misma temática. Tras dicho modelo, se hicieron las primeras pruebas, se mejoró aquello que no estaba correctamente implementado y se reestructuró para conseguir una versión final. Tras conseguirla, se pasó a la fase de *producción*, en la que, como dicho nombre indica, se generaron todos los resultados del modelo que se requerían. Como hito en esta etapa, se redactó un informe presentando los resultados, el cual sería posteriormente un capítulo entero del trabajo. Para terminar, en la fase de *finalización*, se terminó de redactar el trabajo. La fecha de fin viene marcada por la línea verde en el diagrama.

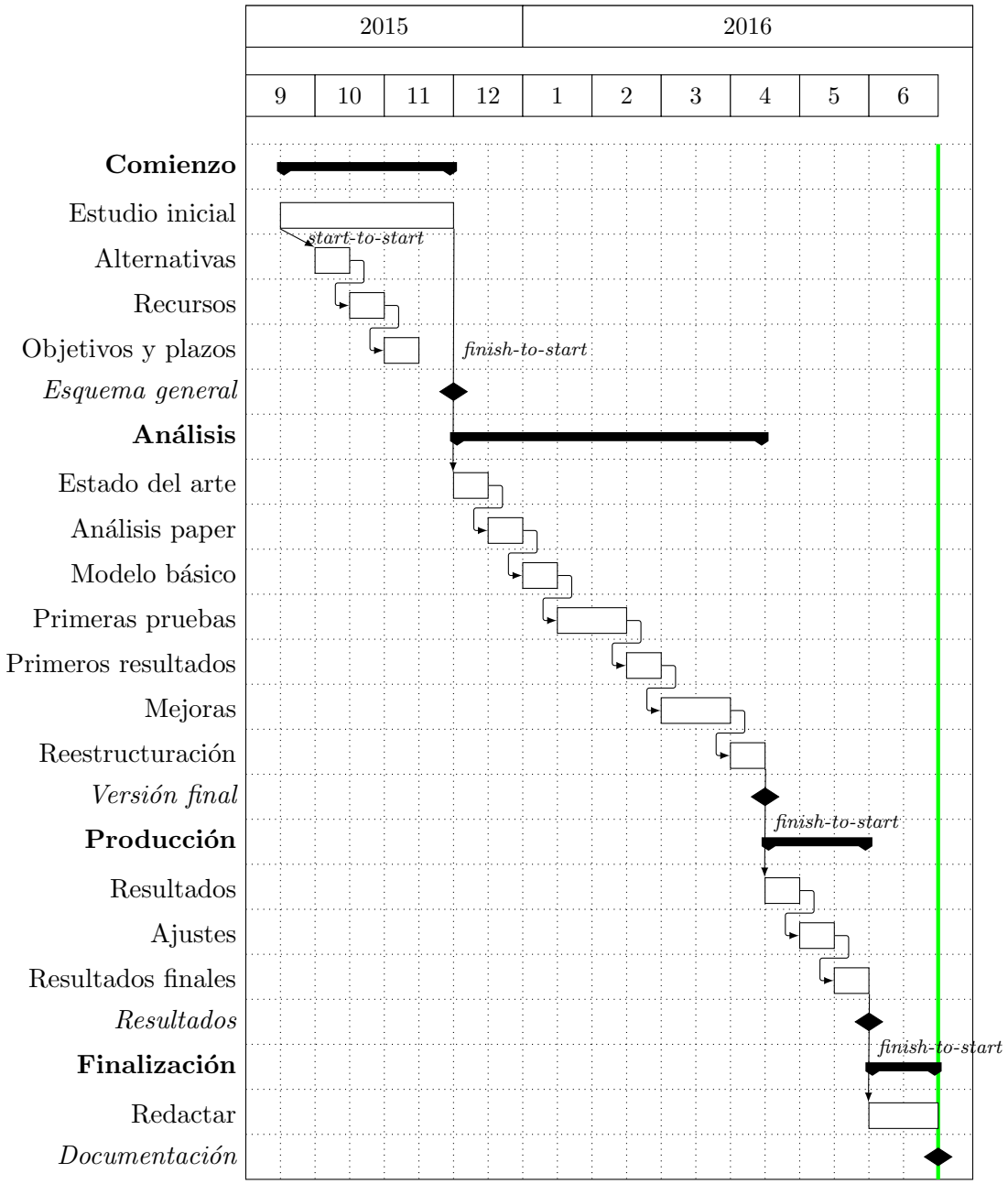


Figura 1: Diagrama de Gantt

Capítulo 1

En este capítulo se expondrá la motivación de la realización de este proyecto, explicando su contexto, de dónde se parte y a dónde se quiere llegar.

Motivación del proyecto

En una sociedad como la actual, conectada mediante redes, la herramienta que guía las decisiones que nos relacionan con otros miembros, generalmente, es la *reputación* de cada individuo. Normalmente, la información sobre la parte contraria está limitada, ya sea a una media de interacciones en el tiempo, a una sola acción o a ninguna. En ocasiones, debido a esta falta de información, comienza a surgir el fraude, que con el paso del tiempo se está convirtiendo en un problema importante en nuestra sociedad.

Para evitar el fraude y conseguir entender el comportamiento de los colectivos de gente, se están realizando numerosos estudios sobre la cooperación y la evolución de las redes formadas por personas. En concreto, este trabajo está basado en el estudio realizado por Alberto Antonioni, Ángel Sánchez y Marco Tomassini, titulado *"Cooperation survives and cheating pays in a networked society with unreliable reputation"* [2].

Introducción al estudio

En dicho documento se explica que, como la mayoría de las interacciones en la sociedad actual se llevan a cabo por medio de internet, en muchas ocasiones dichas interacciones involucran personas que se conocen solamente a través de un perfil en una web, sin que haya habido contacto en el mundo físico. Esto se produce mucho, por ejemplo, en las plataformas de ventas (eBay, Amazon, ...), pero también se da la misma situación en negocios físicos (resturantes, hoteles, ...) a los que nunca se ha acudido, o proveedores de internet. Hay páginas de opiniones (Yelp, TripAdvisor, ...) que usan herramientas de análisis muy sofisticadas para eliminar opiniones falsas, lo cual está provocando que surja todo un campo para detectar, evitar y eliminar los fraudes en la reputación.
Un problema cada vez más acuciante es el fraude. Este es más grave cuando las identidades personales no se pueden comprobar externamente; por lo que lo importante en las interacciones con nuevos miembros es la fiabilidad en que la información procedente de la otra parte es veraz.

En dicho estudio se llevaron a cabo experimentos basados en el juego del *Dilema del Prisionero* como un modelo de interacción social. Este problema se define de la siguiente

manera:

> *Dos presos han sido encarcelados por un delito. La policía los interroga por separado intentando que se incriminen mutuamente y les ofrece una reducción de condena por inculpar al otro, a no ser que a su vez éste confiese. El resultado de la colaboración o traición de cada prisionero viene dado por una tabla.*

	Coopera (1)	Traicionar (1)
Coopera (2)	Cada uno 2 años	(1) sale libre y 10 años para (2)
Traicionar (2)	(2) sale libre y 10 años para (1)	Cada uno 8 años

Cuadro 1: Dilema del Prisionero

Esta es una tabla ejemplo, ya que en los experimentos que se realizaron se proponía al agente la obtención de pagos, no penas de cárcel, aunque el funcionamiento sigue siendo el mismo: maximizar beneficios y minimizar las pérdidas.

Para los economistas, la "solución"del juego, lo que debe hacer una persona racional, viene dada por el *equilibrio de Nash*. Este concepto se refiere a una elección de acción para cada jugador que verifica que si uno de ellos cambia unilateralmente su elección no mejora sus ganancias. En el juego que nos ocupa, el único equilibrio de Nash es que ambos se traicionen, pero en ese caso a ambos les va peor que si se hubieran protegido. Por eso nos encontramos ante un dilema: la solución racional no proporciona el óptimo social.

Experimentos

En las sesiones de los experimentos, se organizaron siete grupos de veinte sujetos cada uno los cuales jugaban al dilema del prisionero, eligiendo si colaborar o no. Los sujetos estaban conectados unos con otros formando una red, y su elección de acción se aplicaba a todos los sujetos con los que se conectaba cada uno, es decir, a todos sus vecinos. Es importante insistir en que esto no es una colección de juegos independientes, sino que la acción se elige una y sólo vez para todos los compañeros de juego. Si un sujeto coopera, recibe R por cada vecino que hace lo mismo, y S por cada vecino que traiciona. Si el sujeto traiciona, recibe T por cada vecino que coopera, y P por cada vecino que traiciona. Para tener un dilema del prisionero, los pagos cumplen $T > R > P \geq S$. El juego se repite durante varias rondas. Tras cada ronda, los jugadores reciben información sobre lo que han hecho todos los demás, sean vecinos o no, y pueden reajustar sus conexiones antes de pasar a la siguiente ronda.

	Coopera (1)	Traicionar (1)
Coopera (2)	R (ambos)	T para (1), S para (2)
Traicionar (2)	S para (1), T para (2)	P (ambos)

Cuadro 2: Pagos en los experimentos

La red inicial era una red regular de grado 4, en la que se jugaban 30 rondas, aunque dicho número exacto no era conocido por los participantes.
Un factor importante a tener en cuenta es que al realizar estos experimentos con las redes fijas, es decir, que no se pueden eliminar o crear enlaces con los vecinos tiendo siempre a los mismos, la *cooperación en la red decae*. Típicamente, un 50 % de los participantes elige cooperar la primera vez, pero a medida que se van jugando más rondas van dejando de hacerlo.

> *Precisamente, el hecho de que la cooperación decaiga en redes fijas es lo que viene motivando a los investigadores a trabajar sobre redes reconfigurables. En este caso, experimentos recientes [1, 12, 13, 14, 15, 16] respaldan a los modelos teóricos [17, 18, 19], observando que el grado de cooperación aumenta significativamente cuando se permite a los individuos controlar con quién interactúan.*

Como se ha mencionado, en estos experimentos, se le proporciona a cada agente información sobre sus vecinos, y un elemento clave de dicha información es el *índice de cooperación*, denominado α. Este índice indica el número de veces que el jugador ha colaborado en las cinco últimas rondas, pero no en qué orden, por lo que $\alpha \in [0, 5]$.

En los juegos se tuvieron en cuenta dos situaciones: una denominada de *reputación real*, en la cual el índice de cooperación que observan los agentes no puede ser manipulado; y otra situación, de *reputación falseada*, en la que sí se puede modificar el índice α que los vecinos observan mediante el pago de un coste. Al comenzar, la reputación de todos los agentes se fijó en 3, basada en una secuencia de acciones arbitrariamente fijada en $CDCDC$; y además los vecinos no pueden ser diferenciados de una ronda a otra salvo por su reputación.

De esta forma, la reputación que un agente puede asignar a los vecinos se basa solamente en la media de la cooperación (el valor de α) sin ninguna referencia cronológica. Si extrapolamos esta situación al mundo real, esto es lo que sucede en muchas plataformas de comercio electrónico, donde solamente se muestra la media de interacciones satisfactorias con compradores o vendedores externos. Así, el experimento reproduce situaciones del mundo real, donde un sujeto interacciona con otro por primera vez, pero tiene información más o menos fiable sobre él procedente de terceras personas.

Entrando en el detalle, en los experimentos con *reputación real*, cada ronda, para cada jugador, consta de cuatro etapas:

1. Elegir una acción
2. Modificar su vecindario
3. Aceptar los nuevos enlaces
4. Recibir información sobre los pagos

En la primera etapa, el jugador recibe información sobre el índice de cooperación de todos los vecinos actuales, y debe elegir si coopera o no. En la segunda, los participantes pueden eliminar un enlace con un vecino (la decisión es unilateral, así que solo se tiene en cuenta la decisión del vecino que la propone), y también pueden proponer un nuevo enlace, de

forma aleatoria, a otro agente de la red. En ambos casos, solamente se conoce el índice α del agente con el que se quiere enlazar o romper el enlace. En la tercera etapa, cada jugador ve las proposiciones de enlaces de otros nodos, con su α correspondiente, y decide si acepta o rechaza. Después de todas estas acciones, se forma una nueva red, y a los jugadores se les paga en función del dilema del prisionero jugado (acción elegida en el paso 1).

A los agentes se les informa de sus pagos, pero en ningún caso se les informa de las acciones individuales de sus vecinos, ni de los pagos a sus vecinos. Tampoco conocen la topología de la red.

Los experimentos con *reputación falseada* son idénticos a los de *reputación real*, solo que con una diferencia fundamental: los agentes nunca saben si la cooperación que observan es la real, o está falseada. Por ello, existe una etapa adicional entre la 1 y la 2, en la cual los participantes eligen si pagar o no un coste para modificar su α. El coste elegido fue de 4 unidades por cada punto de reputación que incrementan, y dicho aumento solamente permanece la ronda actual; si quieren volver a ver su α aumentada en la siguiente ronda, deben pagar otra vez. Este tipo de sistema, en el que la reputación media se puede falsear, no ha sido implementado en el modelo, pero se propone como un trabajo futuro.

Resultados

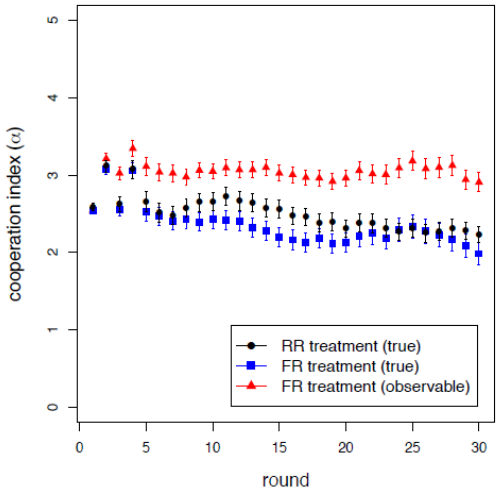
A pesar de que la acción que mayor beneficio reporta es la *traición*, en numerosas ocasiones se observa que la acción predominante era la cooperación.

Para explicar este fenómeno, se han propuesto numerosos mecanismos, la mayoría basados en algún tipo de clasificación entre cooperadores: los individuos de igual comportamiento colaboran entre sí para evitar a aquellos que no colaboran o falsean su identidad. El hecho de que puedan elegir con quién se conectan permite efectivamente restringir las interacciones a aquellos que cooperan.

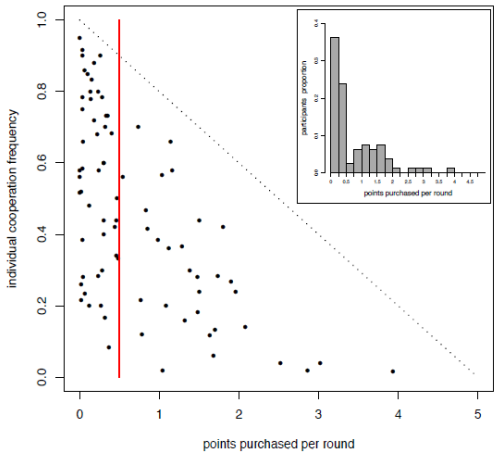
Además, en la investigación se encontró que el nivel de cooperación del conjunto permanecía constante, es decir, el comportamiento global prácticamente no cambiaba. Pero a nivel individual se observaban dos comportamientos: una parte de la población no engañaba, mientras que la otra falseaba su reputación en todas, o casi todas, las rondas. Los que falseaban la reputación terminaban con mejores resultados que aquellos que eran honestos, los cuales eran más cooperativos.

> En la práctica, se ve que los sujetos honestos acaban pagando por el fraude de aquellos que falsean, lo que lleva a una desigualdad en términos de la riqueza, con un aumento del índice de Gini en un 30 %.
> Esto nos lleva a destacar la importancia de asegurar la veracidad de la reputación para una sociedad más equitativa y justa.

Las gráficas del artículo [2], que aportan evidencia apoyando las conclusiones anteriores, se adjuntan en la figura 2.



(a) Cooperación - Ronda



(b) Cooperación - Puntos comprados

Figura 2: Distintos tipos de configuraciones iniciales

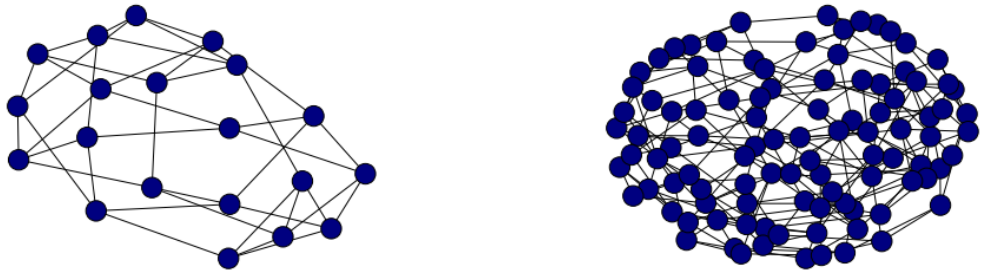
De ellas se puede deducir que, frente a la posibilidad de falsear la reputación de un agente, aunque esto ocurra, no afecta en gran medida al comportamiento medio de cooperación. O lo que es lo mismo, el *verdadero* α de los dos tipos de experimentos (con y sin reputación falseada) son iguales. En contraposición a otros experimentos, en estos, la cooperación no aumenta, sino que se mantiene constante, e incluso puede parecer que disminuye.

Introducción al modelo

La idea a modelizar, que parte del artículo expuesto anteriormente, es cómo surgen las redes y cómo evolucionan a partir de una configuración inicial para los agentes. Como condición inicial para el modelo se parte de que los agentes están conectados entre sí en una red regular del número de enlaces que se quiera y además suponiendo un tratamiento en el que la reputación no se puede falsear.

La configuración que arbitrariamente se eligió para comenzar las simulaciones fue una red regular de 4 enlaces y 20 nodos. También se hicieron simulaciones con redes de 100 nodos, para comprobar si el tamaño de la red afecta a su evolución. Además, se les asignó a los agentes un historial de acciones aleatorio, con una probabilidad del 50 % de que cada acción fuese colaborar, por lo que todas los historiales son distintos, pero la probabilidad inicial de colaborar es del 50 %.

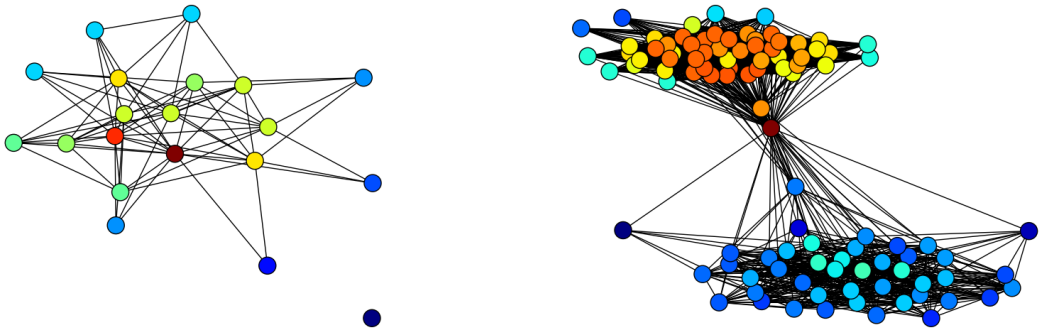
Las redes de la figura 3 son un ejemplo de dichas configuraciones iniciales.



(a) 20 nodos (b) 100 nodos

Figura 3: Distintos tipos de configuraciones iniciales

A partir de dicha configuración inicial, se permite a los agentes realizar tres acciones: colaborar (o, si no, traicionar), crear un nuevo enlace, y cortar un enlace. Las acciones de los agentes se basan en reglas que dependen de su propio historial de cooperación y del de sus vecinos en un momento dado. Las reglas se presentarán en detalle más adelante. Tras haber elegido cada nodo la elección para cada acción, se pasa a la siguiente ronda, la cual tiene asociada una red distinta con los nuevos enlaces creados. Tras haber recorrido todas las rondas, un ejemplo de las redes podrían ser las siguientes:



(a) 20 nodos (b) 100 nodos

Figura 4: Distintos tipos de configuraciones finales

Como se verá más adelante, en función de los parámetros que se fijen en el sistema, este evolucionará a una red u otra. Los detalles de los parámetros y cómo se detallarán en los capítulos 2 y 3.

En el artículo [2] mencionado anteriormente se tenían en cuenta varias situaciones que no se han modelizado en el presente trabajo. Una de las situaciones que no se ha tenido en cuenta en el modelo es la posibilidad de falsear la reputación de uno mismo, es decir, los participantes gastaban dinero para que su cooperación observable incrementase, aunque ellos siguieran traicionando.

Capítulo 2

En este capítulo se describirá detalladamente, con la ayuda de diagramas de flujo, el funcionamiento del programa, su estructura y qué se quiere conseguir con ello, a la vez que se comenta el código del mismo.

Estructura y funcionamiento del programa

En una primera aproximación, se presenta en la figura 5 un diagrama de flujo con la estructura general del programa creado.

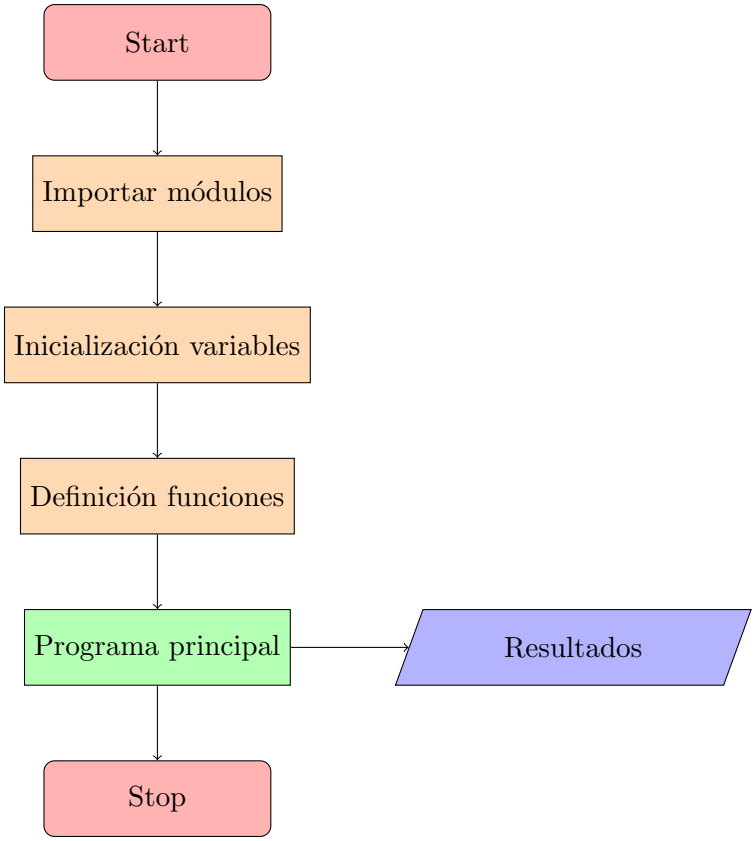


Figura 5: Flujo básico del programa

Como se puede observar, el programa tiene una estructura general de tipo lineal; es decir, una estructura en la cual la secuencia de acciones se va ejecutando en el orden en el

que han sido escritas. Algunos de los nodos del diagrama anterior son, a su vez, programas cíclicos, como por ejemplo el programa principal, que se ejecutan un número determinado de veces, y después se continúa la ejecución del programa.

Importar módulos

El bloque del código dedicado a *importar módulos* tiene como finalidad incorporar al programa funciones de paquetes especializados, las cuales facilitan en gran medida la programación del modelo. Entre las funciones de los paquetes, se encuentran: la gestión de redes (Networkx), la generación de números aleatorios (Random), operaciones matemáticas complejas (Math), los métodos numéricos más empleados (Numpy), la generación de gráficas con los distintos resultados (Matplotlib), el uso del propio sistema operativo (os) y la evaluación de datos procedentes de fuentes externas (ast).

La **implementación** es bastante sencilla. Para importar los módulos en Python, el comando empleado es *import* seguido del nombre del paquete a importar, con un posible sobrenombre con el cual se referirá al paquete a lo largo del programa.

Importar un paquete

```
import networkx as nx
```

Inicialización de las variables

En la sección del código dedicada a *inicializar las variables*, se definen los valores iniciales de las mismas para que Python reconozca los tipos de datos y pueda trabajar correctamente con ellos. Entre los tipos de variables, las más comunes son las listas (denominados *arrays* en otros lenguajes), aunque las listas de listas (implementando así la idea matemática de matriz) son también muy empleadas. El tipo de dato que pueden almacenar va desde *caracteres* hasta *grafos*, pasando por *booleanos* y números en *coma flotante*.

Para **implementar** la creación e inicialización de variables en Python, se emplea la asignación con el operador =. En realidad no solamente se emplea el operador = para crear variables, sino para cualquier tipo de objeto, ya sea estándar o definido previamente por el usuario. Es posible realizar asignaciones anidadas, así como asignaciones condicionales con estructuras de código más complejas.

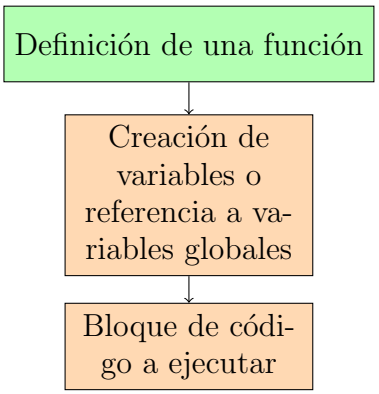
En el siguiente segmento de código, extraído del propio programa, se puede observar cómo se crea una lista de listas (es decir, una matriz), y cómo se asignan aleatoriamente caracteres a los elementos de una lista, de forma iterativa. Asimismo, se muestra cómo se crea un grafo regular de N nodos y grado d, con ayuda de un paquete externo.

Inicializar variables

```
red = nx.random_regular_graph(d, N)

vecRonda = [[4 for x in range(N)] for x in range(NUMrondas)]

for j in range(5):
  if random.random()>0.5:
    pastActions[i] = pastActions[i][1:5]+'c'
  else:
    pastActions[i] = pastActions[i][1:5]+'d'
```

Definición de las funciones

Existe una parte del código dedicada a *definir funciones*. Las funciones son bloques de código que se emplean con frecuencia, y que por conveniencia se agrupan en una solo bloque que puede ser ejecutado cuando se requiera. La estructura de una función consiste en el operador *def* seguido del *identificador* (o nombre de la función), acompañado de los parámetros que recibe dicha función. Tras nombrar la función, se crean las variables si son locales o se importan si son globales y finalmente se escribe el código que se desea agrupar.



A continuación se muestra un ejemplo con una función real del programa, la cual emplea variables definidas de forma global, y opera con ellas usando funciones de un paquete que no forma parte de Python.

Definición funciones

```
def reputacionmedia(ronda):
  // Importamos las variables globales
  global alfa, repmedia

  // Ejecutamos el código
  repmedia[ronda-1]=np.mean(alfa)
```

Programa principal

El siguiente diagrama muestra el flujo del cuerpo principal del código.

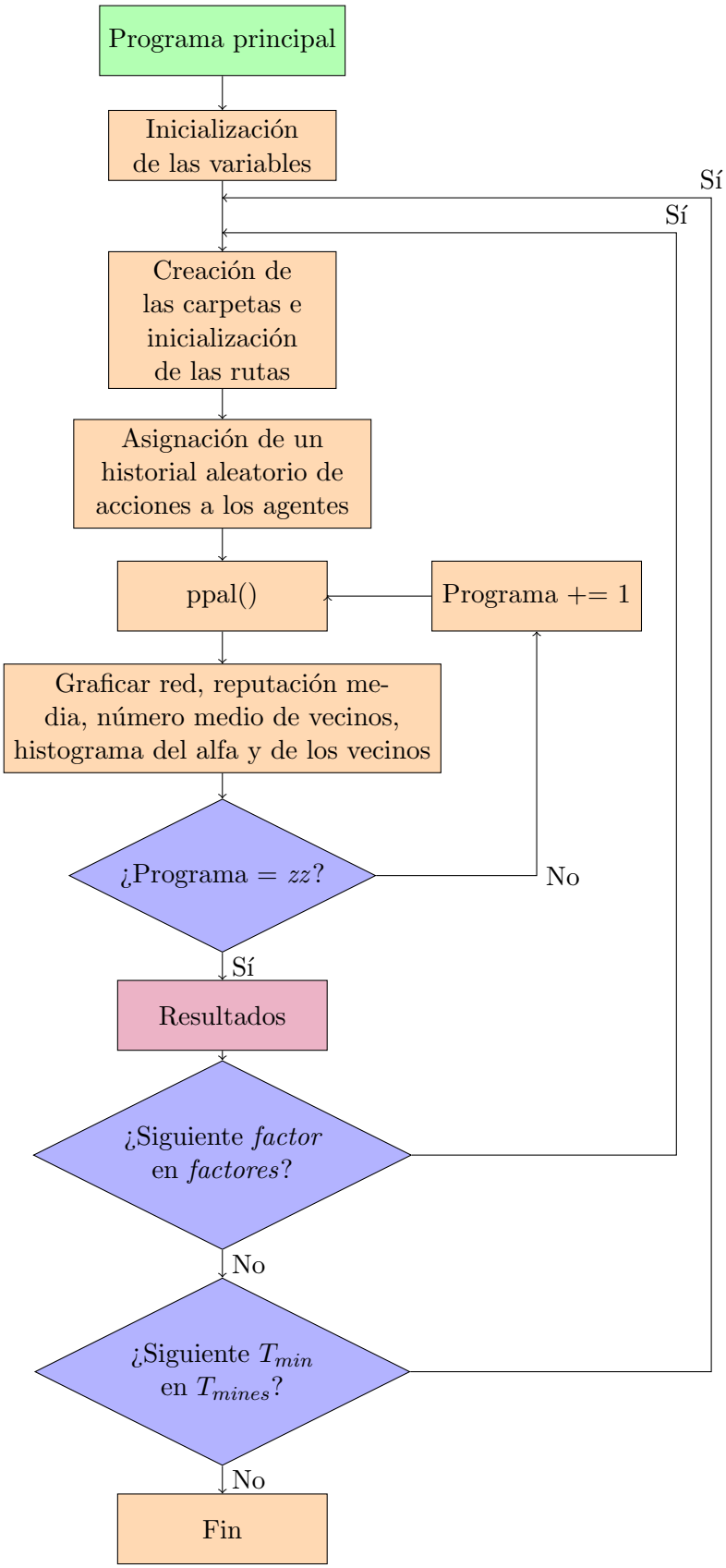


Figura 6: Flujo principal del programa

En la figura 6 se puede observar que el código consta de una parte de inicialización de las variables, necesarias tanto para los posteriores cálculos para gestionar esta parte del programa, como al llamar a la función *ppal()*; y también para poder presentar, posteriormente, los resultados mediante gráficas. Las funciones contienen los bloques de código que son ejecutados cada ronda, durante el número de rondas de juego definidas anteriormente; que a su vez se ejecutan dentro de cada programa. Esto se muestra en el diagrama mediante bloques de decisión (rombos morados), que en el código se implementan mediante bucles *for*.

Bucles *for* que controlan el flujo del programa principal

```
for Tmin in Tmines:
  for factor in factores:
    // Rutas de las carpetas
    [...]
    for zz in range(NUMprogram):
      // Asignaciones
      [...]
      ppal()
```

La finalidad de estos bucles es que el programa itere sobre todos los valores de factor y T_{min}, que se han definido previamente en unas variables de tipo *list*, para facilitar así el estudio de diversos parámetros de forma automática.
El código funciona de la siguiente manera: primeramente se fija el primer valor de T_{min} de la lista $T_{min}es$, y para dicho valor se selecciona el primero de la lista *factores*. Con dichos valores, se realizará un número determinado de iteraciones del programa (zz), guardando los resultados de cada iteración para su posterior análisis. Cuando se termina, se pasa al siguiente valor de la lista *factores* y se realizan de nuevo las zz iteraciones. Cuando se han recorrido todos los valores para *factor*, se pasa al siguiente valor de T_{min} de la lista y se vuelve a comenzar el proceso.
Al haber iterado sobre todos los posibles pares de valores de T_{min} y *factor*, se pasa a la parte del código en la que, con los archivos generados en cada iteración, se ponderan los resultados de todas las iteraciones y se grafican los resultados.

Reglas

Se ha intentado que el modelo se rija por una serie de leyes sencillas, sin por ello perder capacidad para variar las propiedades ni capacidad de análisis sobre el sistema. Siguiendo la idea anterior, se han definido tres grandes reglas por las que se rige el comportamiento general de este modelo, y afectan las siguientes acciones: a la decisión sobre si se colaborará o no con los vecinos, sobre si aceptar o rechazar un enlace, y sobre si se debe cortar o mantener un enlace.

La **primera regla** se ha implementado en el código de la siguiente forma:

Código primera regla

```
if len(vecinos)!=0:
  if random.random()<=factor*(reputacionmediavecindario(i)/5):
    colabSig[i]=True
  else:
    colabSig[i]=False
else:
  colabSig[i]=True
```

La función a la que se llama es:

Código función *reputaciónmediavecindario*

```
def reputacionmediavecindario(nodo):
  global red, alfa, repmediavecindario
  vecs = red.neighbors(nodo)
  temp = 0
  temp2 = 0

  for yy in vecs:
    temp = temp + alfa[yy]
    if len(vecs)>0:
      temp2 = temp/len(vecs)
      repmediavecindario = temp2

  return temp2
```

Como se puede observar en el código, la finalidad de *reputacionmediavecindario* es calcular, a partir de α, el número medio de veces que colaboran los vecinos de un nodo, ya que alfa es el parámetro que indica cuántas veces se ha colaborado en las últimas cinco rondas. El valor de alfa es parte de la información proporcionada, cada ronda, a cada agente sobre sus vecinos.

Al mismo tiempo, el valor que devuelve la función *reputacionmediavecindario()* es

usado en el programa principal. Como dicho valor se calcula sobre una historia de cinco acciones pasadas, para normalizar el valor se divide entre cinco, y posteriormente se multiplica por un valor, que en el código está representado por la variable factor, para simular distintas tendencias a la hora de interactuar en la red. Esto quiere decir que cuanta más colaboración se vea por parte de los vecinos de un nodo, para valores más altos de factor más propenso será este a colaborar.
El valor resultante de las operaciones anteriores, que estará comprendido entre cero y uno, se compara con una función que devuelve un valor aleatorio entre cero y uno, para conseguir cierto nivel de ruido y aleatoriedad, ya que con sujetos reales no se observa un comportamiento basado estrictamente en la información que se le proporciona sobre el sistema. Si el valor aleatorio es menor que el valor total calculado anteriormente, se coopera; y en caso contrario se elige la opción de traicionar. Todo esto se lleva a cabo si el nodo sobre el que se están haciendo los cálculos tiene algún vecino, de otra forma siempre se elegirá la opción de colaborar, intentando así generar una reputación positiva que atraiga a otros jugadores.

La segunda regla y la tercera regla son, ambas, esencialmente la misma idea.

La **segunda regla** se implementa de la siguiente forma:

Código segunda regla

```
if ((aleat1!=i) and (not (aleat1 in vecinos))):
  if random.random() <= probAceptar(i):
    if random.random() <= probAceptar(aleat1):
```

Y la **tercera regla** queda así en el código:

Código tercera regla

```
if ((p==(i,minimo)) and (random.random()<=probCortar(i))):
```

En la *segunda regla*, primeramente comprueba que el enlace propuesto aleatoriamente no sea consigo mismo, ni que tampoco exista ya dicho enlace. Además, siguiendo la idea anterior, se compara un número aleatorio con el valor devuelto por *probAceptar*.

Esta es una función cuyo código es el siguiente:

Código *probAceptar*

```
def probAceptar (nodo):
  global alfa, Tmin
  delta = reputacionmediavecindario(i)

  if (alfa[nodo] <= Tmin):
    prob = 0
  elif delta <= alfa[nodo]:
    prob = 1
  else:
    prob = (alfa[nodo]-Tmin)/(delta-Tmin)

  return prob
```

La idea es conseguir una probabilidad de aceptar (o de rechazar) según se haya colaborado más o menos en la historia de las rondas anteriores, que traducido al código del programa sería basarse en una *probabilidad de aceptar que varíe según el alfa del nodo.* Según el alfa que este tenga, y un parámetro fijado arbitrariamente (T_{min}), se consigue una curva como la de la figura 7.

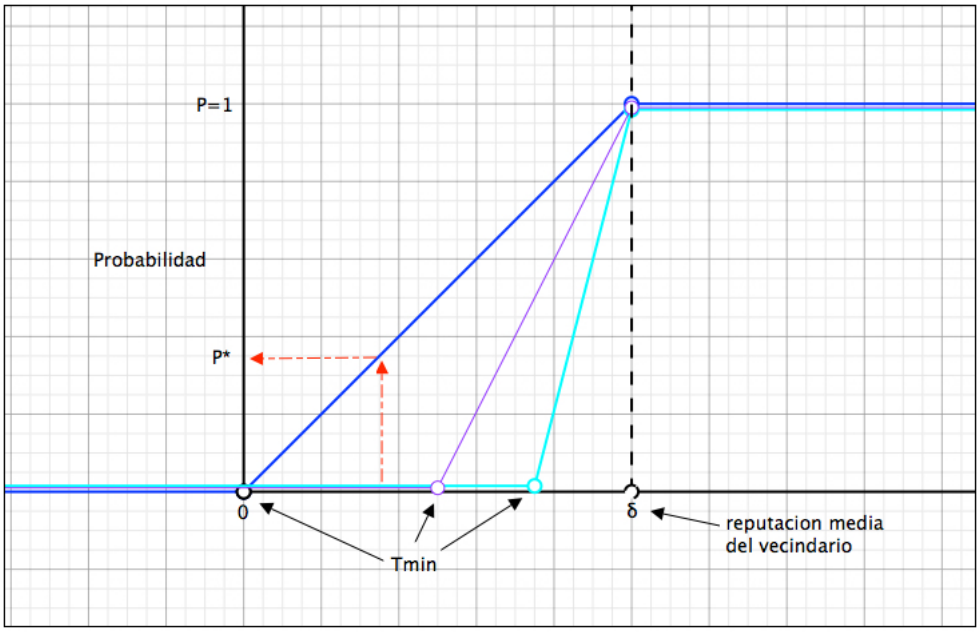


Figura 7: Función de la probabilidad según el alfa

El significado de esta curva es el siguiente: se entra con un valor de alfa a la curva y si dicho valor es menor que el parámetro T_{min} fijado arbitrariamente, la probabilidad que devuelve es siempre cero. Si el valor de alfa es mayor que la reputación media del vecindario del nodo (parámetro δ) sobre el que se está iterando, la probabilidad de enlazar

siempre será uno. En cambio, si el valor de alfa del nodo está comprendido entre delta y T_{min}, la probabilidad vendrá dada por una recta que une T_{min} con delta, y cuya ecuación es la siguiente:

$$probabilidad_{aceptar}[nodo] = \frac{\alpha[nodo] - T_{min}}{\delta[nodo] - T_{min}} \quad (1)$$

La *tercera regla* viene implementada por el código presentado anteriormente, y en él se llama a la función *probCortar*. Su código es el que sigue:

Código función *probCortar*

```
def probCortar(nodo):
    probAux = probAceptar(nodo)
    prob = 1 - probAux

    return prob
```

Esto se interpreta como que la probabilidad de cortar es la complementaria de aceptar. Por ello, si un nodo tiene mucha probabilidad de aceptar un enlace, tendrá por ello poca de romper uno.

Función *agente()*

La función *agente()* es llamada desde dentro de la función *ppal()*. Su función es gestionar la red a través de las rondas. Entre las tareas que desempeña, está la de crear nuevos enlaces, romper los que sean necesarios, todo ello siendo coherente con las reglas impuestas al modelo, y generar los archivos que luego se emplearán para estudiar la evolución de la red.

Es importante aclarar que el proceso de esta función es *síncrono* desde el punto de vista de los nodos; es decir, en cada ronda se va nodo a nodo calculando sus parámetros y si enlaza con otro vecino o elimina algún enlace, pero los agentes solo ven los enlaces con los cuales comenzó la ronda. Cuando se han recorrido todos los nodos, se actualiza la red con los nuevos enlaces, y así los agentes pasan a ver los enlaces creados o eliminados con las decisiones de la ronda anterior.

En el diagrama se puede observar que, como se ha comentado anteriormente, el proceso implementado en la función *agente()* es síncrono. Este proceso se intentó implementar en un principio con una biblioteca de gestión de eventos temporales discretos *Simpy*, pero al contar con poca documentación al respecto y debido a los problemas que daba su implementación, se decidió modelar esta parte del código sin la ayuda de ningún paquete externo.

El código completo de esta parte del programa puede verse en el anexo del código A y en el siguiente capítulo.

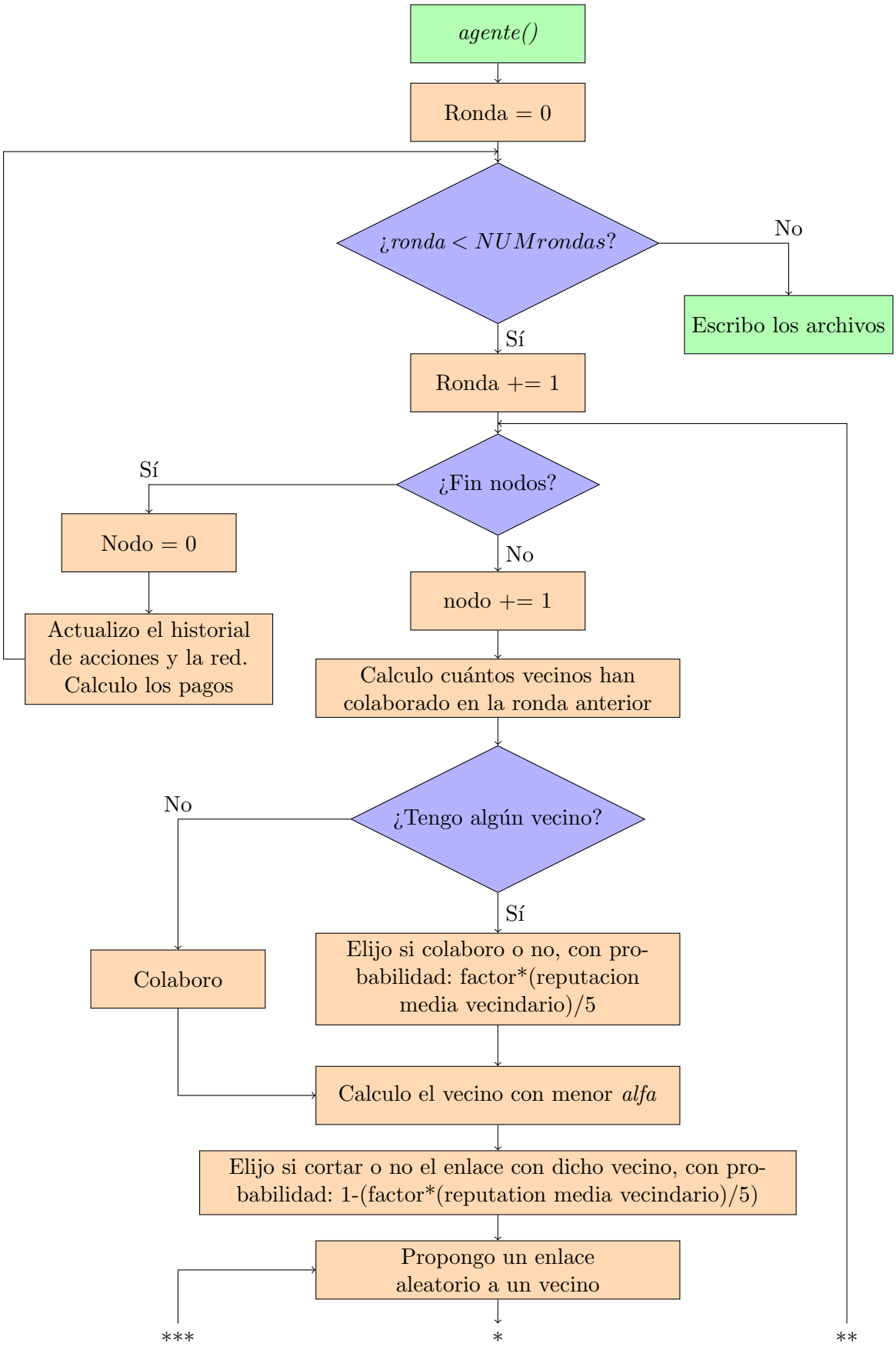


Figura 8: Flujo de la función agente. Parte 1

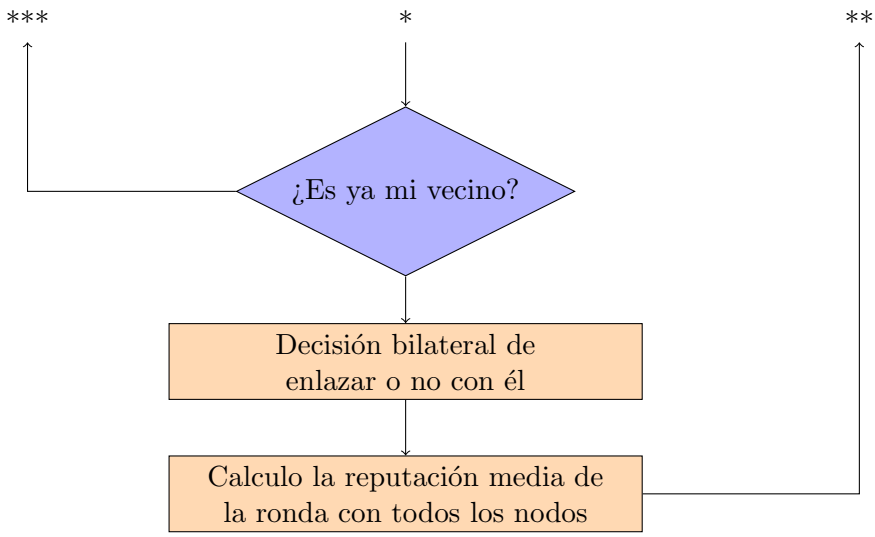


Figura 9: Flujo de la función agente. Parte 2

Generación de resultados

Como se puede observar en el diagrama del programa principal, cada vez que se terminan los *zz* programas definidos previamente (en los resultados que se muestran más adelante, se han tomado 10 programas), se procede a producir unos resultados (guardándolos en ficheros externos), los cuales se emplearán después para generar las gráficas e histogramas con los que se explicará el comportamiento del modelo.

Hay dos fases principales de generación de archivos y tratamiento de datos durante el estudio del modelo: una primera, donde se generan los resultados individuales de cada ronda para el grado medio de los nodos, la colaboración media y el ratio de colaboraciones en la última ronda; y otra, donde se ponderan todos los resultados del primer tipo y se genera las gráficas correspondientes.

Para conseguir el primer tipo de ellos, se parte de los archivos, mencionados anteriormente, los cuales se han generado en cada ronda. Dichos archivos contienen todos los valores del grado medio de los nodos, la colaboración media y el ratio de colaboraciones de los agentes. Se lee cada archivo y se crea otro intermedio que va almacenando los valores leídos. A partir del archivo intermedio, se procede a evaluar los datos. Estos se evalúan de forma segura mediante la función *literal eval* del paquete *ast*, para que se lean los valores de forma correcta, transformándolos al tipo de dato que representaban en un principio, y evitando leer algún tipo de conjunto de caracteres que puedan bloquear el programa. En este modelo, se ha considerado generar la curva de las gráficas con el valor medio y mostrar el error estándar, para poder ver si las curvas son estadísticamente diferentes o siguen un comportamiento similar. Tanto al calcular los valores del grado medio como de la colaboración media, el valor medio y el error estándar se han hallado, respectivamente, con las siguientes fórmulas:

$$\langle c \rangle (ronda) = \frac{1}{R} \sum_{i=1}^{R} c_i(ronda) \tag{2}$$

$$\langle c^2 \rangle - \langle c \rangle^2 = \frac{1}{R} \sum_{i=1}^{R} (c_i(ronda) - \langle c \rangle)^2 \tag{3}$$

Para calcular el error estándar (σ) de las curvas, se emplea la siguiente fórmula:

$$\sigma = \frac{\sigma_c}{\sqrt{n}} \tag{4}$$

Donde n es el número de programas corridos y σ_c igual a:

$$\sigma_c = \sqrt{\langle c^2 \rangle - \langle c \rangle^2} \tag{5}$$

Posteriormente, para una mejor visualización y un mejor estudio de los resultados, se reúnen en una sola gráfica todas las curvas de las simulaciones para un único valor de T_{min} y todos los de *factor*; o para un solo valor de *factor* y todos los valores de T_{min}. Para conseguir esto, se parte de todos los archivos intermedios obtenidos en las fases anteriores y se representan en una sola ventana gráfica. Se puede ver un ejemplo claro en la siguiente figura 10.

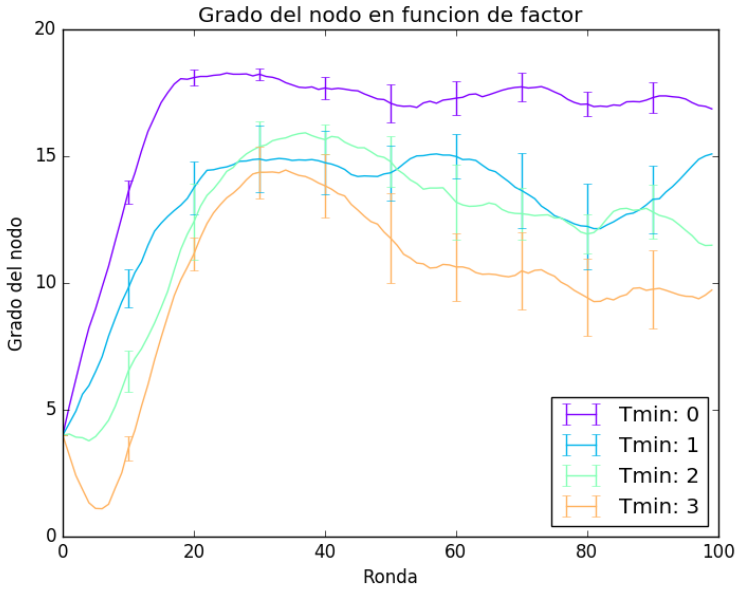


Figura 10: Ejemplo de varias curvas en una sola ventana gráfica

Capítulo 3

En este capítulo se presentan los resultados producidos por el modelo, explicando el comportamiento observado, por qué ocurre este comportamiento y cuales son los parámetros que hacen que el modelo se acerque más a los resultados experimentales del artículo [1].

Resultados del modelo

El modelo se ha diseñado de tal forma que, para una cantidad de nodos o agentes en la red y un número de rondas (ambos fijados arbitrariamente), se itere sobre una lista de valores de T_{min}; y para cada uno de ellos, que itere a su vez sobre una lista de valores de *factor*. Así se consiguen todas las posibles combinaciones de *factor* y T_{min}. Se han hecho dos tipos de simulaciones, una corta de 100 rondas, y otra más larga de 500 rondas. Dentro de cada una, la cantidad de nodos ha sido bien 20 o bien 100, con el fin de estudiar la influencia del tamaño de la red en su evolución. Los valores que ha tomado T_{min} han sido 0, 1, 2 y 3; y los valores de *factor*, 0.7, 0.9, 0.95, 1 y 1.05.

En algunos casos se presentan dos figuras para comparar el efecto de variar el parámetro T_{min} fijando un valor de *factor*. En estas figuras se presentan solamente los valores de T_{min} para *factor* igual a 0.95 e igual a 1, ya que se ha comprobado mediante las simulaciones que es entre esos dos valores donde ocurre el cambio de comportamiento.

20 nodos y 100 rondas

Comenzando por la simulación corta de *100 rondas*, y centrándonos en aquellas que tienen *20 nodos*, se pueden observar varios comportamientos en la figura 11.

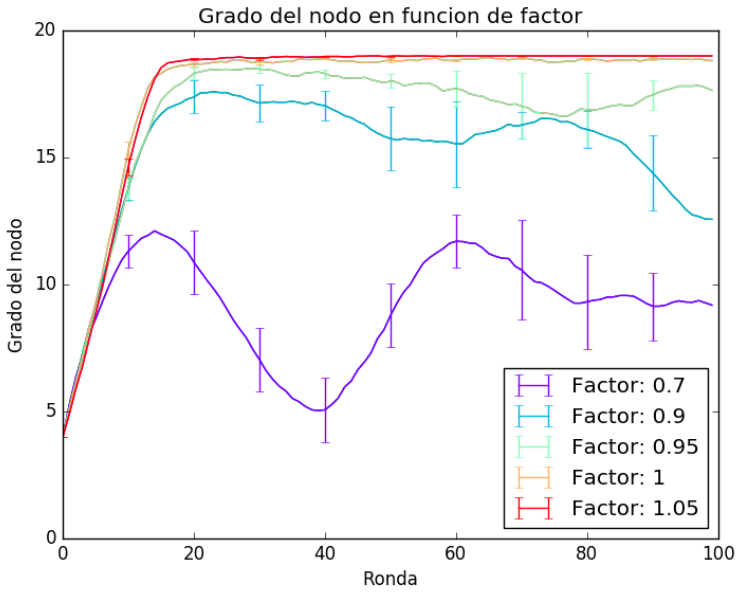


Figura 11: Grado del nodo en función de factor, con T_{min} fijo en 0

En las primeras rondas, el grado medio de los nodos aumenta rápidamente, y cerca de la ronda 20, comienza a bajar para establecerse en torno a un valor. Cuánto tarda en disminuir el grado es proporcional al valor de factor, es decir, a mayor valor de factor, más tarde comienza a descender el grado; y además lo hace hasta un valor más grande cuanto menor sea el parámetro factor. Cuando se alcanzan valores de factor cercanos a la unidad, la evolución de la red se acerca a la de un grafo completo: si la red tiene N nodos, cada nodo está enlazado con N-1 vecinos.

De igual forma que con el grado del nodo, la cooperación media de todos los nodos se ha calculado cada ronda, para distintos valores de T_{min}. Los resultados pueden verse en la figura 12.

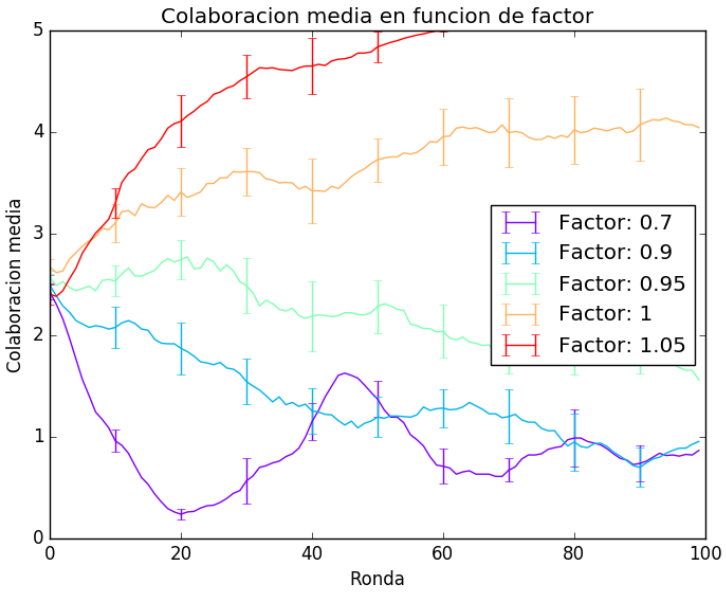


Figura 12: Número medio de colaboraciones función de factor, con T_{min} fijo en 0

El comportamiento observado en el grado del nodo se repite, de forma parecida, en la colaboración media a lo largo de las rondas. Para valores de factor menores que 1, la colaboración decae rápidamente pero no se anula: a menor valor de factor, más rápido decae. Cuando el valor se hace 1, la colaboración comienza a crecer a lo largo de las ronda, en lugar de a decaer, hasta que se hace prácticamente constante, con valor igual a 5 cuando *factor* vale 1.05. Según aumenta el valor de factor, el crecimiento es más rápido.

Para comprobar que este comportamiento no aparece súbitamente por alguna de las leyes que rigen el modelo, se ha obtenido, en cada ronda, la fracción de agentes que han elegido *colaborar* y no *traicionar* en su última acción.

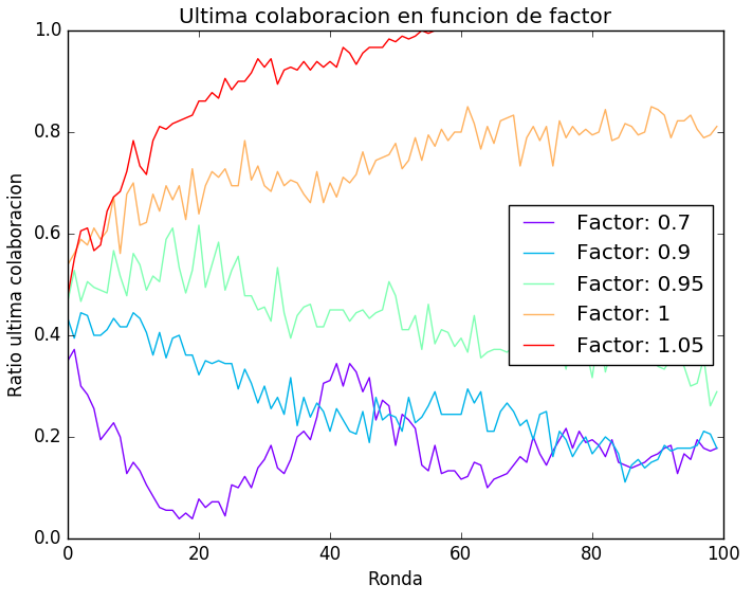


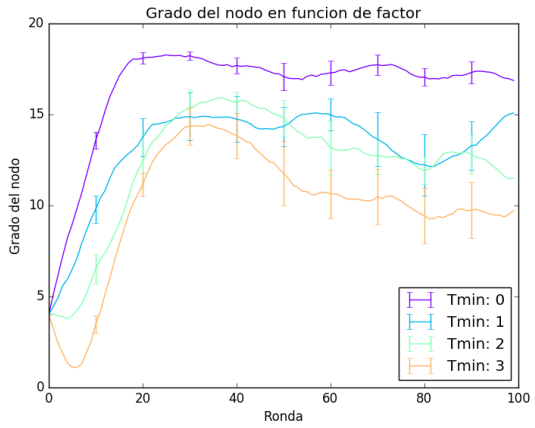
Figura 13: Evolución del ratio de colaboraciones función de factor, con T_{min} fijo en 0

Estas gráficas siguen un comportamiento parecido a las de la figura 12. Al aumentar el valor del parámetro factor la tendencia que se sigue es a aumentar la colaboración, y se observa que la cooperación va aumentando a lo largo de las rondas de forma continuada.

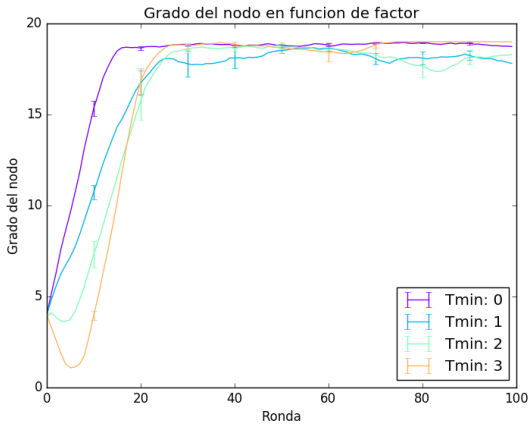
De estas gráficas podemos concluir que, si se mantiene fijo el parámetro T_{min}, es entre los valores para factor de 0.95 y 1 donde ocurre el verdadero cambio de comportamiento. Por debajo de 1 tanto el grado como la cooperación de los nodos tiende a bajar, y cuanto más se acerca a 1 más estable se hace, hasta que pasa a tomar el valor de 1 y tanto el grado como la colaboración comienzan a subir.

Para comprobar ahora la influencia del parámetro T_{min}, se han realizado las simulaciones manteniendo el valor de *factor* fijo, y variando T_{min} entre 0 y 3, con valores enteros.

Los resultados que se muestran a continuación solo son para valores de factor de 0.95 y 1, ya que, como se ha comentado anteriormente, es donde se produce el cambio notable de comportamiento y donde, por ende, habría que estudiar el sistema.



(a) Factor 0.95

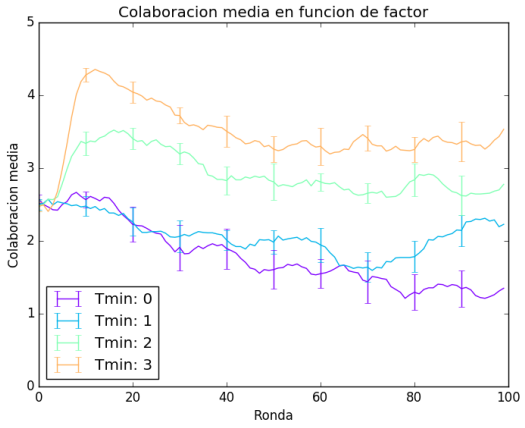


(b) Factor 1

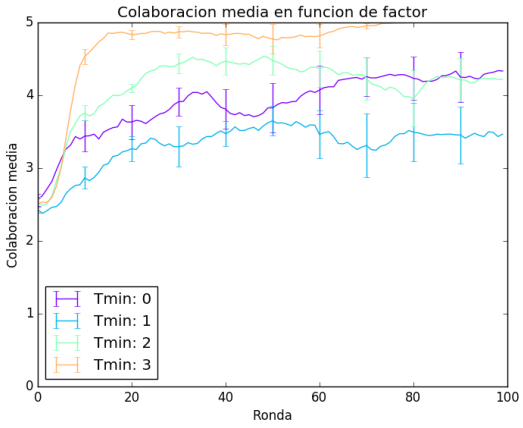
Figura 14: Evolución del grado del nodo en función de factor, para distintos valores de T_{min}

Se puede observar que, según aumenta el valor de T_{min} la *inestabilidad* del sistema lo hace con él. Esto quiere decir que, en cuanto al grado del nodo, se produce un descenso del valor en cuyo entorno se termina estabilizando, además de crecer más lentamente y de existir un mayor cambio de su valor a lo largo de las rondas. Esto se debe a que al aumentar el T_{min}, la función que da el valor para aceptar o no el enlace comienza a tomar valores, casi exclusivamente, que son 0 ó 1. Para valores de factor igual a 1 o mayores, la variación de T_{min} apenas tiene influencia en el sistema; solamente se observa, al comienzo, una disminución del grado, pero luego se recupera hasta los valores típicos.

Analizando de igual forma la cooperación media por ronda, se puede observar que si se aumenta el valor de T_{min}, el valor en el cual se estabiliza a lo largo de las rondas, también aumenta. Pero, contrariamente al grado del nodo, en este caso no se produce un aumento de inestabilidad.



(a) Factor 0.95

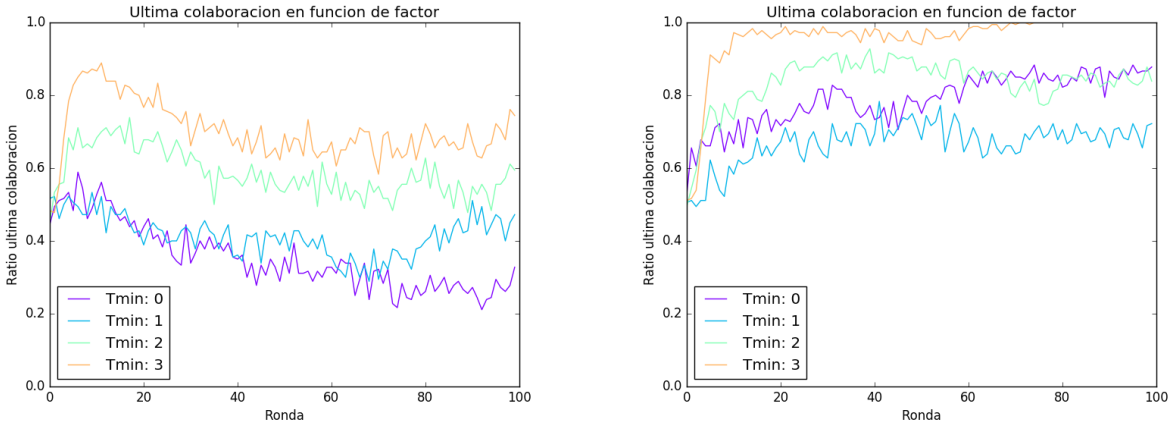


(b) Factor 1

Figura 15: Evolución de la colaboración media en función de factor, para distintos valores de T_{min}

Es interesante observar que cuando *factor* vale 0.95, la cooperación media tiende a descender, o al menos a estabilizarse en un valor y no crecer. Pero cuando se pasa a *factor* igual a 1, el valor de la colaboración media comienza a crecer, independientemente del valor que tome T_{min}.

Para ver que este es un comportamiento que no ocurre de forma espúrea, comparamos el número medio de cooperaciones con el ratio de cooperación en la última ronda.



(a) Factor 0.95 (b) Factor 1

Figura 16: Evolución del ratio de colaboraciones en función de factor, para distintos valores de T_{min}

Observamos que la tendencia de la última colaboración de cada ronda sigue a la media de colaboraciones en cada ronda, luego podemos concluir que el comportamiento mostrado por el sistema es coherente.

100 nodos y 100 rondas

Si ahora queremos estudiar el efecto del tamaño de la red en la evolución del sistema, tendremos que elegir aumentar el número de agentes involucrados. Se ha elegido arbitrariamente un número de nodos de la red de 100. La elección de este número está sujeta a que la red sea bastante mayor que la estudiada, pero no tanto como para que se pierdan por completo (por efecto de su tamaño) los parámetros de estudio.

En la figura 17 se puede observar la evolución de la red según varíe factor, para un valor fijo de 0 del parámetro T_{min}.

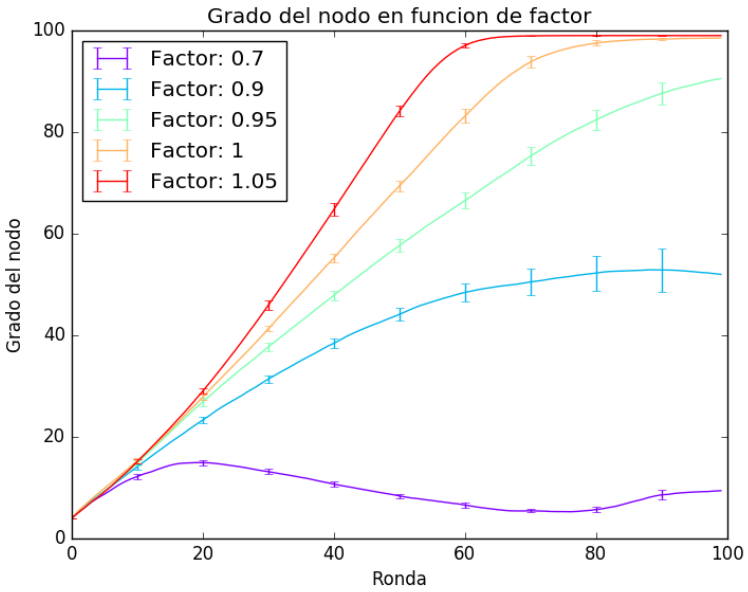


Figura 17: Grado del nodo en función de factor, con T_{min} fijo en 0

En el primero de los casos, fijando T_{min}, para el valor de *factor* = 0.7, el comportamiento que se observa es que crece el grado y luego decae para establecerse en torno a 10, aproximadamente. Para factor = 0.9, en cambio, se observa un comportamiento distinto al observado en redes más pequeñas: el grado crece hasta estabilizarse en torno a 80. Esto se comprobará más adelante que no es correcto: decae más tarde que con *factor* 0.7, pero termina decayendo; es el pico inicial de colaboración que es observado en todas las simulaciones. Cuando fijamos valores de *factor* más altos, como 1 y 1.05, el comportamiento es ligeramente distinto: ahora crece hasta estabilizarse muy cerca de 100. La única diferencia es que con 1, el grado medio crece ligeramente más lentamente que con 1.05.

Pasando a estudiar el efecto del tamaño en la cooperación media, los resultados obtenidos son los siguientes:

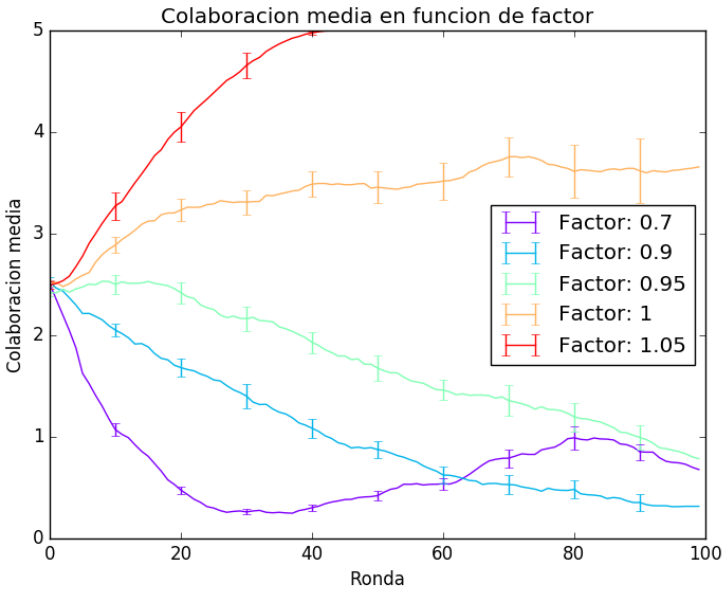


Figura 18: Número medio de colaboraciones función de factor, con T_{min} fijo en 0

Podemos observar que con el valor de *factor* de 0.7, la cooperación desciende rápidamente hasta valores en torno a 0.5 ó 1. Conforme aumenta el valor de *factor*, el descenso de la colaboración media es menos acusado, aunque no por ello el valor al que se llega es mayor que 1. Cuando los valores de factor llegan a 1, la cooperación media a lo largo de las rondas comienza a crecer, y no solamente decrecen. Y como es de esperar, si aumenta el valor por encima de 1, el crecimiento de la cooperación se va haciendo mucho más rápido.

Comparando resultados, se observa que estos últimos son casi idénticos a los obtenidos con las redes de 20 nodos.

Al igual que anteriormente, los valores del ratio de agentes que han colaborado en la última ronda permiten comprobar que el comportamiento observado sea coherente. Los resultados son los siguientes:

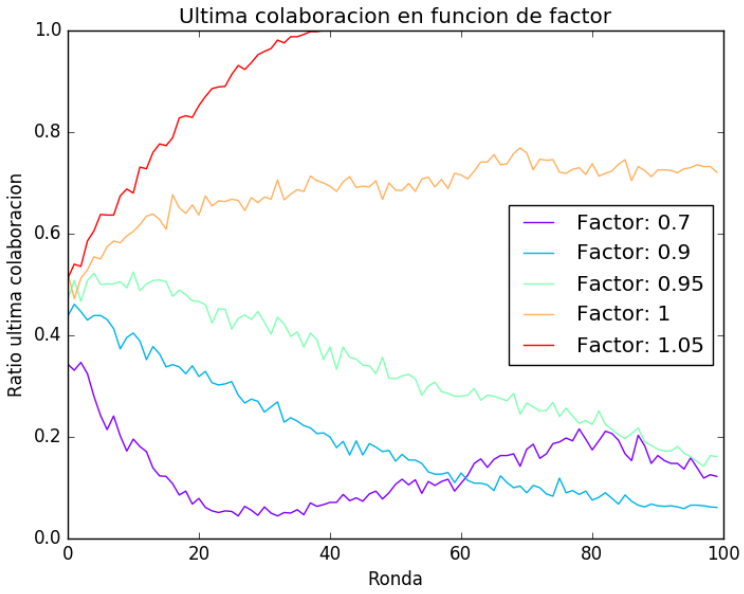
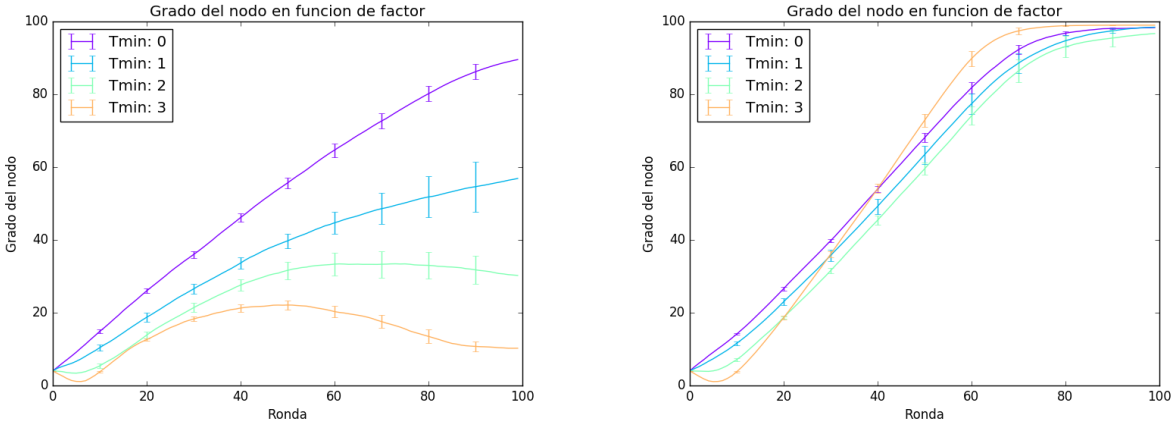


Figura 19: Evolución del ratio de colaboraciones función de factor, con T_{min} fijo en 0

Los valores del ratio siguen a los de la colaboración, luego el comportamiento es el esperado.

Si ahora fijamos el parámetro del factor pero variamos T_{min} para ver cómo afecta, los resultados obtenidos con respecto al grado del nodo son similares a aquellos que se obtuvieron para las redes de 20 agentes, lo que se puede observar en la figura 20.



(a) Factor 0.95

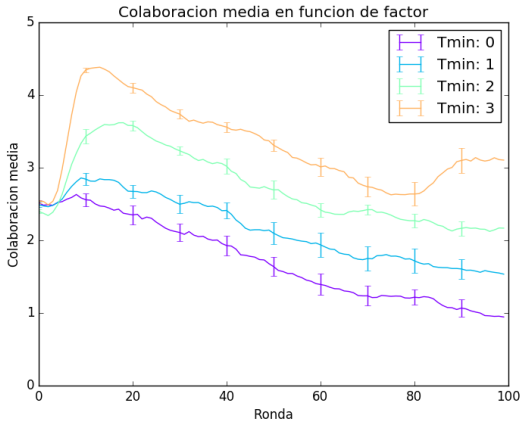
(b) Factor 1

Figura 20: Evolución del grado del nodo en función de factor, para distintos valores de T_{min}

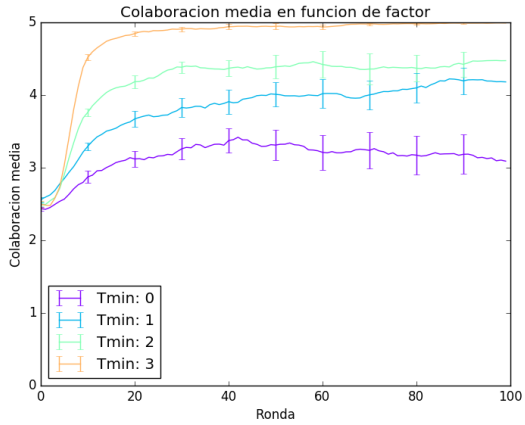
Se puede observar, al igual que en redes más pequeñas, que al aumentar el valor de T_{min}, aumenta la inestabilidad. La única diferencia es que esta inestabilidad (es decir, que el valor del grado del nodo varíe rápidamente a lo largo de las rondas sin estabilizarse) es menos acusada que en las redes de menor tamaño, están más suavizadas por el efecto de

tener una población más grande.

Si nos fijamos en el valor medio de la cooperación (figura 21), los resultados son muy parecidos, pero la *inestabilidad* observada es menos acusada porque el tamaño de la red suaviza los efectos.



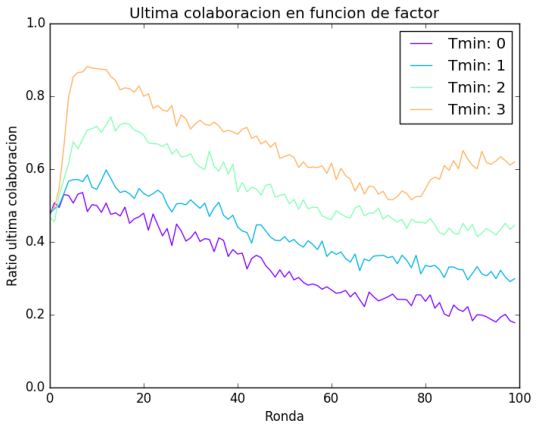
(a) Factor 0.95



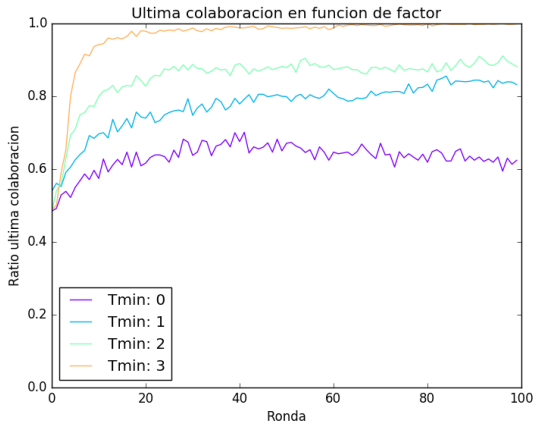
(b) Factor 1

Figura 21: Evolución de la colaboración media en función de factor, para distintos valores de T_{min}

La comprobación de que el comportamiento observado es coherente con su evolución, nos los dan las gráficas del ratio, en la figura 22.



(a) Factor 0.95



(b) Factor 1

Figura 22: Evolución del ratio de colaboraciones en función de factor, para distintos valores de T_{min}

Como el comportamiento del ratio sigue al de la cooperación media, podemos concluir, igual que anteriormente, que la evolución de la red es correcta, de acuerdo a las normas impuestas.

20 nodos y 500 rondas

Ahora se analiza si en las simulaciones de 100 rondas realizadas anteriormente, el numero ha sido suficiente para observar el comportamiento global del sistema. Al igual que antes, el número de rondas se aumenta arbitrariamente en un factor de 5, pasando a ser ahora *500 rondas* por programa; número con el cual se podría ver si el comportamiento cambia con respecto a lo observado anteriormente, o se alcanza un punto a partir del cual el tiempo no es un factor determinante en la evolución del sistema.

Comenzamos fijándonos en aquellas redes de *20 nodos* y, analizando de forma similar a la anterior el comportamiento seguido es el siguiente (variando *factor*, con T_{min} fijo). Esto se puede observar en la figura 23.

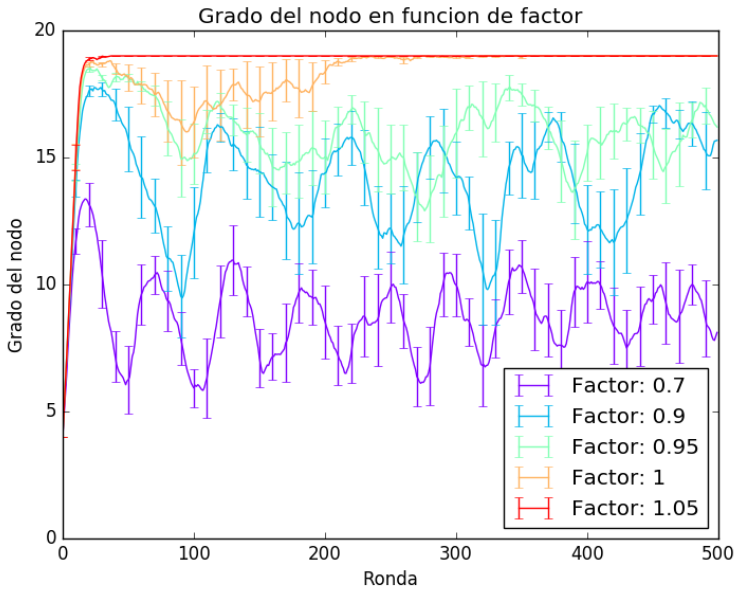


Figura 23: Grado del nodo en función de factor, con T_{min} fijo en 0

Se puede observar, al igual que antes, que el comportamiento hasta las 100 rondas es idéntico al de los resultados anteriores. A partir de las 100 rondas podemos observar que los valores en los que creíamos que se estabilizaban las simulaciones son, sin contar las fluctuaciones, correctos. También podemos concluir que a menor valor de *factor* mayor es la fluctuación entorno a dicho punto. También se puede ver en la figura 23 cómo cuando *factor* vale 1, entre la ronda 50 y 200, el grado comienza a decaer, pero se recupera rápidamente hasta formar la red completa y ya no decae.

También analizamos la influencia del tiempo en la cooperación media, y se puede observar el resultado en la figura 24.

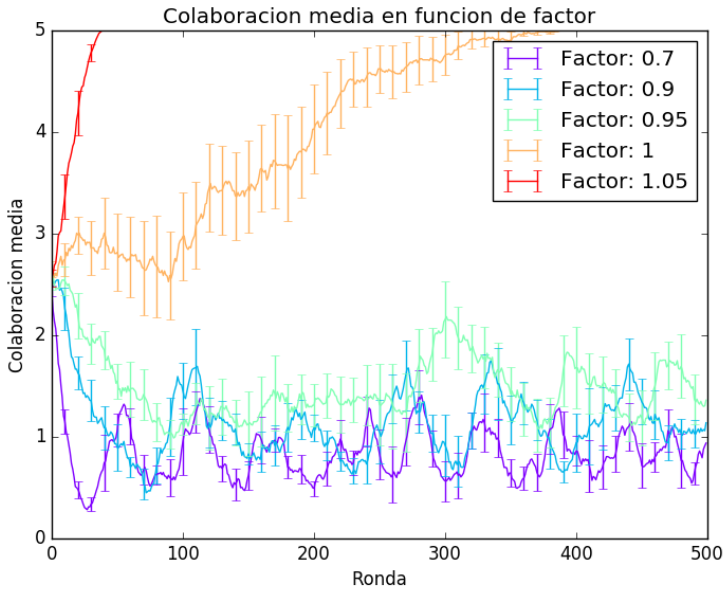


Figura 24: Número medio de colaboraciones función de factor, con T_{min} fijo en 0

Igual que ocurre con el grado, el comportamiento que se observa pasada la ronda número 100 es representativo de lo que ocurre en las 100 primeras. Es notable que en la figura 12 se observaba que la colaboración con *factor* igual a 1, crecía muy lentamente, y suponíamos que el valor llegaría hasta 1; cosa que en esta gráfica se puede ver claramente.

Al igual que antes, vemos si el ratio de cooperación cambia con un número elevado de rondas. Los resultados de las simulaciones se observa en la figura 25.

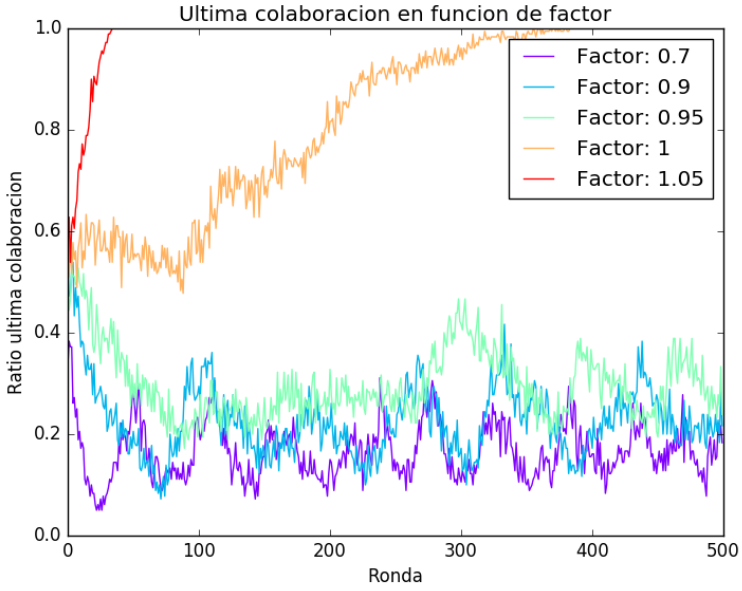
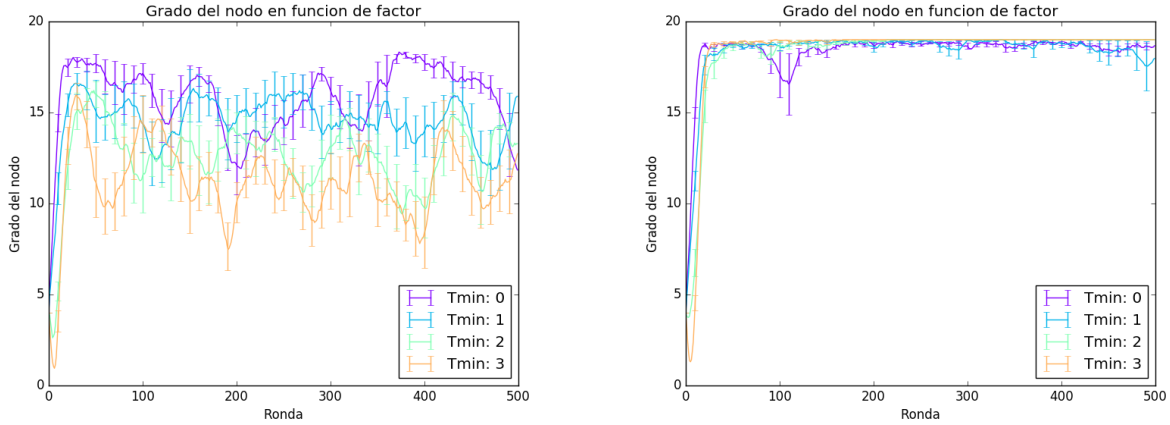


Figura 25: Evolución del ratio de colaboraciones función de factor, con T_{min} fijo en 0

Ahora procedemos a comprobar si, a largo plazo, se observa que la variación de T_{min}

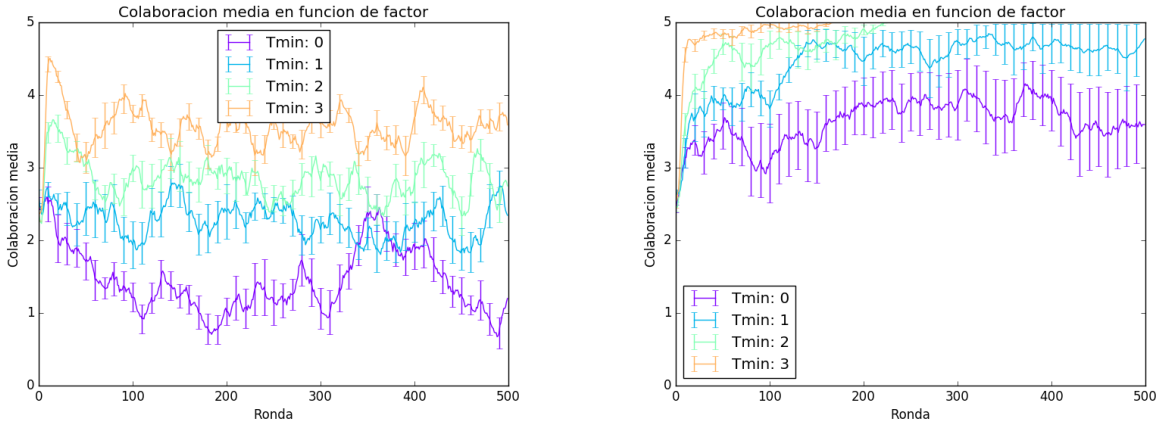
afecta a la evolución de la red, con un comportamiento distinto al de las 100 rondas. Los resultados para el grado del nodo con factor 0.95 y 1 se muestran en la figura 26



(a) Factor 0.95 (b) Factor 1

Figura 26: Evolución del grado del nodo en función de factor, para distintos valores de T_{min}

De estos resultados, podemos observar que el grado, cuando se varía T_{min} con un *factor* fijo, el comportamiento de 0 a 100 rondas es similar al de las 400 rondas sucesivas. Lo mismo ocurre con el comportamiento de la cooperación media. Los resultados se pueden observar en la gráfica 27

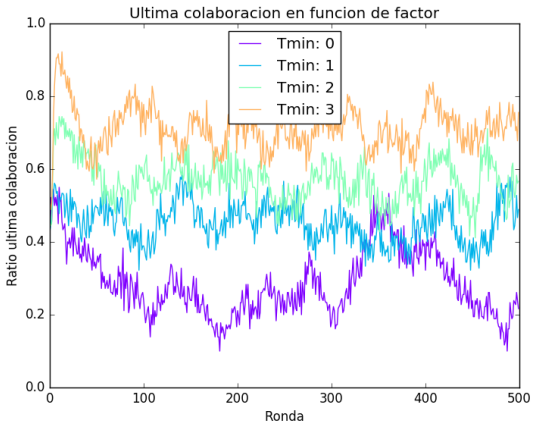


(a) Factor 0.95 (b) Factor 1

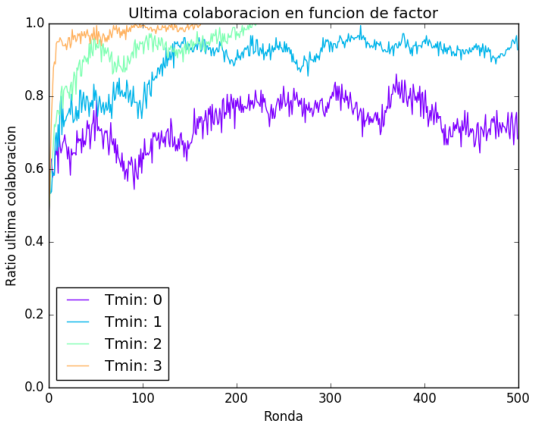
Figura 27: Evolución de la cooperación media en función de factor, para distintos valores de T_{min}

Vemos que el comportamiento de las primeras rondas es representativo de lo que ocurre en el resto de rondas. Podemos concluir, pues, que el tiempo, a largo plazo, no influye en gran medida en el comportamiento del sistema.

Como comprobación se observa si el ratio de colaboraciones sigue, a largo plazo, a la colaboración media de los agentes. Los resultados de las simulaciones se pueden ver en la figura 28.



(a) Factor 0.95



(b) Factor 1

Figura 28: Evolución del ratio de colaboraciones en función de factor, para distintos valores de T_{min}

Como el comportamiento del ratio es idéntico al de la última colaboración, la evolución del sistema es coherente con las reglas impuestas.

100 nodos y 500 rondas

Por último, en las simulaciones con 100 nodos durante 500 rondas, se pretende observar si, en las redes de gran tamaño, el comportamiento a lo largo del tiempo es muy distinto al de las primeras rondas. En la figura 29 los resultados que se observan son bastante similares a aquellos que se obtuvieron en solo 100 rondas (figura 17).

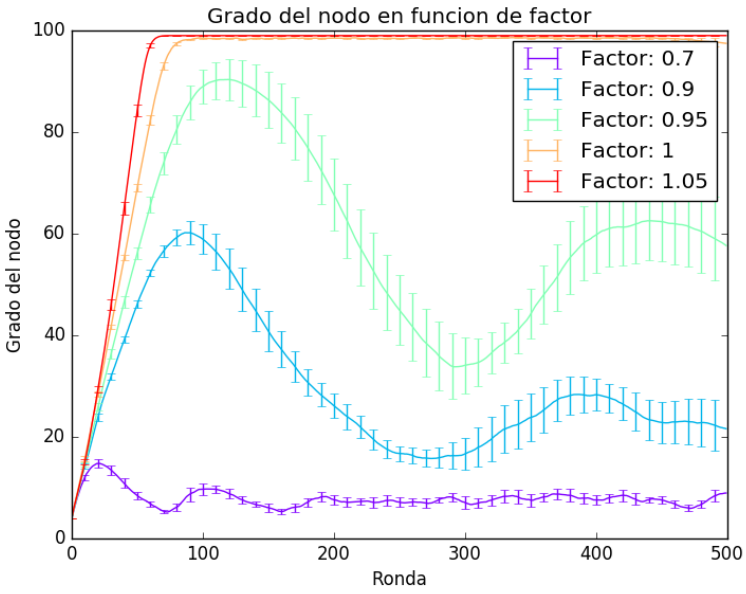


Figura 29: Grado del nodo en función de factor, con T_{min} fijo en 0

Algo interesante en esta gráfica es que, cuando *factor* vale 0.9, se observan aumentos puntuales del grado del nodo, que siempre acaban decayendo hasta el mismo valor. El comportamiento es el mismo que el de *factor* igual a 0.7, pero como los valores que alcanzan son mayores, los aumentos son también mayores. El comportamiento de *factor* igual a 0.95 es el mismo, solo que a una escala mayor.

Cuando el valor de factor es 0.9, en la gráfica de 100 rondas se observaba que el grado iba creciendo lentamente hasta que al final tomaba un valor muy alto (en torno a 70). En cambio, aquí se puede observar que no se comporta como cuando factor vale 1 ó 1.05, sino que es el pico inicial observado en todas las simulaciones, salvo que en este caso es mucho más ancho, y luego cae para estabilizarse en un valor más bajo.

El número de colaboraciones en este tipo de redes evoluciona igual que en la figura 18, por lo que aumentar el número de rondas no aporta más información sobre el sistema.

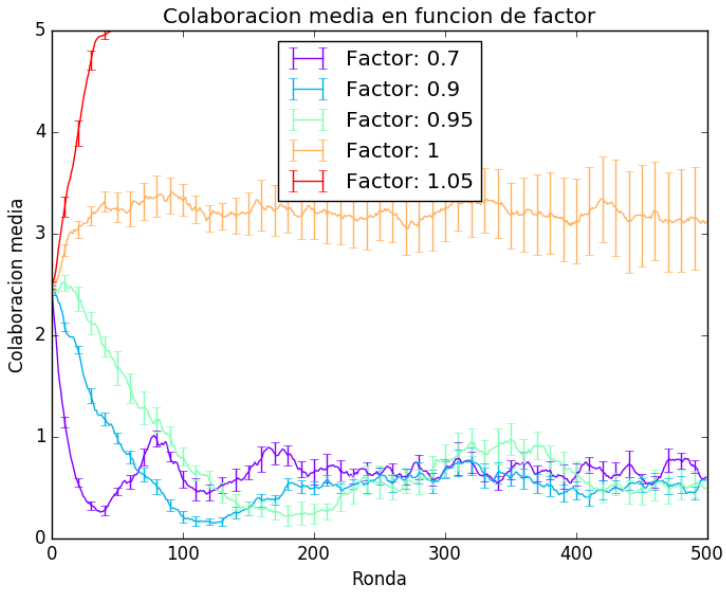


Figura 30: Número medio de colaboraciones función de factor, con T_{min} fijo en 0

Como el comportamiento pasadas las 100 rondas es idéntico al de las 100 primeras, lo único observable es que, en las simulaciones de 100 rondas se supuso que cuando *factor* es 1 crecería hasta llegar a 5, y en esta gráfica [30] se puede observar.

En el ratio (figura 31) observamos el mismo comportamiento que en el número medio de colaboraciones:

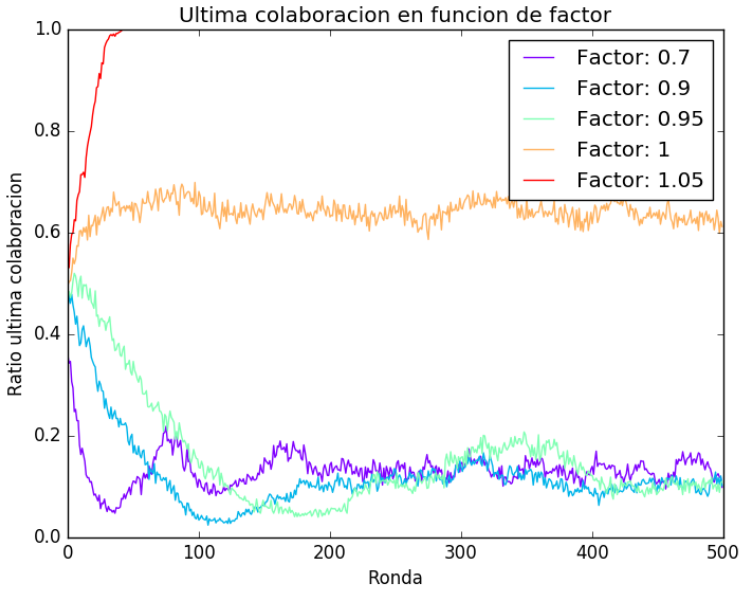
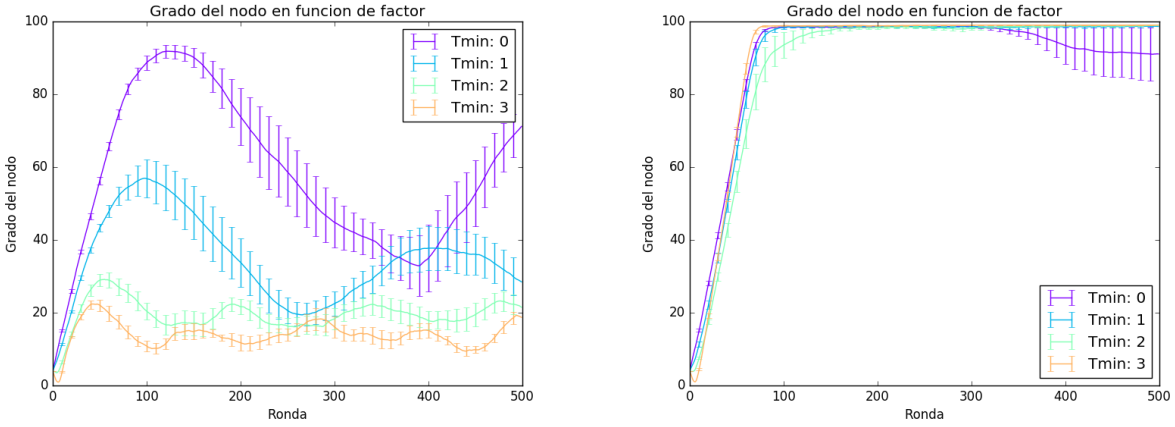


Figura 31: Evolución del ratio de colaboraciones función de factor, con T_{min} fijo en 0

Así pues, concluimos una vez más que un número elevado de rondas no aporta infor-

mación nueva sobre el sistema.

Ahora, igual que anteriormente, comprobamos si, a largo plazo, la variación de T_{min} afecta a la evolución de la red. Los resultados para el grado del nodo con *factor* 0.95 y 1 son los siguientes:



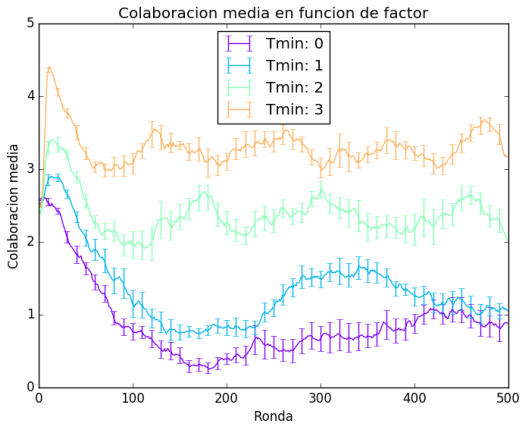
(a) Factor 0.95

(b) Factor 1

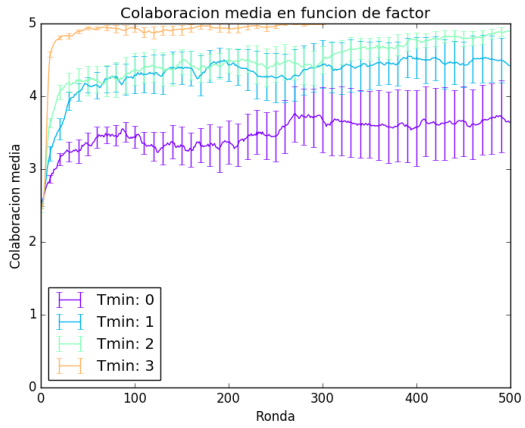
Figura 32: Evolución del grado del nodo en función de factor, para distintos valores de T_{min}

Los resultado presentados en la figura 32 son representativos de los 100 rondas de la figura 20, por lo que se puede concluir que un aumento de rondas no aporta información adicional sobre el sistema.

Para el número medio de colaboraciones en función de la ronda, con *factor* fijo y variando T_{min}, en redes de 100 nodos, se han aumentado las rondas a 500 para ver si el comportamiento cambia. Los resultados se presentan, para *factor* igual a 0.95 y 1, en la figura 33.



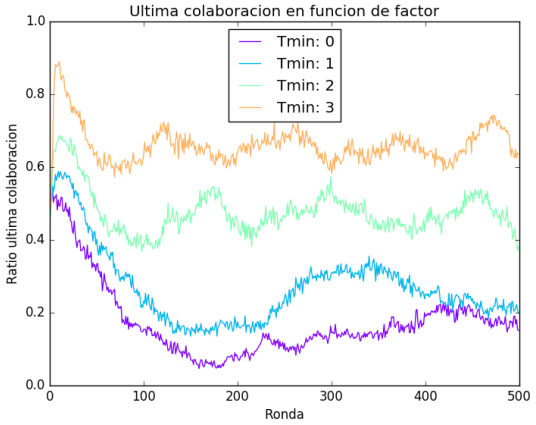
(a) Factor 0.95



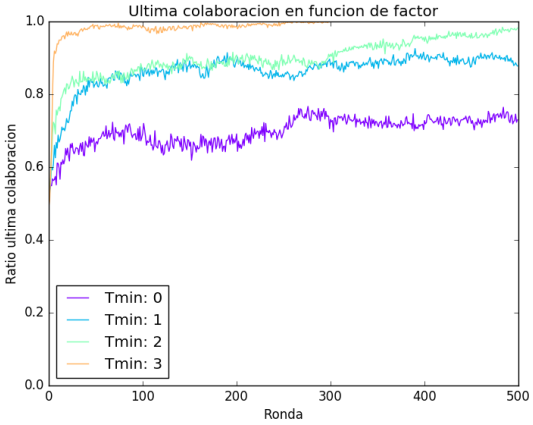
(b) Factor 1

Figura 33: Evolución de la colaboración media en función de factor, para distintos valores de T_{min}

Comprobamos también que ocurra lo mismo que con las redes pequeñas, que el ratio siga a la colaboración media, sin importar el número de rondas que se simulen. En la figura 34 se observa que esto se cumple.



(a) Factor 0.95



(b) Factor 1

Figura 34: Evolución del ratio de colaboraciones en función de factor, para distintos valores de T_{min}

De estas últimas simulaciones con valores grandes para el número de rondas, podemos concluir que a partir de un número elevado de rondas, el efecto del tiempo desaparece, y el resultado final de la simulación es idéntico para cualquier valor de las rondas.

Tiempos de cálculo

Se han hecho 4 tipos de simulaciones, las cuales han tenido distintos tiempos de procesamiento en un ordenador, en función de la cantidad de operaciones que debían realizar, que principalmente vienen determinadas por el número de nodos y de rondas sobre los que se debe iterar.

En cuanto al hardware disponible, se ha empleado un procesador i7-6700K, con una velocidad de reloj de 4GHz, sin aceleración de los cálculos mediante la tarjeta gráfica. Las redes pequeñas, como las de 20 nodos, tardaban alrededor de 15 minutos en iterar todos los parámetros durante las 100 rondas, y alrededor de 30 minutos en iterar las 500 rondas. Las redes más grandes, como las de 100 nodos, en las simulaciones de 100 rondas, el tiempo medio de las simulaciones era de 1 hora; mientras que las simulaciones largas de 500 rondas tardaban aproximadamente 2 horas en iterar sobre todos los valores de *factor* y T_{min}.

Este tipo de simulaciones podría acelerarse con hardware específico, empleando aceleración mediante tarjetas gráficas e incluso mejorando el código del programa, aunque hay que tener en cuenta que el aumento del tiempo de procesamiento es exponencial y a partir de una cantidad de nodos sería difícil disminuir el tiempo.

Conclusiones

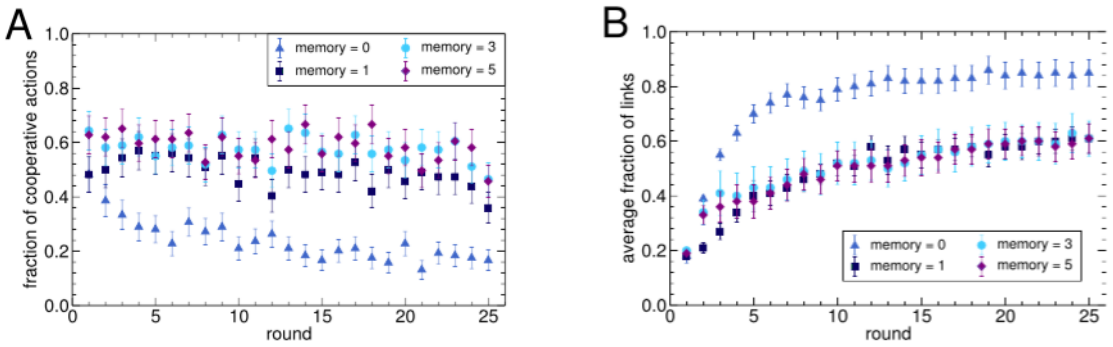
Se presentan a continuación las conclusiones extraídas del modelo, comparándolo con experimentos realizados por investigadores del campo, y se discuten las acciones que se podrían llevar a cabo con el modelo y desarrollos futuros que se podrían realizar en él.

Conclusión

Comparación

Una vez finalizada toda la producción de resultados del modelo, y comentadas sus propiedades en el capítulo anterior, se procede ahora a la comparación de los mismos con el artículo [1]. En él se estudian las redes mediante técnicas de la Teoría de Juegos. Concretamente se realizaron experimentos con grupos de gente, quienes jugaban iteradamente al Dilema del Prisionero en redes dinámicas, para acabar demostrando que es la reputación lo que realmente fomenta la cooperación en redes con una reconfiguración dinámica. Se realizaron 24 experimentos, involucrando en total a 243 individuos, distribuídos en grupos de 17 y 25 personas. Por ello, los experimentos los podemos considerar idénticos a aquellos con redes de 20 nodos. Las redes se configuraban dinámicamente, por lo que la implementación de eliminar y agregar enlaces equivale a esta situación. Las rondas de los experimentos fueron 25, que aunque no son las mismas que en el modelo, permiten comparar ambos resultados.

Los resultados se presentan es las siguiente gráficas. Éstas muestran la evolución de las redes con memoria 0, 1, 3 y 4 acciones pasadas, para el ratio de colaboraciones (figura 35A) y para el grado medio del nodo (figura 35B):

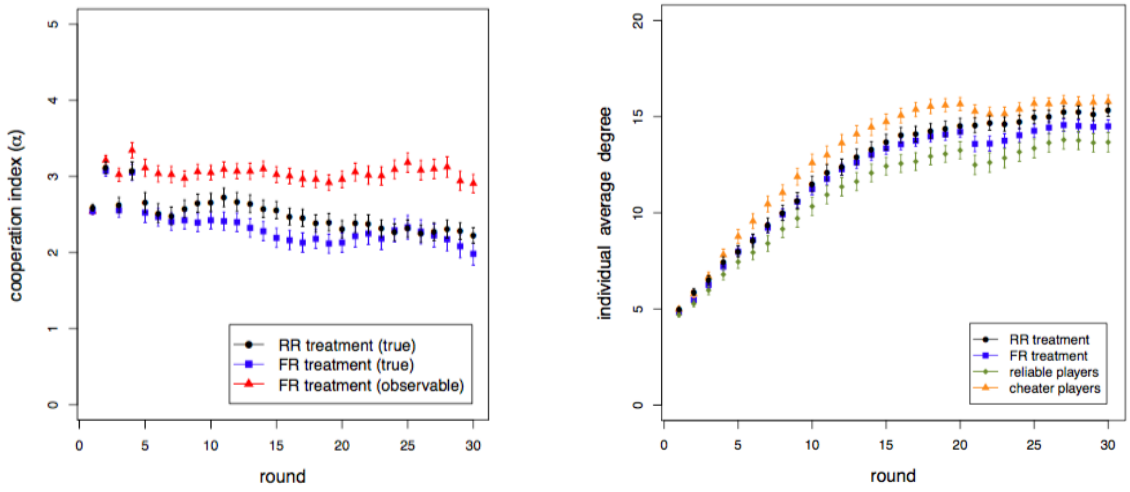


(a) Ratio de colaboraciones (b) Grado medio del nodo

Figura 35: Resultados del paper [1] - Ratio y grado medio

En el modelo, la memoria era constante a 5, no variaba, luego en las gráficas anteriores solamente es necesario fijarse en los rombos morados (fijarse en la leyenda).

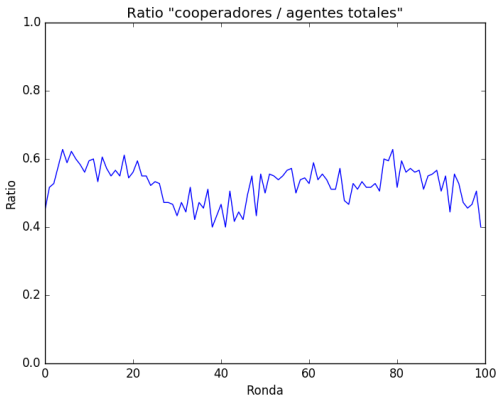
A continuación se muestran figuras del artículo [2] y del material complementario del mismo. Éstas muestran, para las redes con tratamiento real y falseado, la evolución del índice de cooperación α (figura 36A) y la evolución del grado medio del nodo (figura 36B):



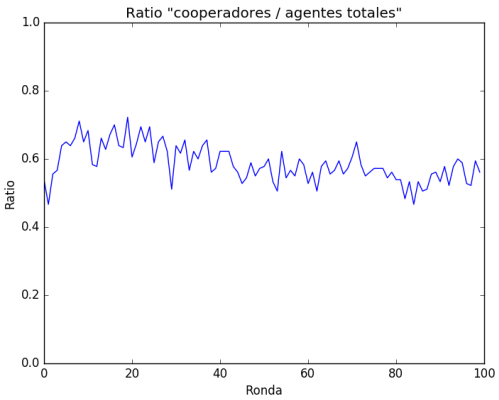
(a) Índice de cooperación (b) Grado medio del nodo

Figura 36: Resultados del paper [2] - Índice cooperación y grado medio

Ahora se presentan los resultados del modelo para los valores del ratio de cooperación en la última ronda, para redes de **20 nodos**.



(a) Factor 0.90 y Tmin 2

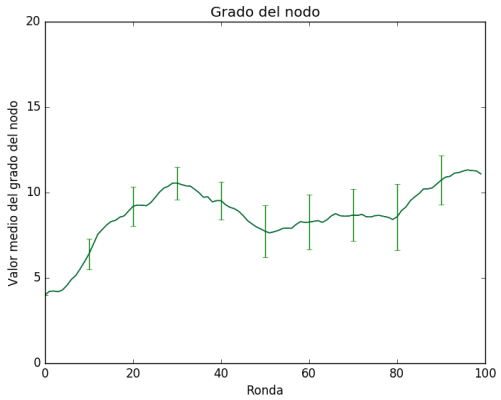


(b) Factor 0.95 y Tmin 2

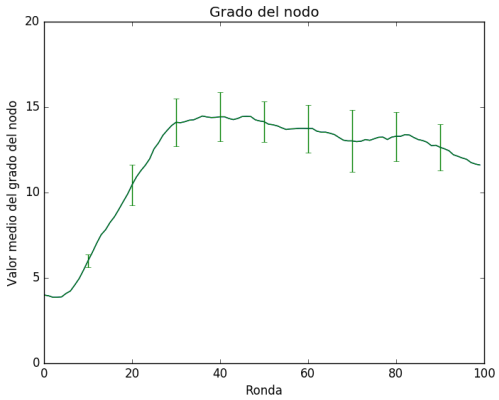
Figura 37: Resultados modelo - Ratio

Si nos fijamos en las gráficas de la figura 35, podemos observar que la cooperación media comienza en torno a 0.6, y se mantiene, más o menos constante alrededor de ese valor, fluctuando ligeramente. Atendiendo ahora a la figura 36, se puede ver que la reputación media (α) comienza cerca de 2.75 - 3, que es un 60 %, igual que anteriormente; y, aunque decae ligeramente, termina en un valor cercano al 45 % de la colaboración media.
Esto también se observa en las redes de 20 nodos, con un *factor* de 0.90 y 0.95 y un T_{min} de 2, en la figura 37. La figura 37A, de *factor* 0.9, se asemeja más al resultado del experimento de la figura 36, ya que comienza en 0.5, pero evoluciona de tal forma que finaliza en un valor cercano al 40 % ó 45 %. Sin embargo, la figura 37B comienza en un valor de 0.5, por haberla fijado arbitrariamente en dicho valor al comienzo; pero el valor crece rápidamente y se queda, fluctuando ligeramente, alrededor de 0.6-0.7, más que cerca de 0.5; por lo que se asemeja más a los resultados de las figuras 35.

Con respecto al grado del nodo de las redes con los mismos parámetros, los resultados (figura 38) son muy similares a los de los experimentos.



(a) Factor 0.90 y Tmin 2



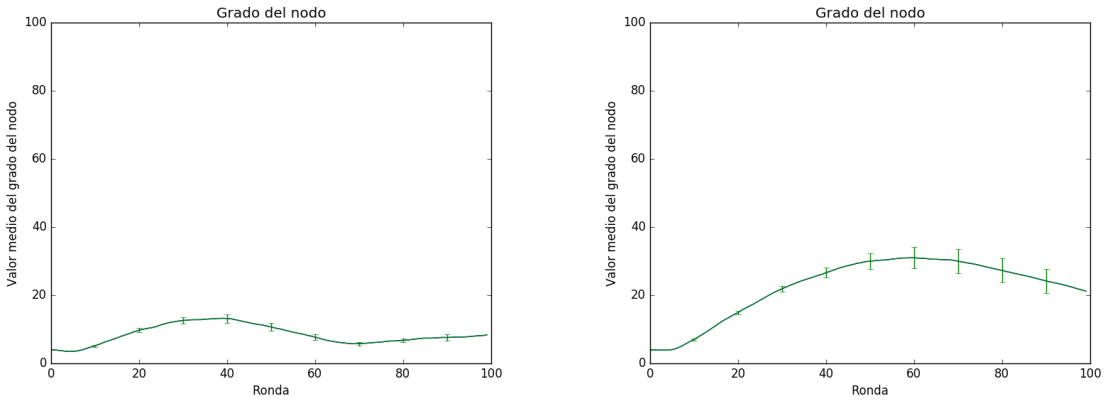
(b) Factor 0.95 y Tmin 2

Figura 38: Resultados modelo - Grado del nodo

En los resultados de los experimentos en las gráficas 35, el grado medio del nodo crece desde un 20 % hasta 60 %; mientras que en los experimentos del artículo [2] de la figura 36, se observa que el grado medio del nodo crece desde cerca del 25 % hasta un 65 %. Como los valores son práctimente iguales, los resultados pueden considerarse idénticos. En el modelo, se parte de 4 enlaces iniciales (20 % también) y aumenta hasta que llega a 12 ó 13, es decir, alcanzando una fracción de enlaces del 60 % ó 65 %, igual que en los experimentos.

Por ello, podemos concluir que, con estos parámetros, las redes del modelo se comportan de forma similar a las estudiadas en [1] y en [2].

En cambio, si ahora nos fijamos en las redes de **100 nodos**, tomando los mismos parámetros (T_{min}=2 y *factor*=0.90 ó 0.95) el ratio sí que mantiene el mismo comportamiento, pero el grado no se comporta igual.



(a) Factor 0.90 y Tmin 2

(b) Factor 0.95 y Tmin 2

Figura 39: Resultados modelo - Grado del nodo

Podemos observar que no se alcanzan los mismos valores, por lo que para redes de un tamaño mayor, es probable que el comportamiento no sea igual que para redes pequeñas, es decir, que el tamaño **sí** que afecte a la evolución de las mismas.

Si nos fijamos en los resultados del modelo (véase apéndice B) para las redes, tanto de 20 y 100 nodos, cuando las simulaciones se realizan durante 500 rondas, no existe variación sustancial en el resultado final entre ambas. Por ello, se podría que la evolución de las redes es rápida, es decir, converge rápidamente hasta la forma final.

Parámetros

Con respecto a los parámetros que puede tomar el sistema, se ha comentado en la sección anterior que los parámetros que hacen que el modelo se comporte de forma más parecida a los resultados de los experimentos eran *factor*=0.90 ó *factor*=0.95 y T_{min}=2, pero ahora se procede a hacer un análisis de sensibilidad del modelo.

Si nos fijamos en el apéndice B, podemos comprobar en la figura 40 que, en redes pequeñas, al aumentar el valor de *factor*, existe un cambio bastante sustancial de comportamiento de 0.7 a 0.9: los valores del **grado** aumentan y pasan a crecer rápidamente, estabilizándose en valores altos. También se puede observar un cambio cuando se pasa de valores de *factor* menores de 1 a valer 1 o más. Las redes pasan a converger a grafos completos de forma más rápida, y ahí se mantienen durante toda la simulación. Esto puede observarse también en redes de mayor tamaño (figura 46 del apéndice B): para *factor* de 0.7 a 0.95 existe un cambio muy sustancial de comportamiento; y cuando pasa a valer 1 o más, la convergencia de la red es a grafo completo.

Si pasamos a mirar ahora el **la cooperación media** de redes pequeñas (o el **ratio de colaboraciones**, ya que el comportamiento de las dos son iguales) en el apéndice B, en la figura 41, se puede observar que los valores donde cambia el comportamiento de las redes, son los mismos que en el grado del nodo. De igual forma que anteriormente, en redes de gran tamaño, los valores no varían.

Variando T_{min} en lugar de *factor*, no se observa que a partir de un valor del parámetro el comportamiento cambie bruscamente, sino que el cambio es gradual, por ejemplo en la figura 44 del anexo B; tanto para el grado del nodo, como para la colaboración media y el ratio de colaboraciones.

Como conclusión general, podemos decir que las reglas de comportamiento propuestas reproducen los resultados experimentales para valores de factor cercanos a (pero por debajo de) 1, mientras que el valor exacto de T_{min} no es tan importante. En este último parámetro, lo que vemos es que sí ha de haber un umbral para aceptar o mantener los enlaces, pero si hay una pequeña región intermedia donde la probabilidad no es 0 ó 1 no es excesivamente relevante. En términos de comportamiento, lo que nos dicen las reglas es que la cooperación es condicional, es decir, es mucho más probable cuanto mayor es la proporción de vecinos que cooperan, y que hay una pequeña probabilidad de no seguir esta regla. Finalmente, para los enlaces, está claro que la gente hace o conserva enlaces que tienen buena reputación, entendiendo por buena la media de la reputación de los vecinos o mayor.

Desarrollos futuros

Existen varios desarrollos posibles al finalizar este proyecto. Entre ellos, el más básico y principal será la implementación en el modelo de las situaciones con *reputación falseada*, que se trataban en el artículo [2].

Otro posible desarrollo futuro, es la automatización del modelo en un programa que compruebe los resultados que se obtendrían a partir del mismo, fijándole los parámetros de determinados experimentos ya realizados, comparándolos con los resultados de dichos experimentos. En función de las diferencias encontradas, el modelo podría ser modificado mediante técnicas de *machine learning* o aprendizaje automático para mejorarse de forma automática e iterativa. Esto sería fácilmente implementable mediante paquetes de Python del ámbito de la programación científica, como *pandas*, *PyBrain* o *scikit-learn*.

Todo ello se podría englobar en una aplicación con interfaz gráfica para realizar pruebas de forma mucho más cómoda y sencilla, sin necesidad de tener que abrir archivos de código y cambiar parámetros de forma manual, lo cual puede ser bastante engorroso para aquellos usuarios que no cuenten con los conocimientos necesarios. Sería posible con los paquetes *PyQt* o *Tinker* para la interfaz gráfica, y *py2exe* para crear un único paquete ejecutable, con todo lo necesario para que se pueda correr el modelo en diferentes sistemas operativos.

Apéndices

A. Código del programa

Se muestra a continuación el código creado para el modelo.

```
#====== IMPORTAR PAQUETES ======
import networkx as nx
import copy
import random
import math
import numpy as np
import matplotlib.pyplot as plt
import os
import ast
from matplotlib.pyplot import cm

#====== VARIABLES ======
factores=[0.7,0.9,0.95,1,1.05]
Tmines = [0,1,2,3]

N = 20   # N = numero de agentes (nodos) en la red
d = 4   # d = grado inicial de los nodos en la red
# Pagos
CC = 7
CD = 0
DC = 10
DD = 0
# Longitud programa
NUMrondas = 100
NUMprogram = 10


#====== FUNCIONES ======
def nuevaAccion(accion, nodo):
  global pastActions

  pastActions[nodo] = pastActions[nodo][1:5]+accion

def actualizaralfa(nodo):
  global alfa, pastActions

  d=pastActions[nodo]
  coop = 0
  for e in d:
    if e=='c':
      coop+=1
  alfa[nodo] = coop

def vecinosronda(ronda):
```

```
  global N, red, vecRonda, numvecmedio

  for i in range(N):
      vecRonda[ronda-1][i]=len(red.neighbors(i))
  numvecmedio[ronda-1] = np.mean(vecRonda[ronda-1])

def reputacionmedia(ronda):
  global alfa, repmedia

  repmedia[ronda-1]=np.mean(alfa)

def reputacionmediavecindario(nodo):
  global red, alfa, repmediavecindario
  vecs = red.neighbors(nodo)
  temp = 0
  temp2 = 0

  for yy in vecs:
      temp = temp + alfa[yy]

  if len(vecs)>0:
      temp2 = temp/len(vecs)
  repmediavecindario = temp2

  return temp2

def plotrepmedia(repmedia):
  x = np.arange(len(repmedia))
  plt.bar(x,repmedia, label='Alfa', width=0.5, color='#add8e6')
  plt.ylabel('Media de la reputacion')
  plt.ylim([0,5])
  plt.xlabel('Ronda')
  plt.title('Reputacion media en funcion de la ronda')
  plt.savefig(pathGraficas +'/Rep Media por ronda - iteracion %i.png' %
↪  (zz+1))
  plt.close()

def plotnumvecmedio(numvecmedio):
  x = np.arange(len(numvecmedio))
  plt.bar(x,numvecmedio,label='Media vecinos', width=0.5,
↪  color='#add8e6')
  plt.ylim([0,N])
  plt.ylabel('Media')
  plt.xlabel('Ronda')
  plt.title('Numero medio de vecinos en funcion de la ronda')
  plt.savefig(pathGraficas +'/Media de vecinos por ronda - iteracion
↪  %i.png' % (zz+1))
```

```
  plt.close()

def plothistAlfa():
  global alfa, N
  x = np.arange(6)
  plt.hist(alfa, x, histtype='bar', color='#add8e6')
  plt.ylabel('Cantidad de nodos')
  plt.xlabel('Reputacion')
  plt.title('Histograma de la reputacion')
  plt.savefig(pathGraficas +'/Histograma reputacion - iteracion %i.png' %
↪  (zz+1))
  plt.close()

def plothistVec():
  global red, N
  hist = [0 for x in range(N)]
  temp = [0 for x in range(N)]
  for i in range(N):
    temp[i] = len(red.neighbors(i))
  for j in range(N):
    for k in range(N):
      if j==temp[k]:
        hist[j] = hist[j] + 1
  plt.bar(np.arange(len(hist)),hist,width=0.5,color='#add8e6')
  plt.ylabel('Cantidad de nodos')
  plt.xlabel('Numero vecinos')
  plt.title('Histograma de vecinos')
  plt.savefig(pathGraficas +'/Histograma vecinos - iteracion %i.png' %
↪  (zz+1))
  plt.close()

def calcularpagos():
  global red, colabSig, pagos

  for q in range(N):
    vecinosAux = red.neighbors(q)
    for r in vecinosAux:
      if((colabSig[q]==True) & (colabSig[r]==True)):
        pagos[q] = pagos[q] + CC
      if((colabSig[q]==True) & (colabSig[r]==False)):
        pagos[q] = pagos[q] + CD
      if((colabSig[q]==False) & (colabSig[r]==True)):
        pagos[q] = pagos[q] + DC
      if((colabSig[q]==False) & (colabSig[r]==False)):
        pagos[q] = pagos[q] + DD

def probAceptar (nodo):
```

```
  global alfa, Tmin
  delta = reputacionmediavecindario(i)
  if (alfa[nodo] <= Tmin):
    prob = 0
  elif delta <= alfa[nodo]:
    prob = 1
  else:
    prob = (alfa[nodo]-Tmin)/(delta-Tmin)

  return prob

def probCortar(nodo):
  probAux = probAceptar(nodo)
  prob = 1 - probAux

  return prob

def agente():
  # traigo las variables globales
  global alfa, colabSig, colabAux, pastActions, red, redSig,
↪  repmediavecindario
  ronda = 0
  ####################
  while(ronda < NUMrondas):
    ronda = ronda + 1
    for i in range(N):   # voy cada nodo
        vecinos=red.neighbors(i)   # miro que vecinos tiene
        colabAux[i]=0   # pongo el # de colaboraciones red 0, para poder
↪  contarlas
        for k in vecinos:
          if pastActions[k][4]=='c':
            colabAux[i]+=1
        if len(vecinos)!=0:
            if random.random()<=factor*(reputacionmediavecindario(i)/5):
                colabSig[i]=True
            else:
                colabSig[i]=False
        else:
          colabSig[i]=True   # si estoy solo, colaboro siempre

        minimoalfa = 5
        # recorro todos los vecinos mirando cual tiene menor alfa
        for m in vecinos:
          if alfa[m] < minimoalfa:
            # almaceno el valor del alfa mas pequeno y de que vecino es
            minimoalfa = alfa[m]
            minimo = m
```

```
        elif ronda==1:
          minimoalfa = alfa[vecinos[0]]
          minimo = vecinos[0]
      # elimino el enlace con el que tenga menos alfa
      for p in red.edges():   # if ((p==(i,minimo)) and (minimoalfa <
↪ Tmin)):
        if ((p==(i,minimo)) and (random.random()<=probCortar(i))):
          redSig.remove_edge(i,minimo)
      # propongo un enlace aleatoriamente
      dentro = True
      c = 0
      while(dentro==True):
        aleat1=random.randint(0,N-1)   # enlace a alguien aleatorio
        if ((aleat1!=i) and (not (aleat1 in vecinos))):   # no enlace
↪ el mismo, no haya enlazado ya
          if random.random() <= probAceptar(i):   # que yo quiera
↪ enlazar con el otro
            if random.random() <= probAceptar(aleat1):   # que el otro
↪ quiera enlazar conmigo
              redSig.add_edge(i,aleat1)   # creo el enlace
              dentro = False   # salgo del bucle
            else:
              dentro = False   # si no cumplo, salgo del bucle sin
↪ crearlo
          else:
            dentro = False   # si no cumple, salgo del bucle sin
↪ crearlo
        else:
          c+=1
        if c==20:
          break
      # saco la reputacion media de la ronda
      reputacionmedia(ronda)
      # actualizo el numero de vecinos que tiene cada nodo en la ronda
      vecinosronda(ronda)

  # Recalculo alfa y actualizo el pastActions
  for i in range(N):
    if colabSig[i]==True:
      nuevaAccion('c',i)
    else:
      nuevaAccion('d',i)
    actualizaralfa(i)
    # cuantos han colaborado
    if pastActions[i][4]=='c':
      cooperadores[ronda-1]+=1
```

```
      # actualizo la red con los nuevos enlaces
      red = redSig.copy()
      # calculo los pagos con la nueva red
      calcularpagos()
    file.write(str(numvecmedio)+'\n')
    file.write('fin')
    file2.write(str(repmedia)+'\n')
    file2.write('fin')

    for i in range(NUMrondas):
      cooperadores[i]=cooperadores[i]/N

    file3.write(str(cooperadores)+'\n')
    file3.write('fin')

def redppal():
    agente()
    d = nx.degree(red)
    node_color=[float(redSig.degree(v)) for v in red]
    nx.draw(red, nodelist=d.keys(), node_color=node_color)
    plt.savefig(pathGraficas +'/red - iteracion %i.png' % (zz+1))
    plt.close()

for Tmin in Tmines:
    for factor in factores:
      #==== Rutas a las carpetas ====
      #pathCarpetas
      if(Tmin==0) and (factor==0.7):
          os.makedirs('%i' %N + ' nodos y %i' %NUMrondas + ' rondas')
      pathCarpetas='%i' %N + ' nodos y %i' %NUMrondas + ' rondas'

      #pathPrograma
      os.makedirs(pathCarpetas + '/Factor %.2f' %factor + ' y Tmin %i'
↪   %Tmin)
      pathPrograma=pathCarpetas + '/Factor %.2f' %factor + ' y Tmin %i'
↪   %Tmin

      #pathGraficas
      os.makedirs(pathPrograma + '/Graficas de %i' %N +' nodos y %i'
↪   %NUMrondas + ' rondas')
      pathGraficas=pathPrograma + '/Graficas de %i' %N +' nodos y %i'
↪   %NUMrondas + ' rondas'

      #pathArchivos
      os.makedirs(pathPrograma + '/Archivos de %i' %N +' nodos y %i'
↪   %NUMrondas + ' rondas')
```

```
    pathArchivos=pathPrograma + '/Archivos de %i' %N +' nodos y %i'
↪   %NUMrondas + ' rondas'

    #pathJuntar
    if(factor==0.7):
      os.makedirs(pathCarpetas + '/Juntar con Tmin %i' %Tmin)
    pathJuntar=pathCarpetas + '/Juntar con Tmin %i' %Tmin


    #====== ITERACION DEL PROGAMA ======

    for zz in range(NUMprogram):   # repeticion del programa
      #====== VARIABLES PARA GESTIONAR LA RED ======
      file = open(pathArchivos + "/archivo_red %i tfg.txt" % (zz+1),'w')
      file2 = open(pathArchivos + "/archivo_red %i tfgC.txt" %
↪   (zz+1),'w')
      file3 = open(pathArchivos + "/archivo_red %i tfgC_ult.txt" %
↪   (zz+1),'w')
      red = nx.random_regular_graph(d, N)   # crea un grafo aleatorio de
↪   N nodos con grado d
      redSig = red.copy()

      # creo y expando las listas hasta los N agentes, con los valores
↪   por defecto
      alfa = [0 for x in range(N)]   # alfa de cada integrante de la red
      colabSig = [False for x in range(N)]   # indica si el agente
↪   colaborara en la ronda o no
      colabAux = [0 for x in range(N)]   # indica cuantos vecinos de cada
↪   agente colaboran
      pastActions = ['aaaaa' for x in range(N)]   # colaboraciones y
↪   defects en las ultimas 5 rondas
      repmedia = [0 for x in range(NUMrondas)]   # reputacion media por
↪   cada ronda
      repmediavecindario = 0   # el alfa medio del vecindario de un nodo
      vecRonda = [[4 for x in range(N)] for x in range(NUMrondas)]    #
↪   vecinos de cada nodo por ronda
      numvecmedio = [4 for x in range(NUMrondas)]   # numero medio de
↪   vecinos en cada ronda
      pagos = [0 for x in range(N)]   # cuanto lleva ganado cada jugador
      cooperadores = [0 for x in range(NUMrondas)]   # cuantos han
↪   colaborado en la ronda

      for i in range (N):   # a cada nodo le asigno un historial
↪   aleatorio de acciones
        for j in range(5):
          if random.random()>0.5:
            pastActions[i] = pastActions[i][1:5]+'c'
```

```
        else:
          pastActions[i] = pastActions[i][1:5]+'d'

    for i in range(N):
      if pastActions[i][4]=='c':
        colabSig[i] = True
      else:
        colabSig[i] = False
    for i in range(N):
      actualizaralfa(i)
    # llamo  las funciones
    file.write("Iteracion %i\n" % (zz+1))
    file2.write("Iteracion %i\n" %(zz+1))
    redppal()
    plotrepmedia(repmedia)
    plotnumvecmedio(numvecmedio)
    plothistAlfa()
    plothistVec()
    file.close()
    file2.close()

      #====== FIN DEL PROGRAMA ======
  # Graficamos la evolucion de la red con los archivos de la parte
↪ anterior del programa

  aux0 = [0 for x in range(NUMrondas)]
  aux1 = [0 for x in range(NUMrondas)]
  aux2 = [0 for x in range(NUMrondas)]
  aux3 = [0 for x in range(NUMrondas)]

  std_error = [0 for x in range(NUMrondas)]
  std_errorC = [0 for x in range(NUMrondas)]

  coop00 = [0 for x in range(NUMrondas)]

  ########################
  ########################

  # saco de todos los archivos los datos, y creo uno que los almacene
  graphicfile=open(pathArchivos + '/graficar.txt', mode='w')
  for i in range(zz):
    filename = (pathArchivos + '/archivo_red %i tfg.txt' %(i+1))
    with open(filename) as input_data:
      # Se salta todo hasta el bloque que interesa:
      for line in input_data:
        if line.strip() == 'Iteracion %i'%(i+1):  # O la condicion que
↪ se requiera
```

```
        break
    # Lee hasta el siguiente bloque:
    for line in input_data:  # Sigue leyendo el resto del archivo
      if line.strip() == 'fin':
        break
      line = ast.literal_eval(line)
      graphicfile.write(str(line)+'\n')
graphicfile.close()


graphicfile=open(pathArchivos + '/graficar.txt', mode='r')
for line in graphicfile:
  line = line.strip()
  line = ast.literal_eval(line)
  for i in range(NUMrondas):
    aux0[i]=line[i]+aux0[i]
graphicfile.close()

for i in range(NUMrondas):
  aux0[i]=round(aux0[i],4)

aux1=aux0.copy()
for i in range(NUMrondas):
  aux1[i]=aux1[i]/zz
for i in range(NUMrondas):
  aux1[i]=round(aux1[i],4)

with open(pathJuntar + "/juntar.txt", "a") as myfile:
  myfile.write(str(aux1)+"\n")

############################
graphicfile=open(pathArchivos + '/graficar.txt', mode='r')
for line in graphicfile:
  line = line.strip()
  line = ast.literal_eval(line)
  for i in range(NUMrondas):
    aux2[i]=(line[i]-aux1[i])**2+aux2[i]
graphicfile.close()

for i in range(NUMrondas):
  aux2[i]=round(aux2[i],4)

aux3=aux2.copy()
for i in range(NUMrondas):
  aux3[i]=aux3[i]/zz
for i in range(NUMrondas):
  aux3[i]=round(aux3[i],4)
```

```
    sigmak = [0 for x in range(NUMrondas)]

    for i in range(NUMrondas):
      sigmak[i]=math.sqrt(aux3[i])    # varianza = sigmak

    for i in range(NUMrondas):
      sigmak[i]=round(sigmak[i],4)

    ##############
    # Standard error
    for i in range(NUMrondas):
      std_error[i]=sigmak[i]/math.sqrt(zz)

   ####################################
    # Graficar
    x=np.arange(NUMrondas)

    plt.plot(x, aux1)
    plt.errorbar(x, aux1, yerr=std_error, errorevery=10)
    plt.ylim([0,N])
    plt.ylabel('Valor medio del grado del nodo')
    plt.xlabel('Ronda')
    plt.title('Grado del nodo')
    plt.savefig(pathPrograma + '/Grado del nodo.png')
    plt.close()
    ########################
    ########################
    aux00 = [0 for x in range(NUMrondas)]
    aux11 = [0 for x in range(NUMrondas)]
    aux22 = [0 for x in range(NUMrondas)]
    aux33 = [0 for x in range(NUMrondas)]

    # saco de todos los archivos los datos, y creo uno que los almacene
    graphicfile=open(pathArchivos + '/graficarC.txt', mode='w')
    for i in range(zz):
      filename = (pathArchivos + '/archivo_red %i tfgC.txt' %(i+1))
      with open(filename) as input_data:
        # Skips text before the beginning of the interesting block:
        for line in input_data:
          if line.strip() == 'Iteracion %i'%(i+1):  # Or whatever test is
↪  needed
            break
        # Reads text until the end of the block:
        for line in input_data:  # This keeps reading the file
          if line.strip() == 'fin':
            break
```

```
    line = ast.literal_eval(line)
    graphicfile.write(str(line)+'\n')
graphicfile.close()


graphicfile=open(pathArchivos + '/graficarC.txt', mode='r')
for line in graphicfile:
  line = line.strip()
  line = ast.literal_eval(line)
  for i in range(NUMrondas):
    aux00[i]=line[i]+aux00[i]
graphicfile.close()

for i in range(NUMrondas):
  aux00[i]=round(aux00[i],4)

aux11=aux00.copy()
for i in range(NUMrondas):
  aux11[i]=aux11[i]/zz
for i in range(NUMrondas):
  aux11[i]=round(aux11[i],4)

with open(pathJuntar + "/juntarC.txt", "a") as myfile:
  myfile.write(str(aux11)+"\n")

######################################################################
graphicfile=open(pathArchivos + '/graficarC.txt', mode='r')
for line in graphicfile:
  line = line.strip()
  line = ast.literal_eval(line)
  for i in range(NUMrondas):
    aux22[i]=(line[i]-aux11[i])**2+aux22[i]
graphicfile.close()

for i in range(NUMrondas):
  aux22[i]=round(aux22[i],4)

aux33=aux22.copy()
for i in range(NUMrondas):
  aux33[i]=aux33[i]/zz
for i in range(NUMrondas):
  aux33[i]=round(aux33[i],4)

sigmac = [0 for x in range(NUMrondas)]

for i in range(NUMrondas):
  sigmac[i]=math.sqrt(aux33[i])
```

```
for i in range(NUMrondas):
  sigmac[i]=round(sigmac[i],4)

##############################
#Standard error
for i in range(NUMrondas):
  std_errorC[i] = sigmac[i]/math.sqrt(zz)

##############################
# Graficar
xx=np.arange(NUMrondas)

plt.plot(x, aux11)
plt.errorbar(x, aux11, yerr=std_errorC, errorevery=10)
plt.ylim([0,5])
plt.ylabel('Valor medio de colaboraciones')
plt.xlabel('Ronda')
plt.title('Numero medio de colaboraciones')
plt.savefig(pathPrograma + '/Numero medio de colaboraciones.png')
plt.close()
#######################
#######################
graphicfile=open(pathArchivos + '/graficarC_ult.txt', mode='w')
for i in range(zz):
  filename = (pathArchivos + '/archivo_red %i tfgC_ult.txt' %(i+1))
  with open(filename) as input_data:
    # Reads text until the end of the block:
    for line in input_data:  # This keeps reading the file
      if line.strip() == 'fin':
        break
      line = ast.literal_eval(line)
      graphicfile.write(str(line)+'\n')
graphicfile.close()

#print('ratio:')
graphicfile=open(pathArchivos + '/graficarC_ult.txt', mode='r')
for line in graphicfile:
  line = line.strip()
  line = ast.literal_eval(line)
  for i in range(NUMrondas):
    coop00[i]=line[i]+coop00[i]
for i in range(NUMrondas):
  coop00[i]=coop00[i]/zz
  coop00[i]=round(coop00[i],4)

graphicfile.close()
```

```
    with open(pathJuntar + "/juntarC_ult.txt", "a") as myfile:
      myfile.write(str(coop00)+"\n")

    ##############################################
    # Graficar
    x=np.arange(NUMrondas)

    plt.plot(x,coop00)
    plt.ylim([0,1])
    plt.ylabel('Ratio')
    plt.xlabel('Ronda')
    plt.title('Ratio "cooperadores / agentes totales"')
    plt.savefig(pathPrograma + '/Ratio.png')
    plt.close()
    # Al llegar al final de los valores de Tmin o de Factores,
↪   promediamos todas las iteraciones
    if((factor==1.05) or (Tmin==3)):
      if(factor==1.05):
        NUMfactor = len(factores)
      elif(Tmin==3):
        NUMfactor = len(Tmines)

      vector_juntar = [[0 for x in range(NUMrondas)] for x in
↪   range(NUMfactor)]
      j=0

      graphicfile=open(pathJuntar + '/juntar.txt', mode='r')
      for line in graphicfile:
        line = line.strip()
        line = ast.literal_eval(line)
        vector_juntar[j] = line
        j+=1
      graphicfile.close()

      x = np.arange(NUMrondas)

      color=iter(cm.rainbow(np.linspace(0,1,5)))
      j=0

      for j in range(NUMfactor):
        label="Factor: " + str(factores[j])
        c=next(color)
        plt.plot(x,vector_juntar[j], label=label, color=c)
        plt.ylim([0,N])
        plt.ylabel('Grado del nodo')
        plt.xlabel('Ronda')
```

```
        plt.title('Grado del nodo en funcion de factor')
        plt.legend()
        plt.savefig(pathCarpetas+ '/Grado en fn de factor para Tmin %i'
↪  %Tmin + '.png')
      #plt.show()
      plt.close()

      ########################
      ########################
      ########################
      ########################

      vector_juntar = [[0 for x in range(NUMrondas)] for x in
↪  range(NUMfactor)]
      j=0

      graphicfile=open(pathJuntar + '/juntarC.txt', mode='r')
      for line in graphicfile:
        line = line.strip()
        line = ast.literal_eval(line)
        vector_juntar[j] = line
        j+=1
      graphicfile.close()

      x = np.arange(NUMrondas)

      color=iter(cm.rainbow(np.linspace(0,1,5)))
      j=0

      for j in range(NUMfactor):
        label="Factor: " + str(factores[j])
        c=next(color)
        plt.plot(x,vector_juntar[j], label=label, color=c)
        plt.ylim([0,5])
        plt.ylabel('Colaboracion media')
        plt.xlabel('Ronda')
        plt.title('Colaboracion media en funcion de factor')
        plt.legend()
        plt.savefig(pathCarpetas + '/Colab media en fn de factor para
↪  Tmin %i' %Tmin + '.png')
      plt.close()

      ########################
      ########################

      vector_juntar = [[0 for x in range(NUMrondas)] for x in
↪  range(NUMfactor)]
```

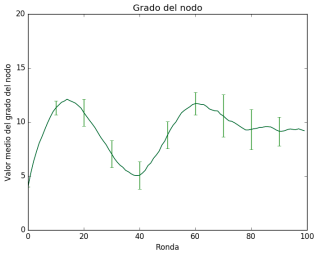
```
    j=0

    graphicfile=open(pathJuntar + '/juntarC_ult.txt', mode='r')
    for line in graphicfile:
      line = line.strip()
      line = ast.literal_eval(line)
      vector_juntar[j] = line
      j+=1
    graphicfile.close()

    x = np.arange(NUMrondas)

    color=iter(cm.rainbow(np.linspace(0,1,5)))
    j=0

    for j in range(NUMfactor):
      label="Factor: " + str(factores[j])
      c=next(color)
      plt.plot(x,vector_juntar[j], label=label, color=c)
      plt.ylim([0,1])
      plt.ylabel('Ratio ultima colaboracion')
      plt.xlabel('Ronda')
      plt.title('Ultima colaboracion en funcion de factor')
      plt.legend()
      plt.savefig(pathCarpetas + '/Ultima colab en fn de factor para
↪  Tmin %i' %Tmin + '.png')
```

B. Resultados de las simulaciones

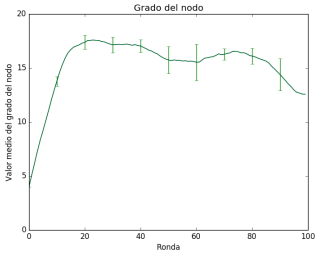
20 nodos y 100 rondas.

Variando *factor*, se obtienen las siguiente gráficas.

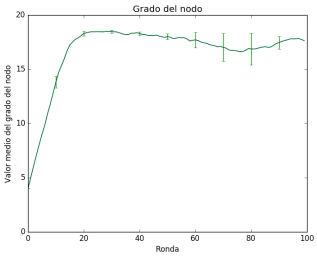
Grado medio del nodo, variando factor.



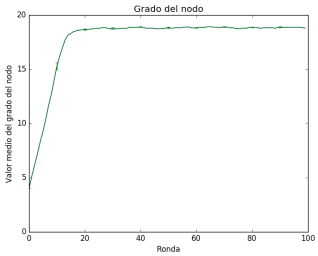
(a) Factor 0.7



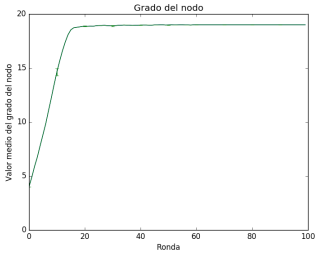
(b) Factor 0.9



(c) Factor 0.95



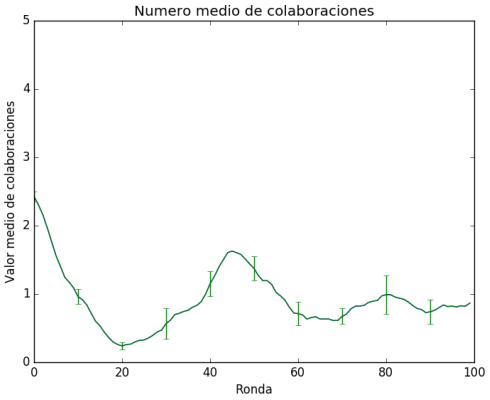
(d) Factor 1



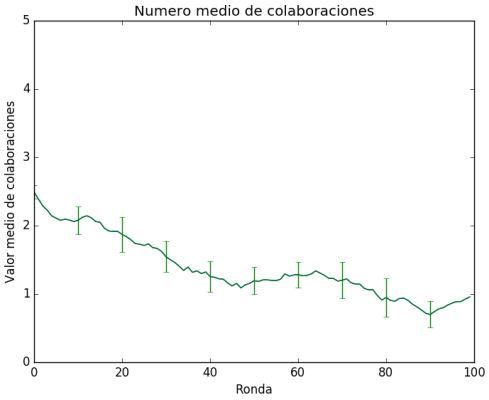
(e) Factor 1.05

Figura 40: Evolución del grado del nodo en función de factor, con T_{min} fijo en 0

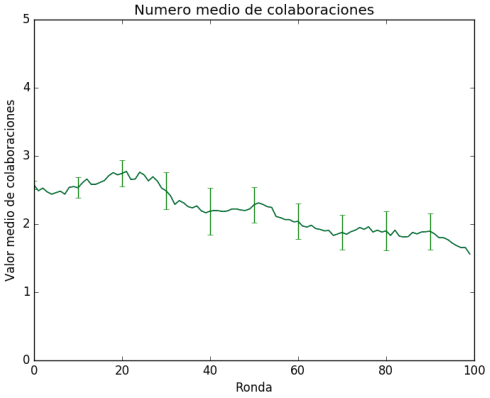
Número medio de colaboraciones.



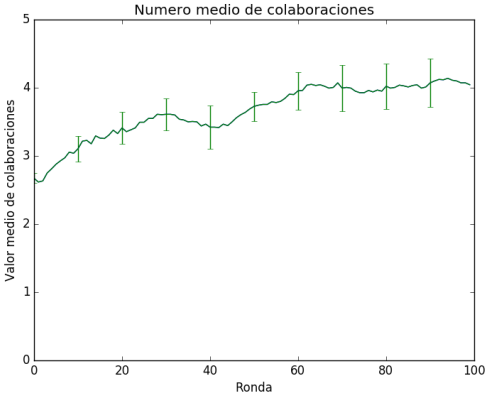
(a) Factor 0.7



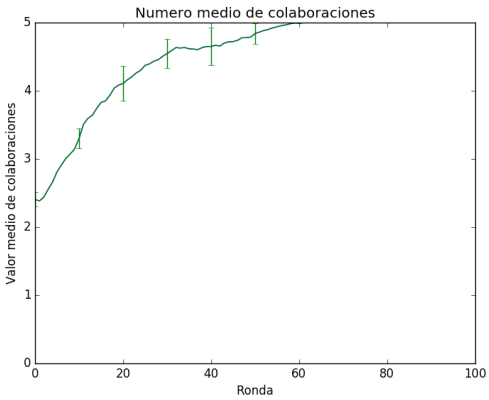
(b) Factor 0.9



(c) Factor 0.95



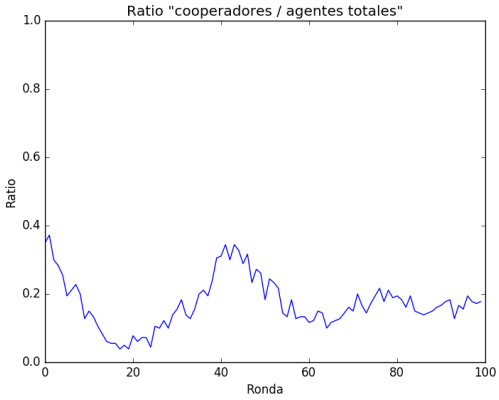
(d) Factor 1



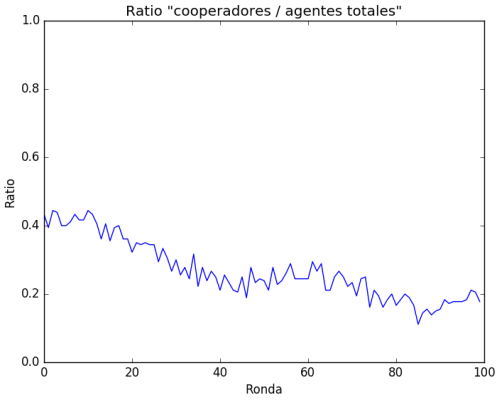
(e) Factor 1.05

Figura 41: Evolución del número medio de colaboraciones función de factor, con T_{min} fijo en 0

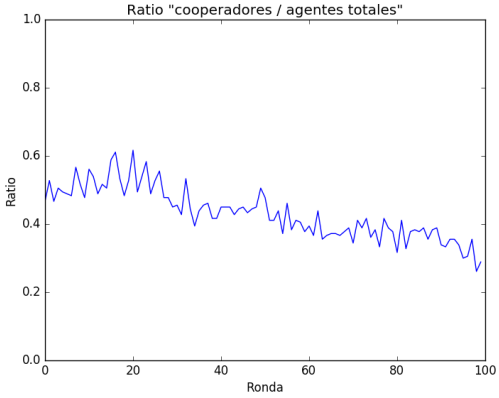
Ratio de colaboraciones.



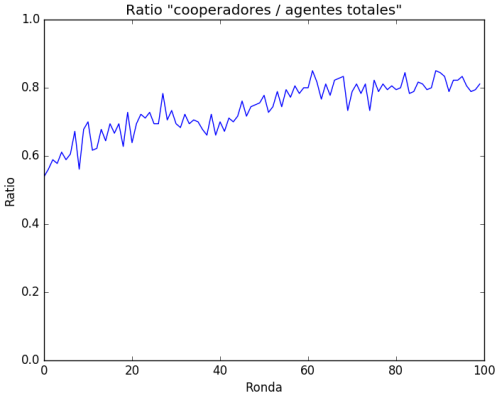
(a) Factor 0.7



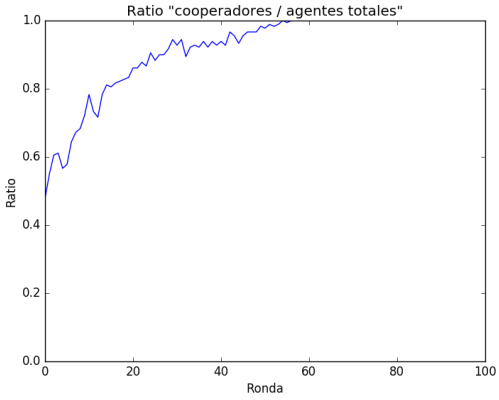
(b) Factor 0.9



(c) Factor 0.95



(d) Factor 1

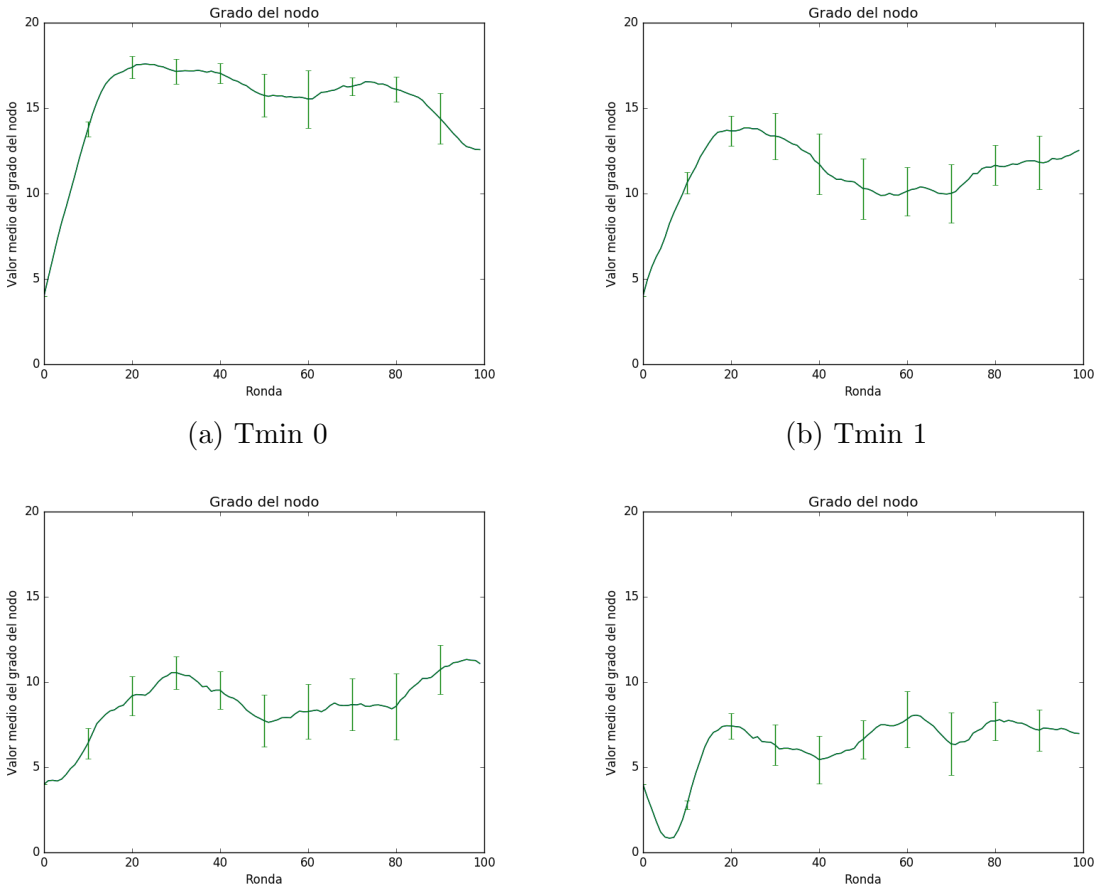


(e) Factor 1.05

Figura 42: Evolución del ratio de colaboraciones función de factor, con T_{min} fijo en 0

Variando *Tmin*, se obtienen las siguiente gráficas.

Evolución del grado medio del nodo, con *factor* 0.9.



(a) Tmin 0

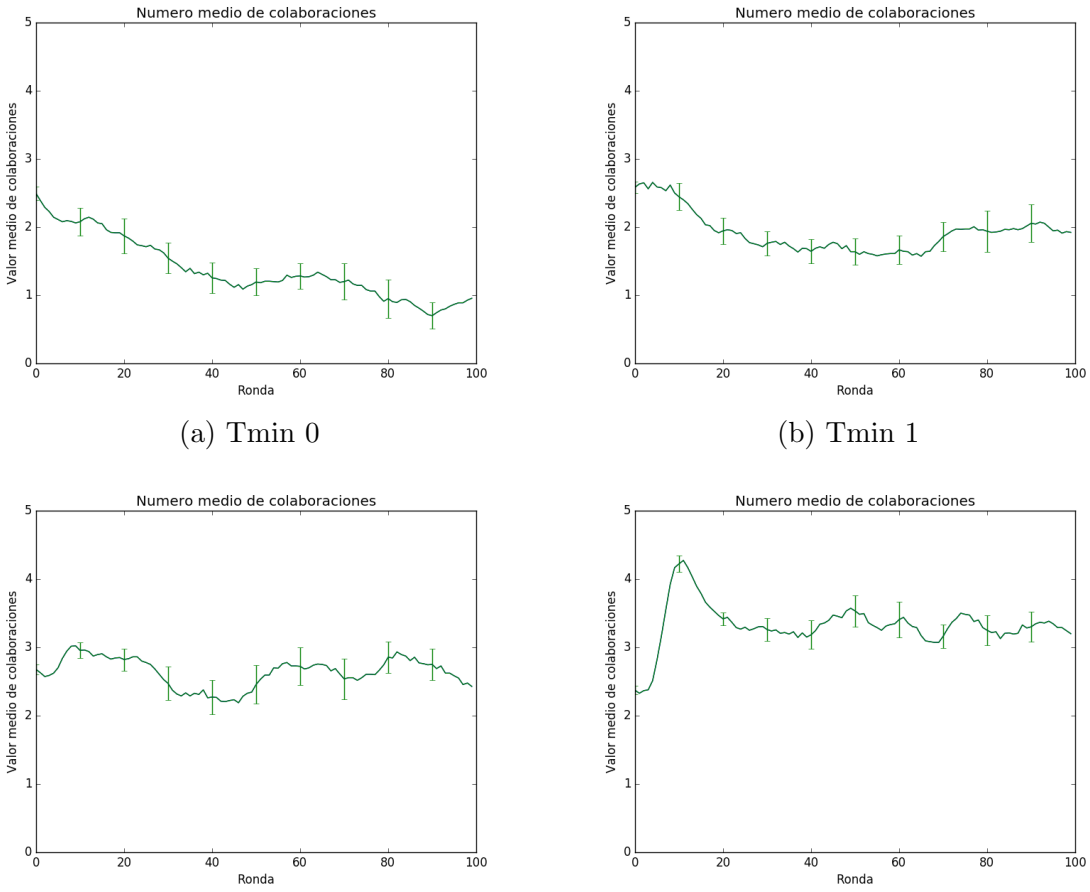
(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 43: Evolución del grado del nodo en función de Tmin, con factor fijo en 0.9

Evolución de la colaboración media, con *factor* 0.9.



(a) Tmin 0

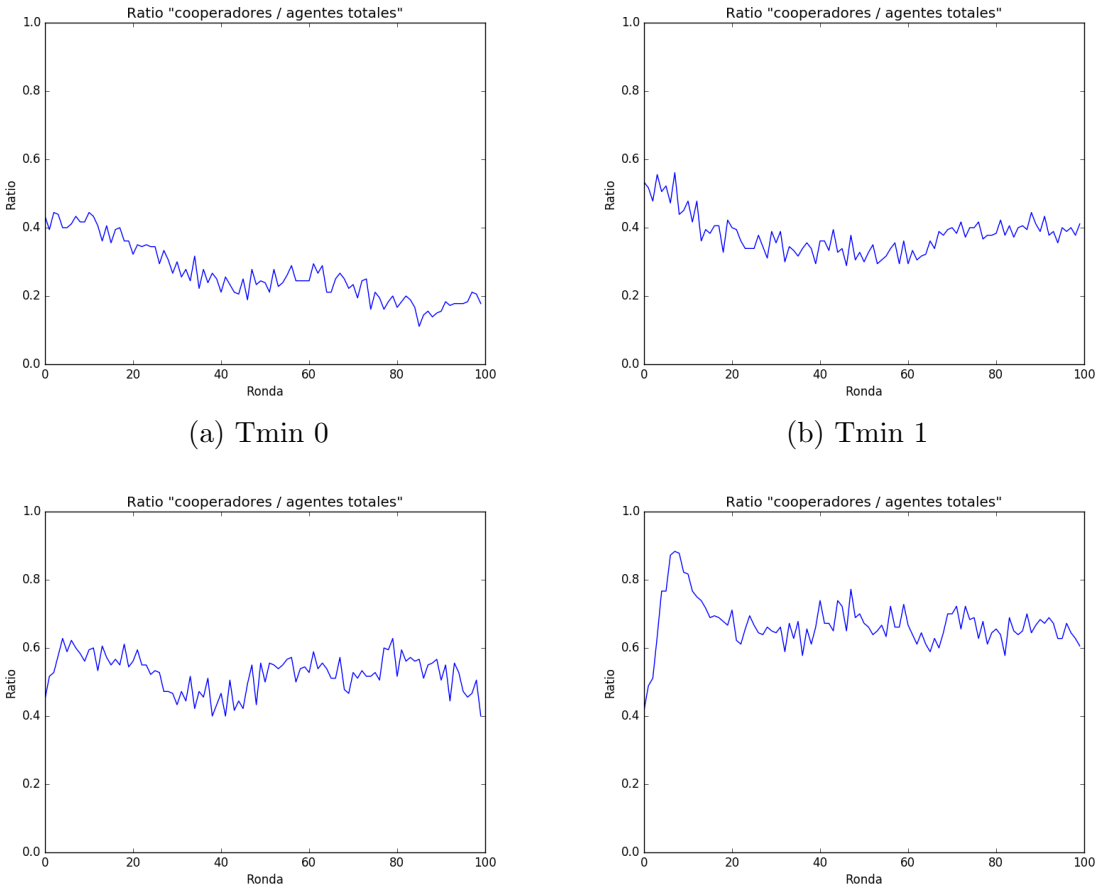
(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 44: Evolución del numero medio colaboraciones función de Tmin, con factor fijo en 0.9

Evolución del ratio de colaboraciones, con *factor* 0.9.



(a) Tmin 0

(b) Tmin 1

(c) Tmin 2

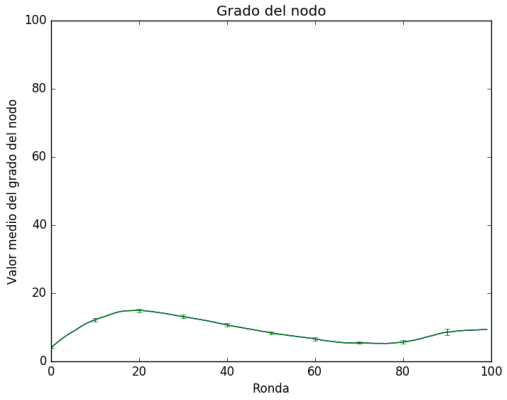
(d) Tmin 3

Figura 45: Evolución del ratio de colaboraciones en función de Tmin, con factor fijo en 0.9

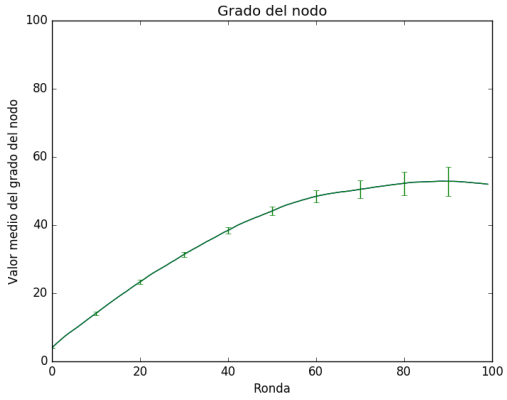
100 nodos y 100 rondas.

Variando *factor*, se obtienen las siguiente gráficas.

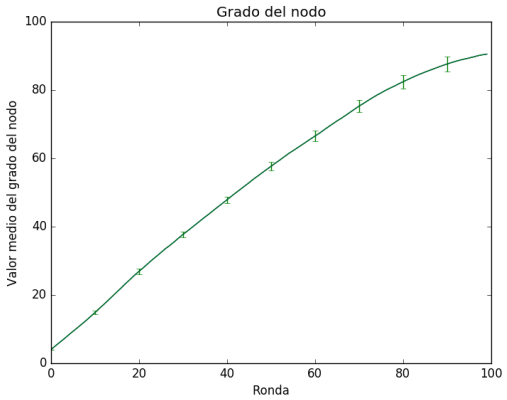
Grado medio del nodo.



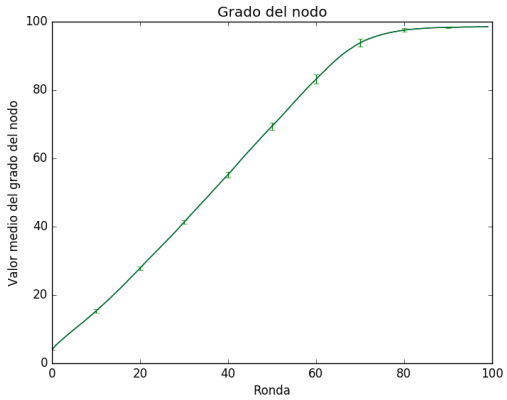
(a) Factor 0.7



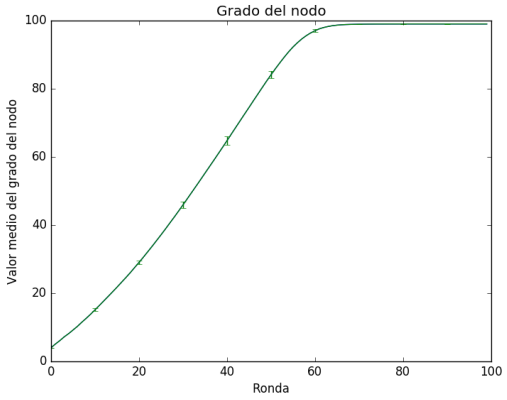
(b) Factor 0.9



(c) Factor 0.95



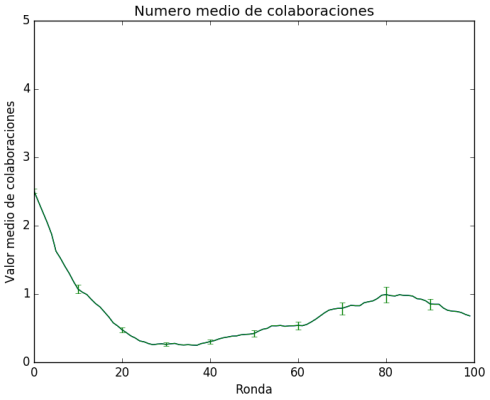
(d) Factor 1



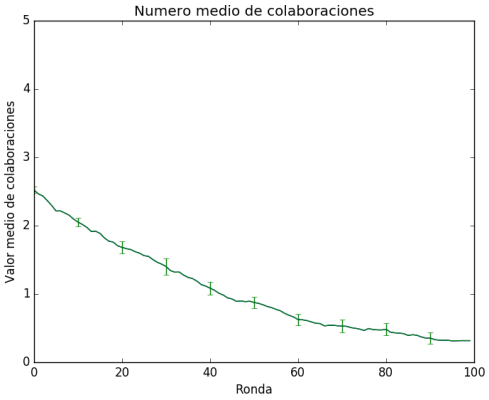
(e) Factor 1.05

Figura 46: Evolución del grado del nodo en función de factor, con T_{min} fijo en 0

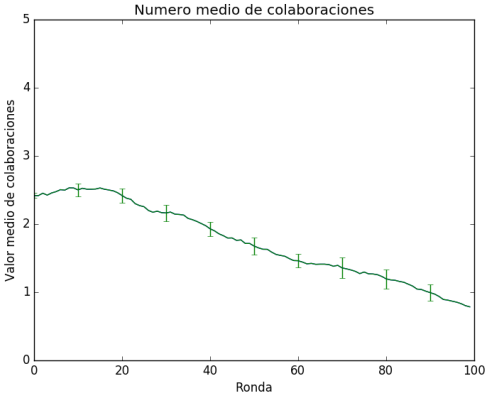
Número medio de colaboraciones.



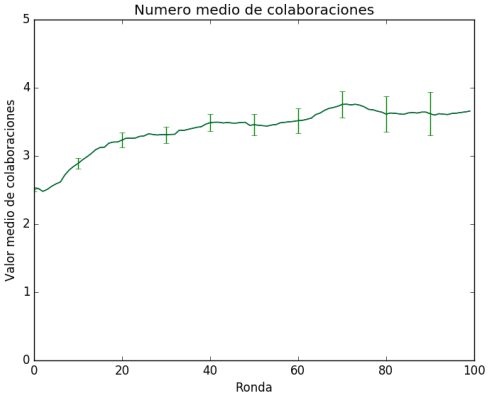
(a) Factor 0.7



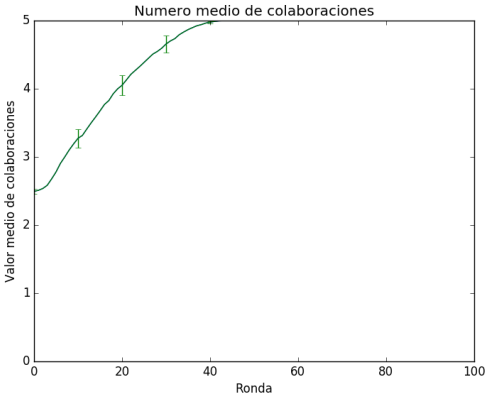
(b) Factor 0.9



(c) Factor 0.95



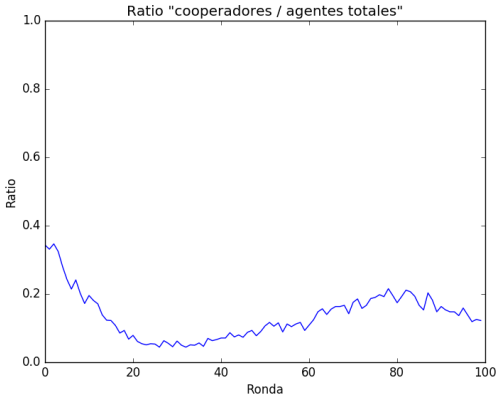
(d) Factor 1



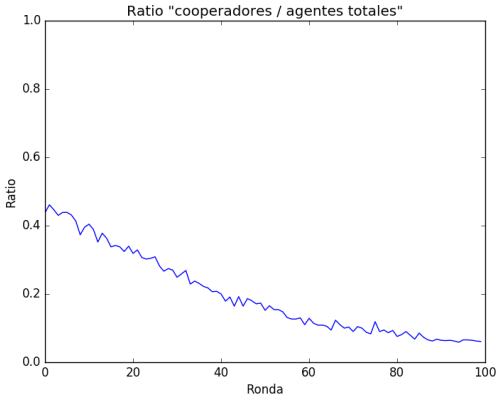
(e) Factor 1.05

Figura 47: Evolución del número medio de colaboraciones función de factor, con T_{min} fijo en 0

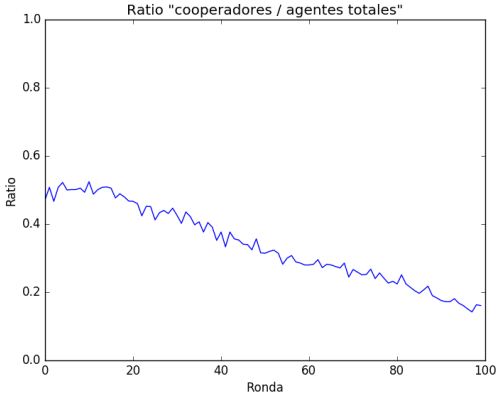
Ratio de colaboraciones.



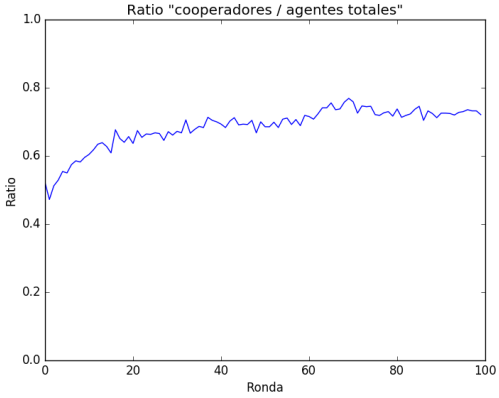
(a) Factor 0.7



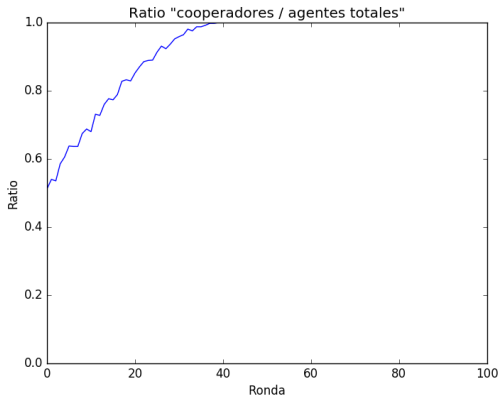
(b) Factor 0.9



(c) Factor 0.95



(d) Factor 1

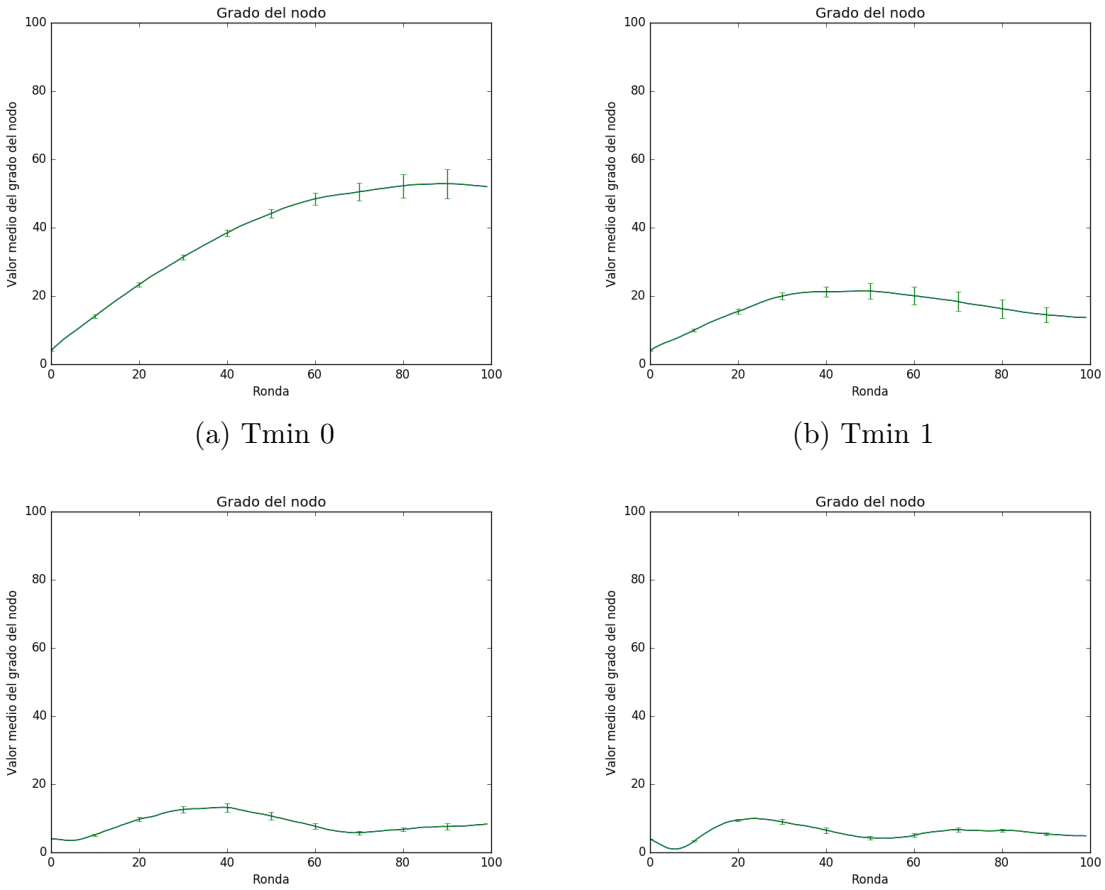


(e) Factor 1.05

Figura 48: Evolución del ratio de colaboraciones función de factor, con T_{min} fijo en 0

Variando *Tmin*, se obtienen las siguiente gráficas.

Evolución del grado medio del nodo, con *factor 0.9*.



(a) Tmin 0

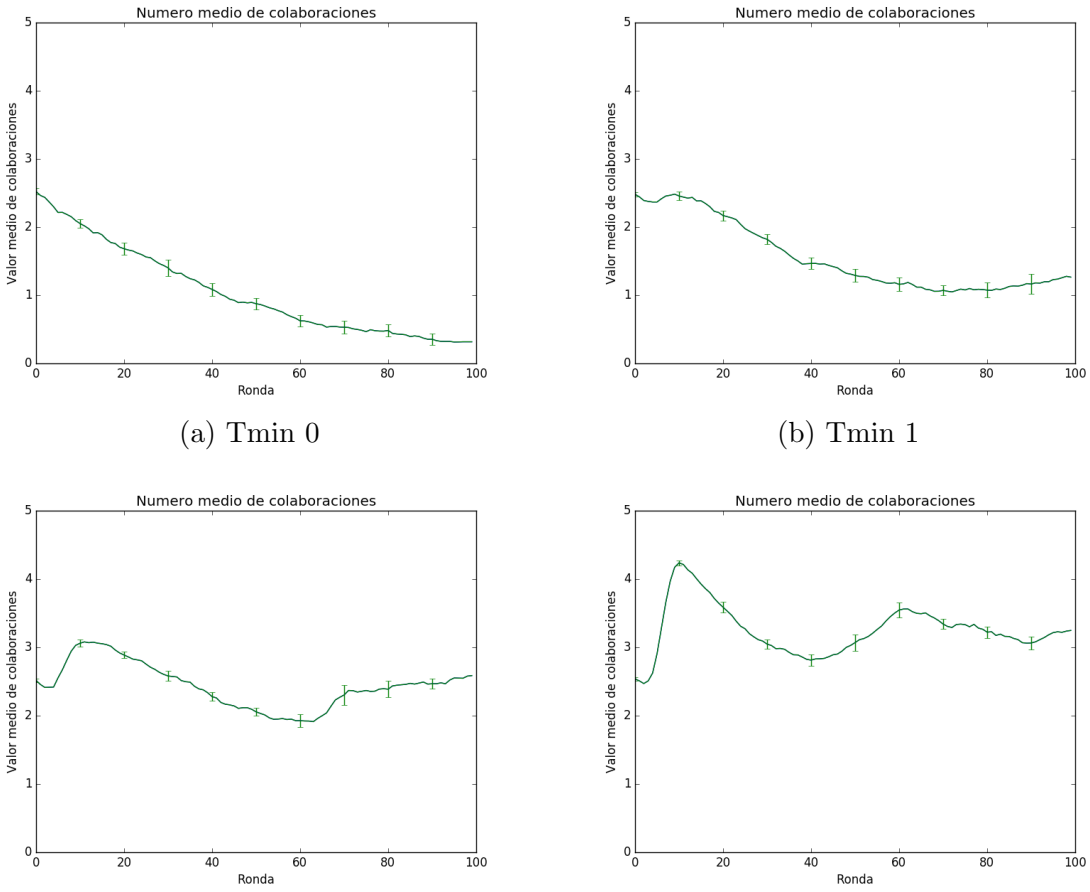
(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 49: Evolución del grado del nodo en función de Tmin, con factor fijo en 0.9

Evolución de la colaboración media, con *factor 0.9*.



(a) Tmin 0

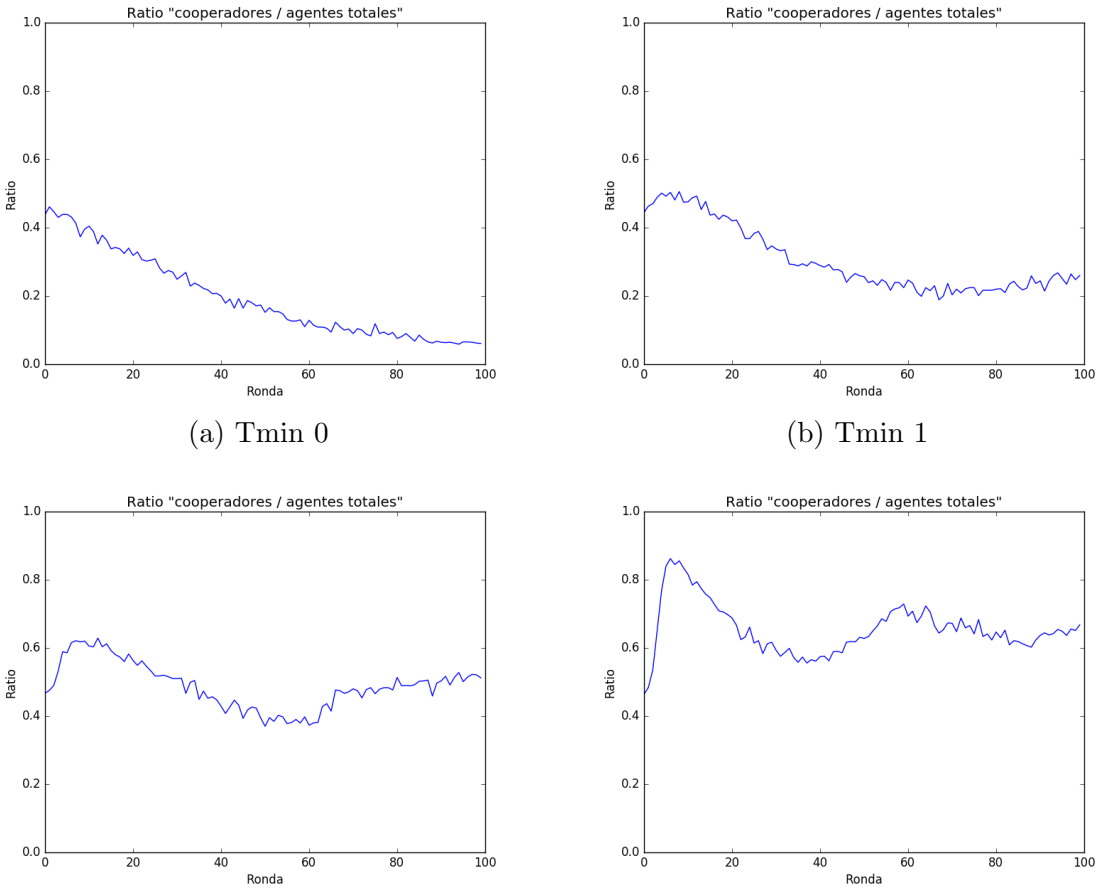
(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 50: Evolución del numero medio colaboraciones función de Tmin, con factor fijo en 0.9

Evolución del ratio de colaboraciones, con *factor* 0.9.



(a) Tmin 0

(b) Tmin 1

(c) Tmin 2

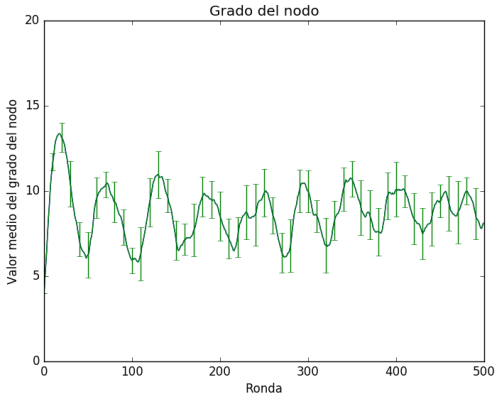
(d) Tmin 3

Figura 51: Evolución del ratio de colaboraciones en función de Tmin, con factor fijo en 0.9

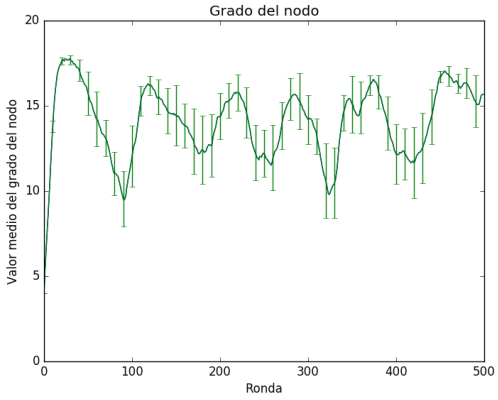
20 nodos y 500 rondas.

Variando *factor*, se obtienen las siguiente gráficas.

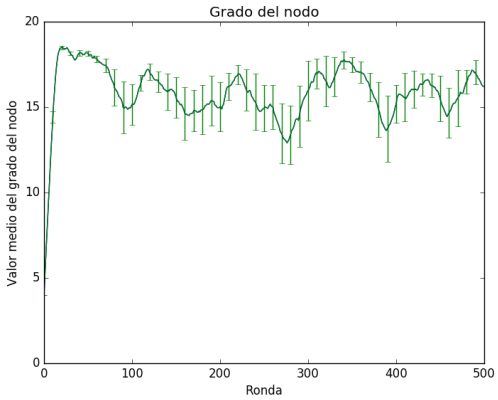
Grado medio del nodo.



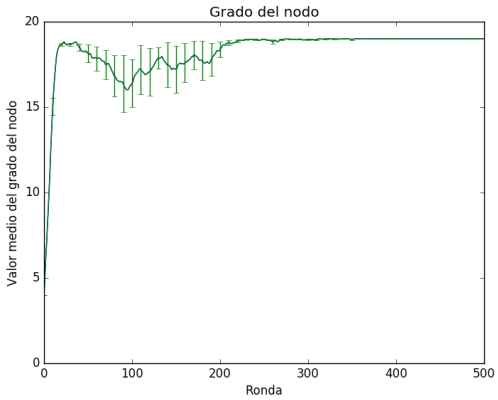
(a) Factor 0.7



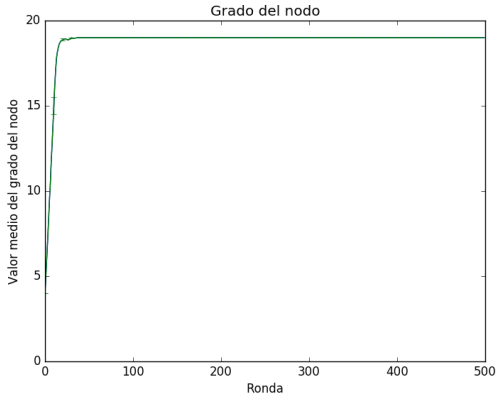
(b) Factor 0.9



(c) Factor 0.95



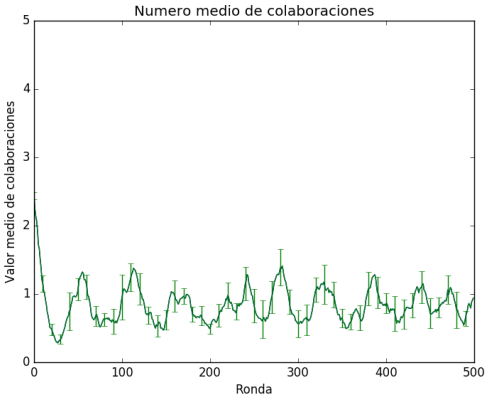
(d) Factor 1



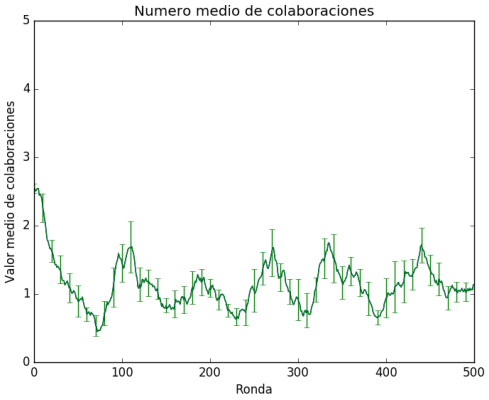
(e) Factor 1.05

Figura 52: Evolución del grado del nodo en función de factor, con T_{min} fijo en 0

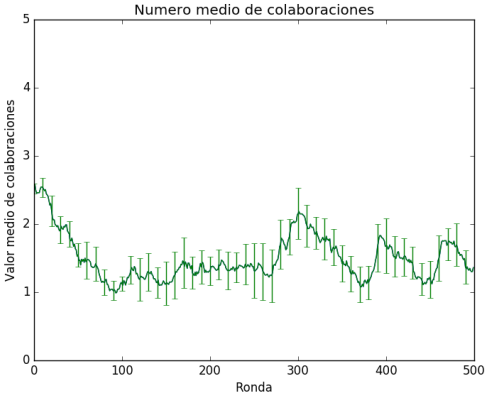
Número medio de colaboraciones.



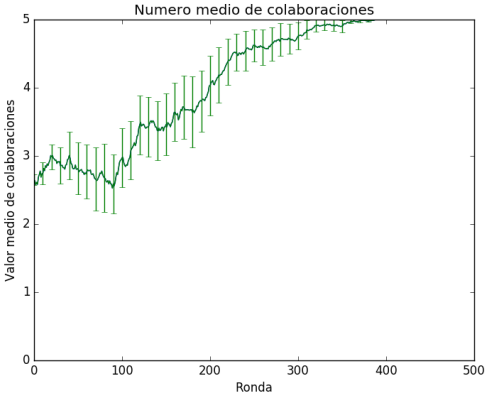
(a) Factor 0.7



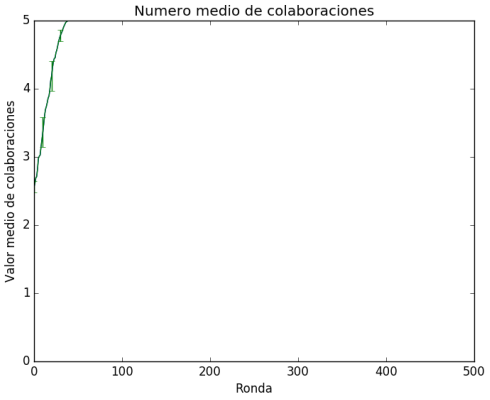
(b) Factor 0.9



(c) Factor 0.95



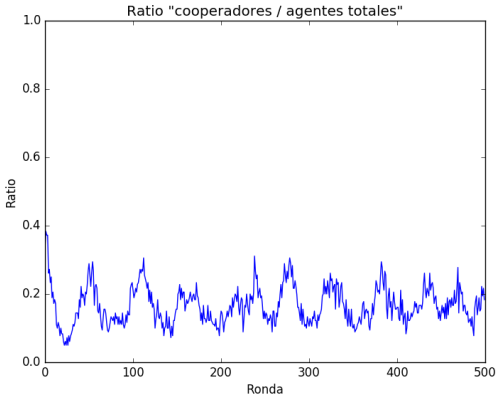
(d) Factor 1



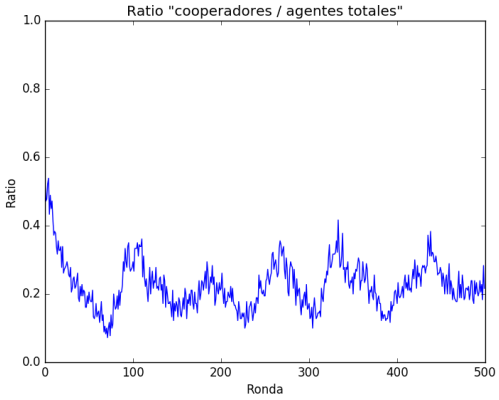
(e) Factor 1.05

Figura 53: Evolución del número medio de colaboraciones función de factor, con T_{min} fijo en 0

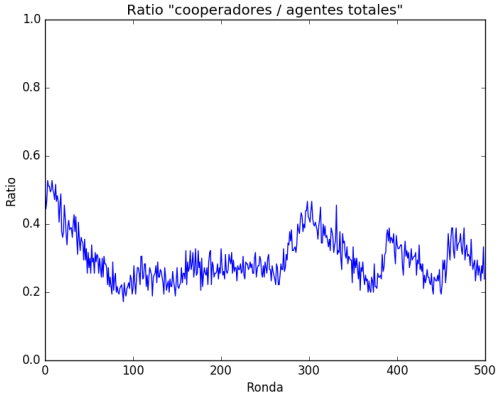
Ratio de colaboraciones.



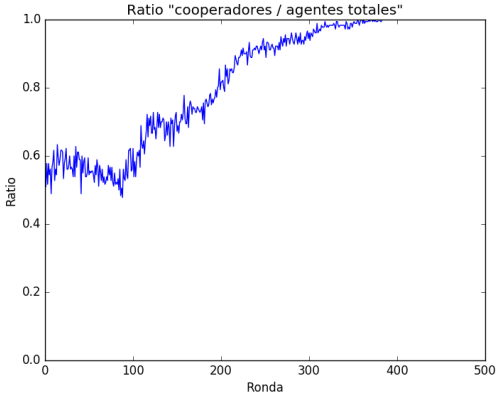
(a) Factor 0.7



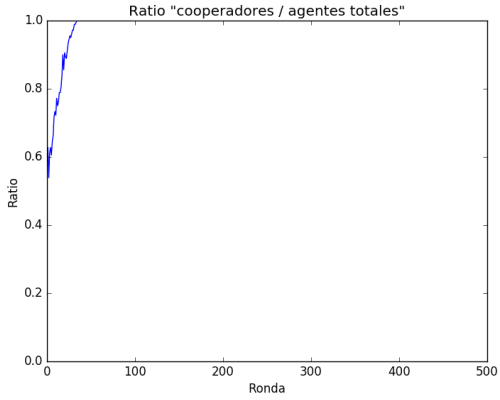
(b) Factor 0.9



(c) Factor 0.95



(d) Factor 1

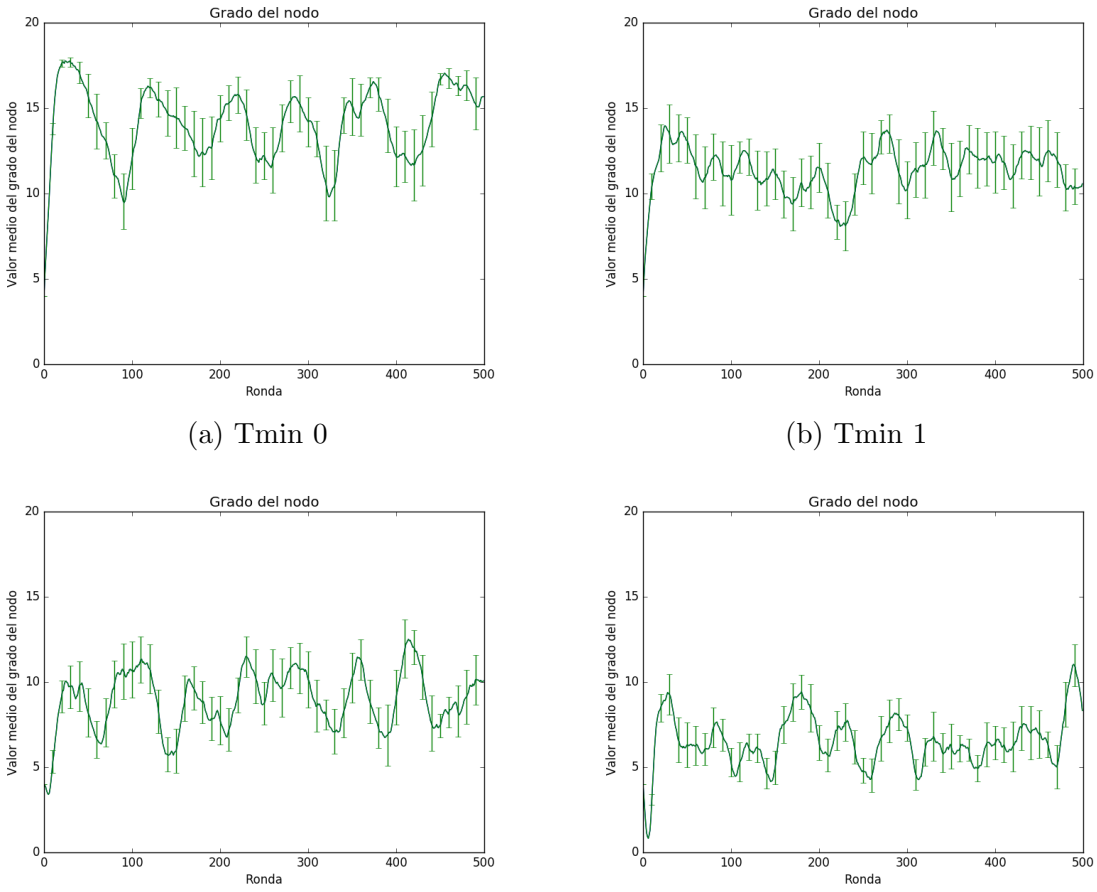


(e) Factor 1.05

Figura 54: Evolución del ratio de colaboraciones función de factor, con T_{min} fijo en 0

Variando *Tmin*, se obtienen las siguiente gráficas.

Evolución del grado medio del nodo, con *factor* 0.9.



(a) Tmin 0

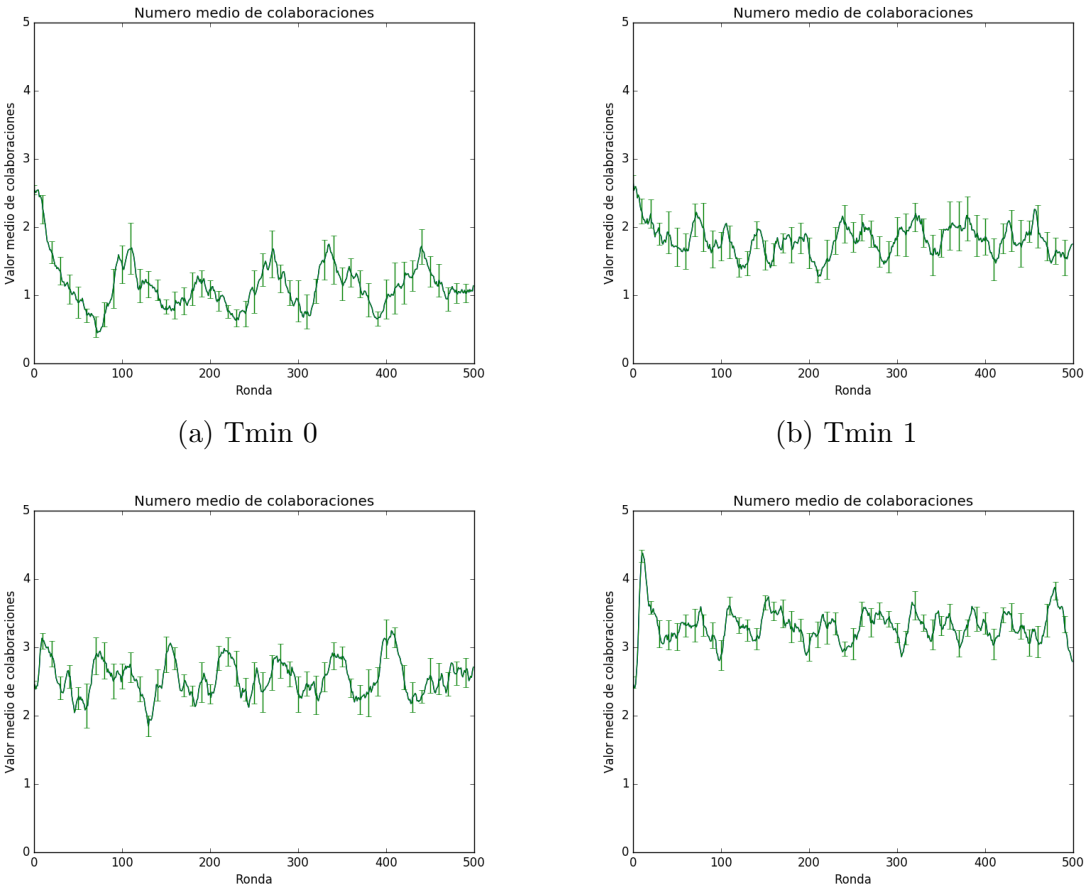
(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 55: Evolución del grado del nodo en función de Tmin, con factor fijo en 0.9

Evolución de la colaboración media, con *factor* 0.9.



(a) Tmin 0

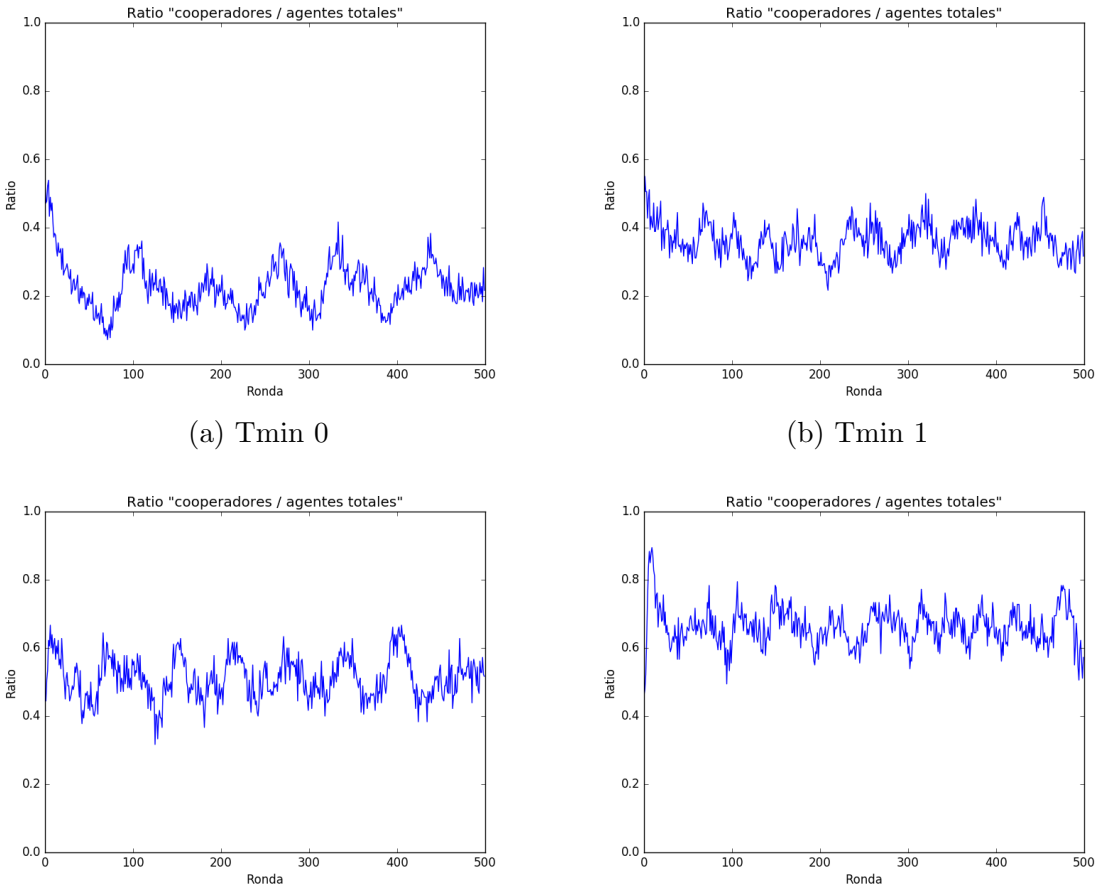
(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 56: Evolución del numero medio colaboraciones función de Tmin, con factor fijo en 0.9

Evolución del ratio de colaboraciones, con *factor* 0.9.



(a) Tmin 0

(b) Tmin 1

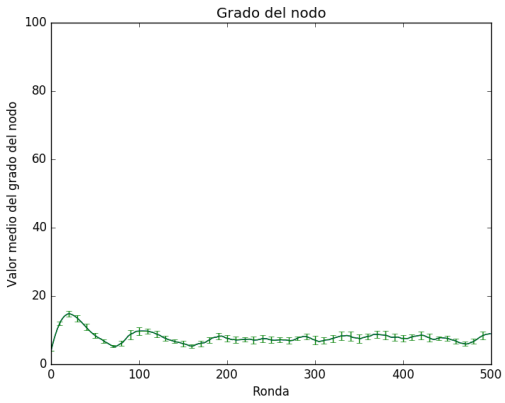
(c) Tmin 2

(d) Tmin 3

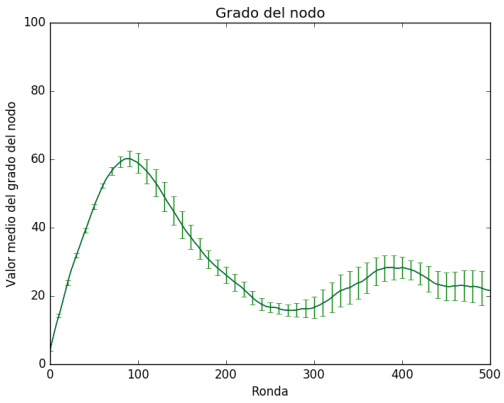
Figura 57: Evolución del ratio de colaboraciones en función de Tmin, con factor fijo en 0.9

100 nodos y 500 rondas.

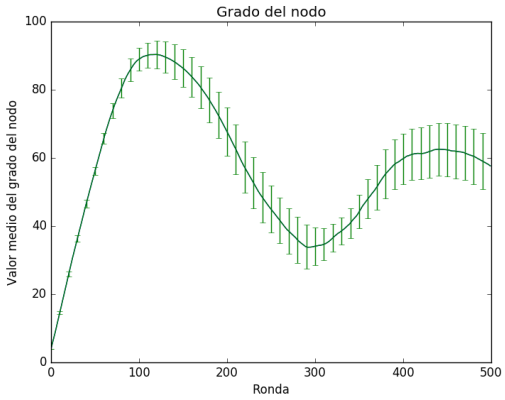
Grado medio del nodo.



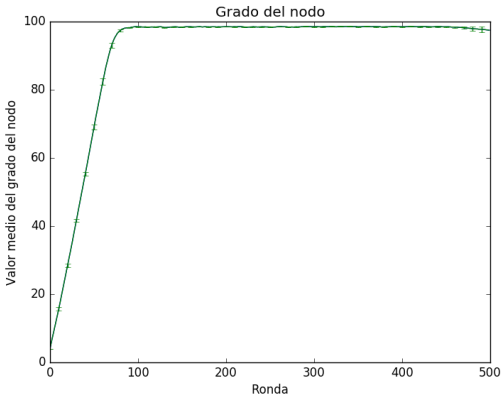
(a) Factor 0.7



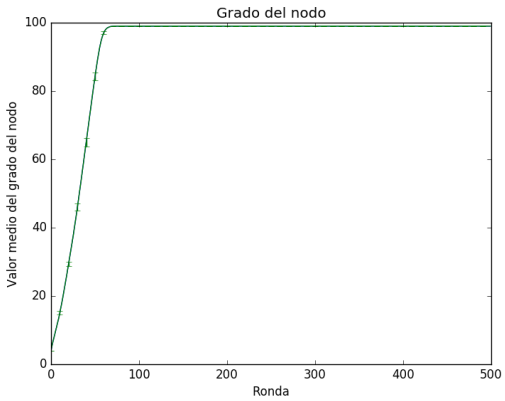
(b) Factor 0.9



(c) Factor 0.95



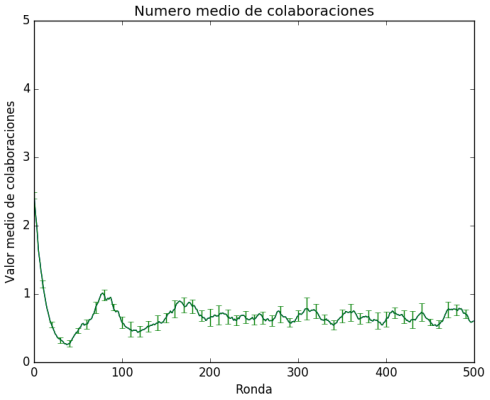
(d) Factor 1



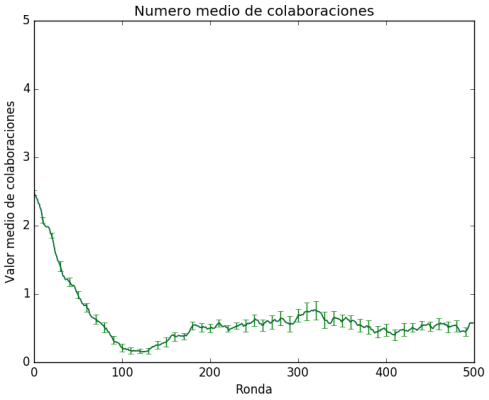
(e) Factor 1.05

Figura 58: Evolución del grado del nodo en función de factor, con T_{min} fijo en 0

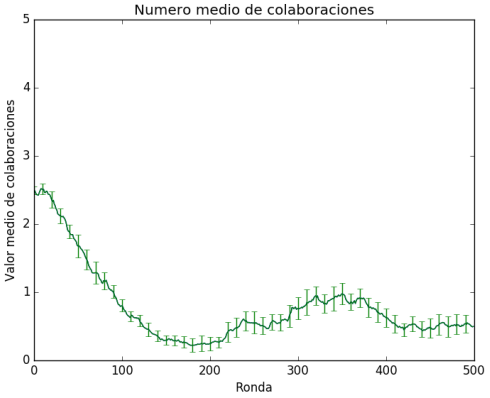
Número medio de colaboraciones.



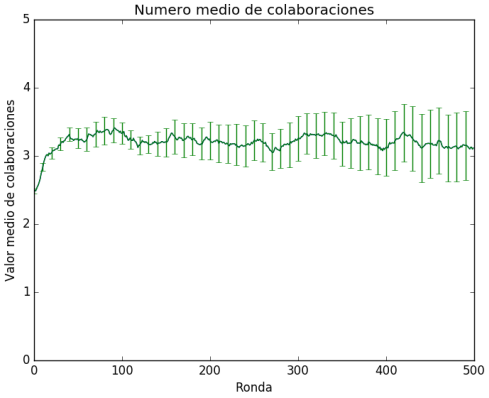
(a) Factor 0.7



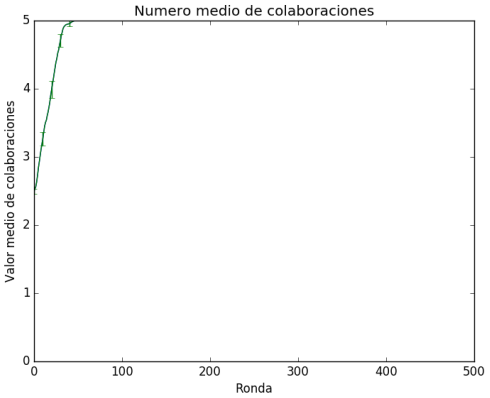
(b) Factor 0.9



(c) Factor 0.95



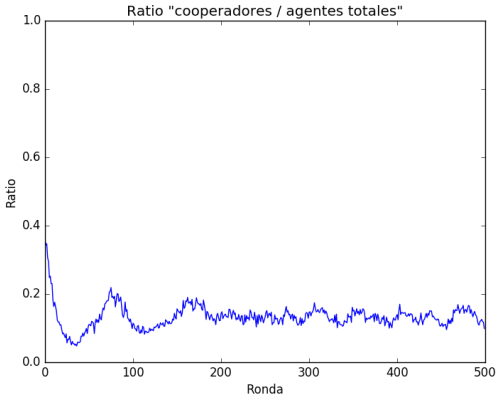
(d) Factor 1



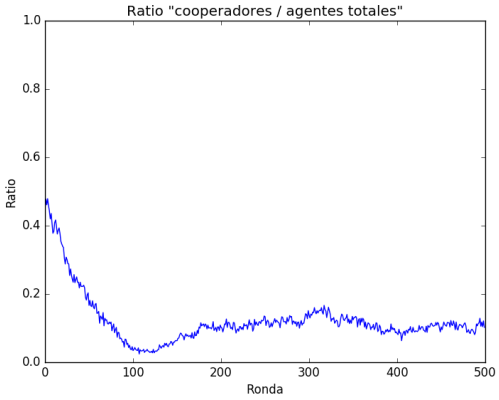
(e) Factor 1.05

Figura 59: Evolución del número medio de colaboraciones función de factor, con T_{min} fijo en 0

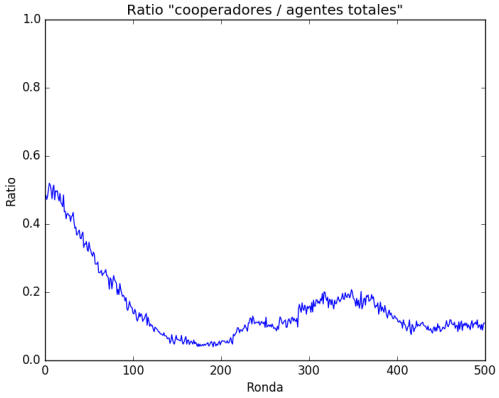
Ratio de colaboraciones.



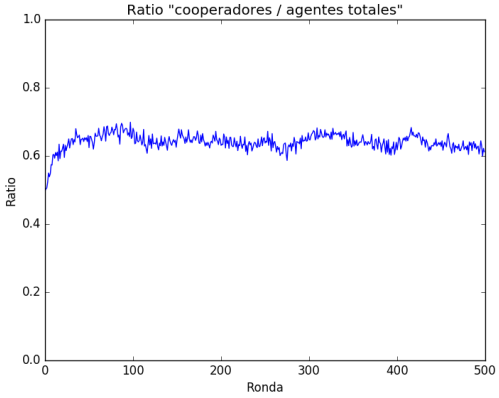
(a) Factor 0.7



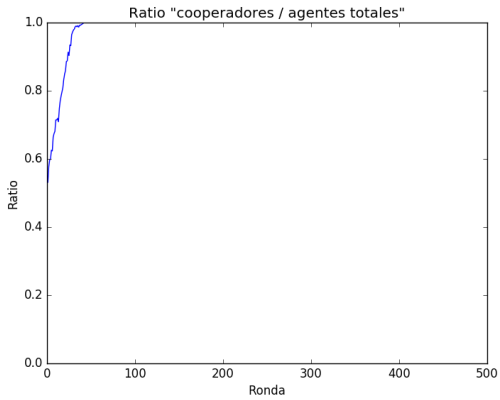
(b) Factor 0.9



(c) Factor 0.95



(d) Factor 1

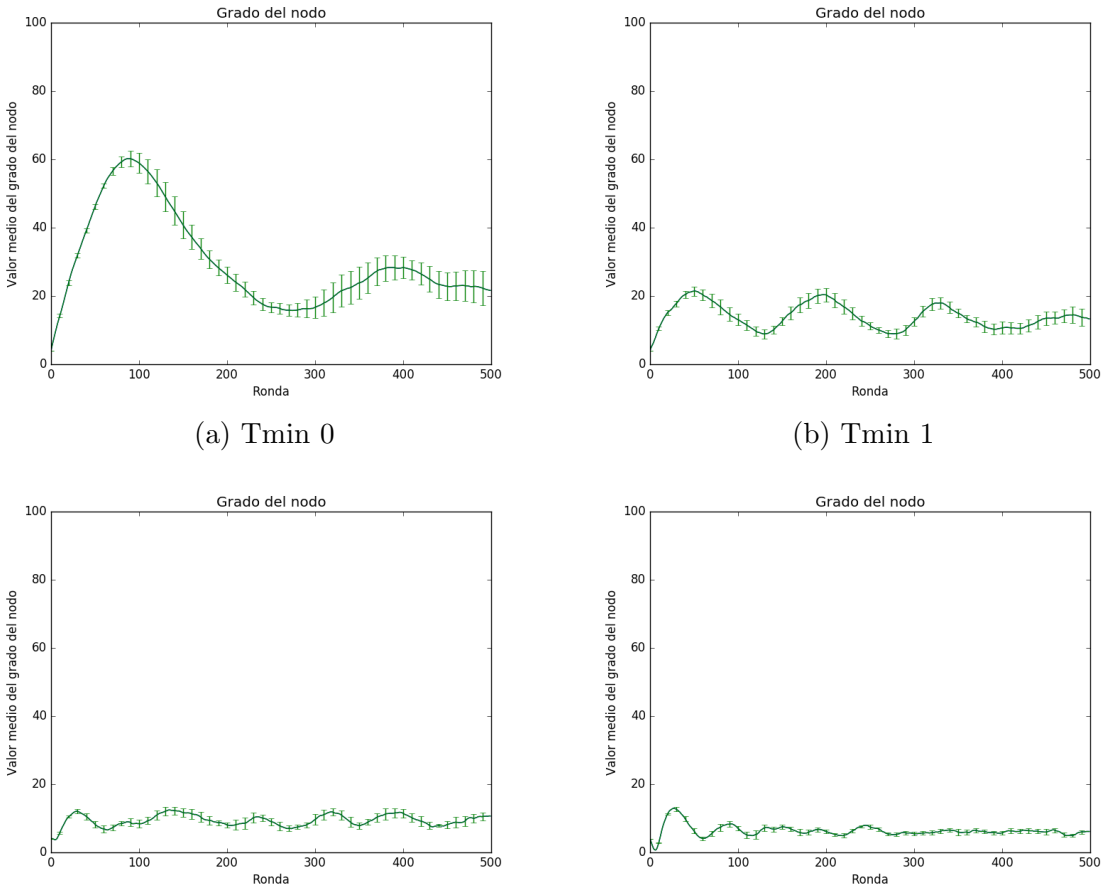


(e) Factor 1.05

Figura 60: Evolución del ratio de colaboraciones función de factor, con T_{min} fijo en 0

Variando *Tmin*, se obtienen las siguiente gráficas.

Evolución del grado medio del nodo, con *factor 0.9*.



(a) Tmin 0

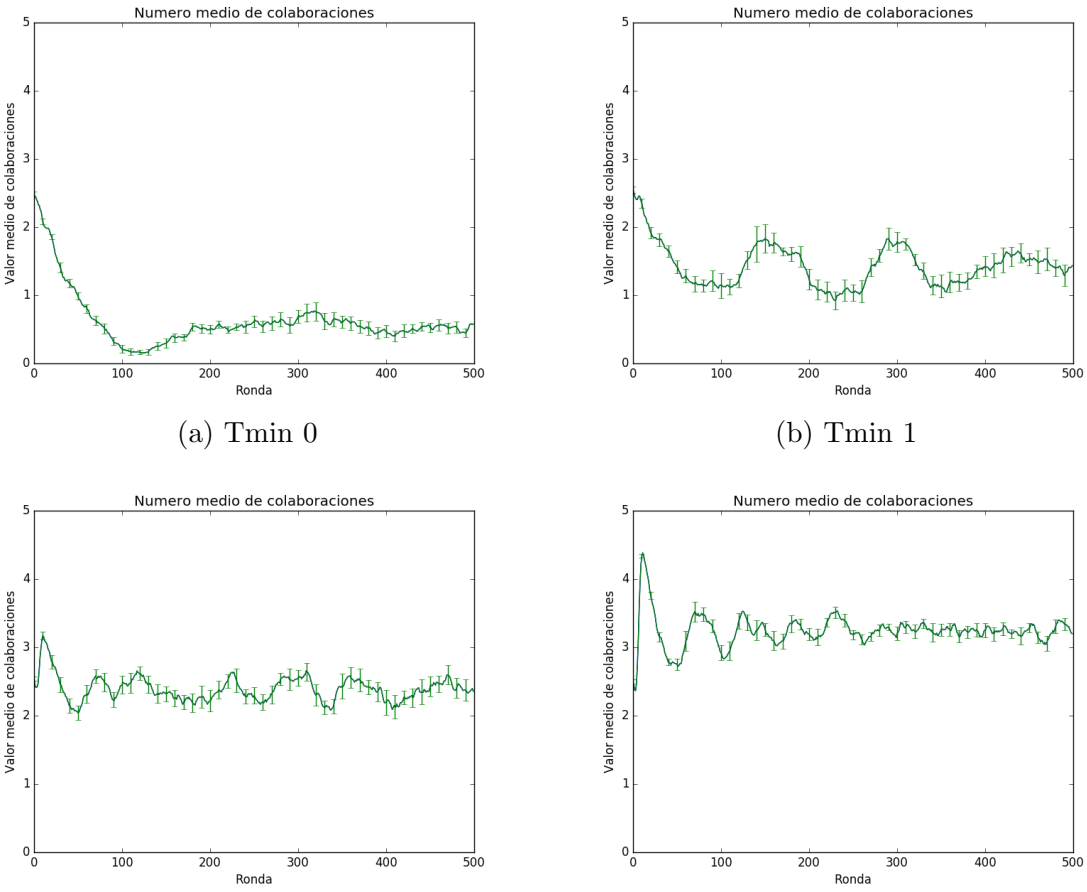
(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 61: Evolución del grado del nodo en función de Tmin, con factor fijo en 0.9

Evolución de la colaboración media, con *factor 0.9*.



(a) Tmin 0

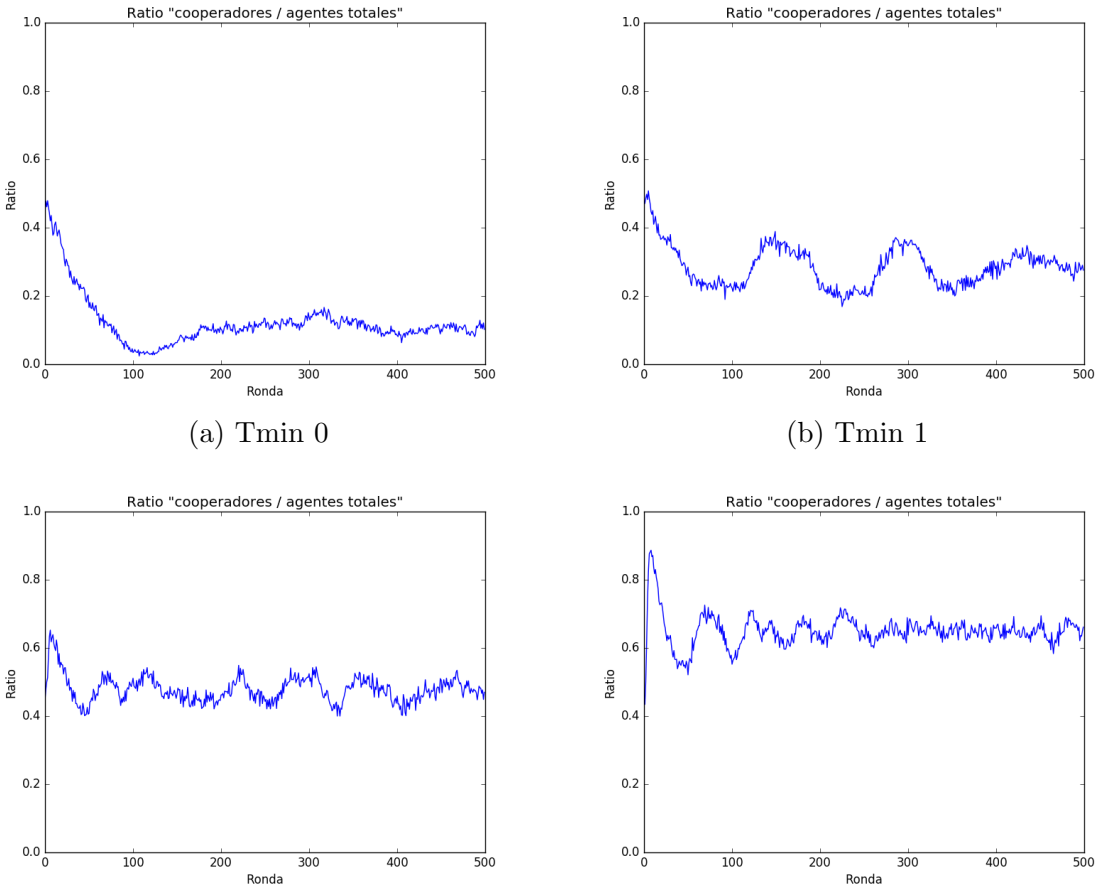
(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 62: Evolución del numero medio colaboraciones función de Tmin, con factor fijo en 0.9

Evolución del ratio de colaboraciones, con *factor* 0.9.



(a) Tmin 0

(b) Tmin 1

(c) Tmin 2

(d) Tmin 3

Figura 63: Evolución del ratio de colaboraciones en función de Tmin, con factor fijo en 0.9

Bibliografía

[1] J. A. Cuesta, C. Gracia-Lázaro, A. Ferrer, Y. Moreno, and A. Sánchez. Reputation drives cooperative behaviour and network formation in human groups. *Scientific reports*, 5:7843, 2014.

[2] Alberto Antonioni, Ángel Sánchez, and Marco Tomassini. Cooperation survives and cheating pays in a networked society with unreliable reputation. *Scientific Reports*, 6(27160), 2016.

[3] Morten Blomhoj. Mathematical modelling - a theory for practice. *National center for mathematics education*, 2005.

[4] Rodney C. Bassanezi and M. Sallet Biembengut. Modelización matemática: Una antigua forma de investigación, un nuevo método de enseñanza. *Números, revista de didáctica de las matemáticas*, (32):13–25, 1997.

[5] Wolfgang Weidlich. Sociodynamics - a systematic approach to mathematical modelling in the social sciences. 2002.

[6] Antonio Caselles Moncho. *Modelización y Simulación de Sistemas Complejos*. Publicacions de la Universidad de Valencia, 2008.

[7] Hassane Alla and René David. A modelling and analysis tool for discrete events systems: continuous petri net. *Perfomance Evaluation*, 33(3):175–199, 1993.

[8] *Game Theory*. Stanford Encyclopedia of Philosophy, 1997.

[9] EUMED. *Enciclopedia y Biblioteca Virtual de Ciencias Sociales, Económicas y Jurídicas*.

[10] A. Rapoport and A. M. Chammah. Prisoner's dilema. *University of Michigan Press*, 1965.

[11] Katherine A. Cronin, Daniel J. Acheson, Penélope Hernández, and Angel Sánchez. Hierarchy is detrimental for human cooperation. *Scientific Reports*, 5(18634), 2015.

[12] A. Antonioni, M. P. Cacault, R. Lalive, and M. Tomassini. Know thy neighbor: Costly information can hurt cooperation in dynamic networks. *PLOS ONE*, 9(10):e110788, 2014.

[13] K. Fehl, D. J. van der Post, and D. J. Semmann. Co-evolution of behavior and social network structure promotes human cooperation. *Ecol. Lett.*, 14:546–551, 2011.

[14] E. Gallo and C. Yan. The effect of repetitional and social knowledge on cooperation. *Proc. Natl. Acad. Sci. USA*, 2015.

[15] D. G. Rand, S. Arbesman, and N. A. Christakis. Dynamic social networks promote cooperation in experiments with humans. *Proc. Natl. Acad. Sci. USA*, 108:19193–19198,, 2011.

[16] J. Wang, S. Suri, and D. J. Watts. Cooperation and assortativity with dynamic partner updating. *Proc. Natl. Acad. Sci. USA*, 109:14363–14368, 2012.

[17] V. M. Eguíliz, M. G. Zimmerman, C. J. Cela-Conde, and M. S. Miguel. Cooperation and the emergence of role differentiation in the dynamics of social networks. *American Journal of Psychology*, 110(977-1008), 2005.

[18] F. C. Santos, J. M. Pacheco, and T. Lenaert. Cooperation prevails when individuals adjust their social ties. *PLoS Computational Biology*, 2:1284–1291, 2006.

[19] B. Skyrms and R. Pemantle. A dynamic model for social network formation. *Proc. Natl. Acad. Sci*, 97:9340–9346, 2000.