Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Modelización de la epidemia de la gripe A (H1N1) mediante redes aleatorias en un entorno de computación distribuida

Proyecto Final de Carrera

[Ingeniería Informática]

Autor: [Javier Ruiz Baragaño]

Directores: [Rafael J. Villanueva Micó, Maria José Rodríguez Álvarez]

[24/09/2012]

RESUMEN

El virus AH1N1/09 es un nuevo virus de la gripe que fue identificado en abril del 2009 en México y los EEUU. El virus se expandió muy rápido al resto del mundo y el 11 de junio de 2009, la Organización Mundial de la Salud declaró la pandemia de la gripe A (H1N1). La transmisión de este virus sólo es posible mediante contactos efectivos con sujetos infecciosos. Este proyecto trata de caracterizar la dinámica de transmisión del virus de la influenza A (H1N1) mediante una red dinámica de nodos aleatorios. El programa ha sido implementado para ser ejecutado de forma distribuida con el fin de reducir el tiempo total de cómputo. Con este simulador se busca la obtención de los parámetros característicos de la enfermedad, como la tasa de transmisión de la infección y transiciones entre estados de la enfermedad mediante datos reportados. Se han tomado datos reportados del estado de Nueva Esparta (Venezuela) para el ajuste del modelo. Con el fin de mejorar la velocidad y los resultados del modelo se ha realizado el ajuste con ayuda del algoritmo PSO (Particle Swarm Optimization). Debido a los altos requerimientos de eficiencia y portabilidad, el lenguaje de programación elegido ha sido C++, al igual que el entorno de desarrollo Code::Blocks por su flexibilidad multiplataforma. Se han generado versiones para ser ejecutadas en los sistemas Windows y Linux.

Palabras clave: AH1N1, random networks, SEIR, pandemic virus, PSO, Particle Swarm Optimization, distributed environment.

AGRADECIMIENTOS

Ha llegado el momento de agradecer el gran apoyo recibido durante el desarrollo de este proyecto, así como en el transcurso de mi estancia en calidad de becario en el IMM de la UPV. Debo expresar mis mayores gratitudes al tutor del mismo, Rafael Villanueva, por su ayuda incondicional en el desarrollo del presente y, en general, por el tiempo que he podido disfrutar trabajando en aquellos otros proyectos en los que se me ha permitido intervenir gracias a él.

Asimismo, debo agradecer a la directora del proyecto Maria José Rodríguez, el haberme brindado la oportunidad de conocer tanto a mi tutor, como el Instituto de Matemática Multidisciplinar, ya que, sin su guía, no hubiera sido posible llevar a término este trabajo.

También he de manifestar mi más sincero agradecimiento a aquellos profesores y compañeros con los que he tenido la fortuna de colaborar. Javier Villanueva puso mucho interés, al echarme una mano en los comienzos de mi labor, mostrándome siempre el camino correcto. Con la ayuda de Gilberto González he podido fundamentar y extender gran parte del contenido aquí tratado. Además, aunque fueran proyectos al margen de este, Almudena y Juan Carlos también han tomado un papel muy importante en mi formación extrauniversitaria.

Por último, quiero dar las gracias al Instituto de Matemática Multidisciplinar, a la Universidad Politécnica de Valencia y, especialmente, a la Escuela Técnica Superior de Informática por los conocimientos que me han otorgado durante estos cinco últimos años.

TABLA DE CONTENIDOS

1. Introducción 12
1.1 Virus del AH1N1 12
1.2 Modelos epidemiológicos 14
1.2.1 Modelos de redes 14
1.2.2 Modelo AH1N1 SEIR implementado 14
1.2.3 Pseudocódigo del algoritmo 16
1.2.4 Datos reportados del AH1N1 y su adaptación al modelo epidemiológico 18
1.2.5 El programa calculador del AH1N1 20
1.3 Computación distribuida 21
1.3.1 Introducción 21
1.3.2 Equipos utilizados 22
1.3.3 Sistema distribuido empleado: Sísifo 23
1.4 Propósito de este proyecto 25
1.4.1 Problemas para ajustar modelos de redes con datos reales y poblaciones grandes 25
1.4.2 Barrido de parámetros 25
1.4.3 Algoritmo P.S.O. (Particle Swarm Optimization) 26
1.4.4 Aleatoriedad intrínseca del modelo. Distribuciones probabilísticas (Poisson) 29
2. El modelo del AH1N1 32
2.1 Análisis 32
2.1.1 Introducción 32
2.1.2 Pseudocódigo detallado del modelo en pasos 32
2.1.2 Entrada y salida del modelo 34
2.1.3 Requisitos del ejecutable 34
2.2 Diseño 35
2.2.1 Metodología de desarrollo 35
2.2.2 Descripción de clases principales 37
2.2.3 Librerías utilizadas 39
2.2.4 Funcionamiento de la aplicación 39
2.2.5 Documentación detallada 39

3. Demonio PSO 40
3.1 Análisis 40
3.1.1 Introducción 40
3.1.2 Pseudocódigo detallado del algoritmo en pasos 41
3.1.3 Entrada y salida del demonio 42
3.2 Diseño 43
3.2.1 Descripción de clases principales 43
3.2.2 Librerías utilizadas 45
3.2.3 Funcionamiento de la aplicación 46
3.2.4 Documentación detallada 46
3.3 Adaptación del calculador AH1N1 con el demonio PSO 47
3.3.1 Ficha 47
3.3.2 Script 48
3.3.3 Otras consideraciones 49
4. Resultados 50
4.1 Muestra 50
4.2 Barrido 50
4.3 PSO 52
5. Conclusiones 53
Anexo A: Configuración del sistema Sísifo (Javier Villanueva) 55
A.1. Componentes del cliente de Sísifo 55
A.2. Componentes del servidor de Sísifo. 55
A.2.1 Archivo config.cfg 56
2.2 Archivo IP_filtradas.cfg 58
A.2.3 Archivo Ejecutor|exe.exe 58
A.2.4 Archivo Ejecutor|exe.lin 58
A.2.5 Archivo Servidor 59
A.2.6 Archivo Monitor 59
A.3. Reparto de memoria y de CPU 60
A.4. Consideraciones de implementación 60
A.4.1 Calculador 60
A.4.2 Definición del problema 60

A.4.3 Definición de la solución 61
A.4.4 Sistema de watchdog del cliente 62
A.5. Otras notas 62
Anexo B: Manual del calculador AH1N1 63
B.0. Introducción 63
B.1. Parámetros de ejecución 63
B.2. Fichero de configuración 64
B.3. Fichero solución 66
B.4. Interfaz 67
B.5. Análisis de soluciones 67
B.6. Herramientas adicionales 68
B.7. Solución de problemas 69
Anexo C: Manual del demonio PSO 70
C.0. Resumen 70
C.1. Parámetros de entrada 70
C.2. Fichero de configuración 71
C.3. Guardado y restauración del estado del algoritmo 77
C.4. El intérprete y fichero script 78
C.4.1 Operadores permitidos 78
C.4.2 Variables 79
C.4.3 Funciones integradas 79
C.4.4 Condición IF 80
C.4.5 Bucle FOR 80
C.4.6 Otras puntualizaciones 80
C.5. Puesta en marcha 81
C.6. Mensajes de error y soluciones 82
Anexo D: Documentación del calculador AH1N1 83
Índice de clases 83
Lista de clases 83
Documentación de las clases 83
Referencia de la Clase CContabilidad 83
Referencia de la Clase CFicha 86

Referencia de la Clase CModelo 88
Referencia de la Estructura CPersona 93
Anexo E: Documentación del demonio PSO 94
Índice de clases 94
Lista de clases 94
Documentación de las clases 94
Referencia de la Clase CFicha 94
Referencia de la Clase CGenProblemas 96
Referencia de la Clase CInterprete 98
Referencia de la Clase CPajaro 99
Referencia de la Clase CParametro 101
Referencia de la Clase CPSO 102
Referencia de la Clase CPunto 104
Anexo F: Gramática y tokens del intérprete 105
Bibliografía 109

1. INTRODUCCIÓN

1.1 VIRUS DEL AH1N1

El virus pandémico AH1N1/09 es un nuevo virus de la gripe de origen humano, porcino y aviar. En Abril de 2009 se detectó por primera vez el virus en Méjico y los Estados Unidos de América [1] [2]. De 381 pacientes estadounidenses de los cuales se tienen datos, el 18% afirmó que viajó a la ciudad de Méjico 7 días antes de su malestar provocado por el virus. De estos pacientes, 7 fueron posteriormente hospitalizados [3]. El virus se extendió rápidamente al resto del mundo y el 11 de Junio de 2009, la OMS determinó a la gripe A (H1N1) como una nueva pandemia [2].

Las primeras pruebas preliminares realizadas por el centro de control y prevención de enfermedades (CDC) indicaron que el virus era una nueva cepa que contenía elementos genéticos procedentes de aves, cerdos y seres humanos [2] . La transmisión del virus AH1N1 sólo es posible mediante el contacto efectivo con un individuo infeccioso.

Las intervenciones habituales incluían la cuarentena, el aislamiento, restricciones de viaje, clausura de lugares públicos, cuarentena individual y general basada en el miedo y la cancelación de eventos [2]. Estas medidas tuvieron costes económicos y repercusiones sociales, como pérdida de días de trabajo, absentismo escolar y una disminución de beneficios empresariales [2][4][5][6].

Es importante mencionar que el proceso de producción de una vacuna suele durar alrededor de 6 meses en completarse, sin embargo en los Estados Unidos de América la vacuna de la pandemia de la gripe A (H1N1) estaba disponible a partir de Octubre de 2009 [4].

Puesto que hay más de 14.000 muertes mundiales confirmadas por el virus AH1N1/09, es importante entender la dinámica de evolución de este virus. Cada año mueren aproximadamente 36.000 personas debido a gripes estacionarias u otros casos gripales en los Estados Unidos [4].

Se han presentado algunos modelos para estudiar la dinámica de propagación del virus H1N1 en todo el mundo. El modelo epidemiológico clásico SIR con una función estacional forzada se ha usado para modelar la propagación de la gripe AH1N1/09 en la población estadounidense [7]. Adicionalmente, el modelo clásico SEIR se ha usado para predecir los individuos infectados, número de hospitalizaciones y efectividad de la vacunación en una ciudad japonesa. En Singapur se ha mezclado con métodos estadísticos para pronosticar la prevalencia del AH1N1 dentro del país [4][8].

Un modelo estructurado global, que integraba la movilidad y datos de las rutas de transporte en el mundo entero, fue desarrollado para identificar los posibles escenarios plausibles en el despliegue de la pandemia del virus AH1N1/09 [9]. La estacionalidad forzada es modelada mediante la inclusión de una función sinusoidal de periodo de 12 meses. Adicionalmente se realizaron un total de 2000 pruebas estocásticas de la evolución pandémica global, con un modelo

SEIR, para poder estimar los valores de los parámetros de transición entre las clases del modelo. Es importante tener en cuenta que aunque algunos de los parámetros pueden ser determinados basándose en investigaciones anteriores, otros parámetros deben ser estimados parametrizando el modelo con los datos disponibles. Por tanto, parametrizar modelos epidemiológicos con datos reales se convierte en el problema o la tarea principal dentro del campo de enfermedades epidemiológicas infecciosas [10].

Incorporar patrones de contacto de la población realistas puede incrementar la efectividad de estos modelos epidemiológicos. El proceso de las enfermedades dentro del mundo real es muy complejo, puesto que pueden afectar de forma distinta a diferentes grupos de edades, pueden haber varios periodos de incubación según el grupo de edad o pueden difundirse irregularmente según el entorno. Varios investigadores han desarrollado modelos fiables usando simulaciones basadas en agentes, en las cuales cada individuo se examina exhaustivamente. Obviamente estos modelos incluyen una compleja parametrización y a menudo un tiempo de computación. Hablando en términos de complejidad, el modelo de redes se encuentra entre los modelos comportamentales y las simulaciones basadas en agentes [11].

1.2 MODELOS EPIDEMIOLÓGICOS

1.2.1 MODELOS DE REDES

El estudio de las redes, y generalmente de redes sociales, es de gran importancia en un amplio número de disciplinas [12]. Simular redes de un tamaño moderado es una tarea bastante costosa en tiempo computacional: una epidemia en una población de algunos cientos de miles de nodos puede tomar horas de cálculo para obtener resultados medios en un rango de parámetros del modelo. [13]

A nuestro entender han habido muy pocos antecedentes respecto a modelos matemáticos basados en redes que se hayan usado para el estudio de la dinámica de las epidemias dentro del mundo real. Algunos trabajos anteriores interesantes son: Un modelo de redes para estudiar la expansión del Virus Respiratorio Sincitial (VRS) en la Comunidad Valenciana (España) [14] [15]; un modelo de redes del virus dengue en Singapur [16]; una red gráfica completa modelando la obesidad como una epidemia social [17]; una red social para investigar el brote de la gripe equina en Australia [18]; un modelo de redes de un brote del virus del sarampión en Hallegoch (Alemania) en el año 1861, afectando a 188 individuos [19]; y otro modelo de redes que describe los datos empíricos de la epidemia del cólera entre los años 2000/2001 dentro de la ciudad de Kwa Zulu-Natal Province (Sudáfrica) [20].

En este proyecto se propone el desarrollo y uso de redes aleatorias en un entorno de computación distribuida. Mediante éste modelo podemos predecir y/o reproducir la dinámica del virus AH1N1 en diversas poblaciones o ciudades con parámetros característicos. concretamente se ha estudiado la ciudad de Nueva Esparta (Venezuela), ya que se trata de una ciudad de la que tenemos datos reales.

Usamos un modelo SEIR dentro del modelo de redes para explicar el comportamiento de la pandemia de la gripe A(H1N1) dentro de la región. Para reproducir diferentes periodos de tiempo de los que tenemos datos, usamos el mismo modelo de redes. En este caso realizamos simulaciones de nuestro modelo de redes SEIR para analizar la efectividad del modelo y el comportamiento del virus pandémico, en 2009 dentro del estado venezolano de Nueva Esparta.

1.2.2 MODELO AH1N1 SEIR IMPLEMENTADO

En nuestro modelo se considera el proceso de nacimientos y muertes dentro de la población, mediante tasas de natalidad y mortalidad. Sin embargo, un modelo sin estos parámetros puede considerarse viable debido a que la escala de tiempo de la propagación del virus AH1N1 es comúnmente más rápida que este proceso. La comprobación de un nacimiento o defunción se realiza en cada paso de tiempo (día o semana), y en el caso de que se trate de una defunción, se toma un nodo de forma aleatoria y se borra junto con sus aristas. La implementación de ésta y similares características es interesante en caso de que se quiera modificar el algoritmo

para otro tipo de virus, ya que los modelos de contactos suelen tener bastantes similitudes entre sí.

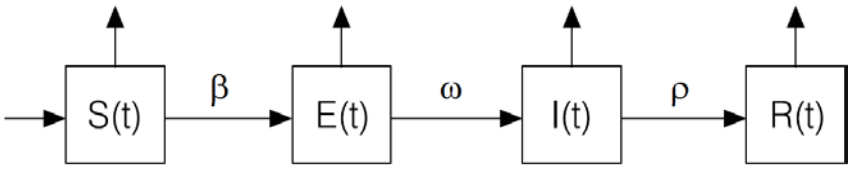


Fig.1: Modelo SEIR

El modelo SEIR considera que toda la población **N(t)** está dividida en cuatro clases: los susceptibles **S(t)**, es decir, aquellos individuos sanos que pueden contagiarse; la clase de latentes **E(t),** los cuales están infectados pero aun no contagian; la clase de los infecciosos **I(t)**, que son los que estando enfermos pueden transmitir la enfermedad; y finalmente la clase de los recuperados **R(t)**, que corresponden a aquellos que, habiéndose enfermado y curado, han adquirido inmunidad permanente. Los parámetros **β**, **ω** y **ρ** representan la probabilidad de pasar de un estado o clase al siguiente. Adicionalmente, hay dos parámetros asociados a las tasas de natalidad y mortalidad ya mencionadas, se tratan de **μ** y **δ** respectivamente. La Figura 1 muestra el diagrama del modelo SEIR que presenta la dinámica de un individuo en general [21].

Las probabilidades de paso **ω** y **ρ** entre los estados del modelo SEIR se modelan mediante procesos probabilísticos llamados distribuciones de Poisson de medias **ω** y **ρ**. Dicha distribución ha sido usada en varios ámbitos científicos relacionados con la simulación y muestreo, debido a comportamientos que siguen patrones similares a los encontrados en nuestra vida cotidiana. Por esta razón se ha considerado oportuna su inclusión dentro de este modelo.

Las redes de contactos se crean partir de datos reales y son usadas para el estudio de algunos temas relacionados con la salud. La propagación de diversas enfermedades infecciosas está determinada por encuentros aleatorios entre personas que viven en el mismo área geográfica, por ejemplo en paradas de autobús, paseando por la calle, encontrándose en centros comerciales, etc... La transmisión del virus AH1N1 es posible mediante el contacto efectivo con un individuo infeccioso. En este proyecto se considera el modelo aleatorio basado en redes como el más apropiado para el modelado de la transmisión del virus AH1N1. Sin embargo, la monitorización de todas las actividades detalladas de un individuo en el mundo real para determinar posibles contactos infecciosos es compleja, y requiere una gran cantidad de conocimiento y de recursos humanos.

Todo este comportamiento se quiere plasmar dentro de nuestra red de contactos, por tanto, cada individuo de la población real se representa como un nodo o vértice dentro de nuestro modelo de redes. Si pensamos en teoría de grafos, una población se representaría por un conjunto de vértices y aristas no ponderadas, representando un contacto entre dos individuos dicha arista.

Este contacto indica la posibilidad de transmisión del virus entre un nodo que esté infectado y otro que no lo esté. Para indicar el grado medio de relaciones o aristas que tiene cada individuo, utilizamos el parámetro **k**. Por tanto, en nuestro modelo, si tenemos un total de **N** nodos y **k** relaciones por nodo, podemos calcular el número total de relaciones deduciéndolo del lema del apretón de manos de Euler: **k**·**N**/2.

Como estado inicial de nuestro modelo SEIR, se propone una pequeña fracción de infectados dentro de nuestro modelo. También se puede proponer la existencia de un pequeño grupo de latentes, pero los experimentos nos dicen que esto supone un comportamiento similar al de los infectados, teniendo en cuenta el retraso de propagación debido a la transición de los latentes al estado infeccioso. También suponemos que en nuestra red no hay ningún individuo inmune al virus, o en el estado de recuperado, puesto que es la primera vez que la población ha sido expuesta a este tipo de virus.

El hecho de incluir las tasas de natalidad y mortalidad, obliga a que nuestra red sea de carácter dinámico, es decir, que ésta debe de permitir la adición y eliminación de nodos posteriormente a su generación. En el caso de un nacimiento, se ha de agregar un nuevo nodo a la red y relacionar correctamente. Para determinar el número de enlaces del recién nacido usaremos una distribución de Poisson de media **k**. Cuando se trate de un caso de mortalidad, basta con eliminar dicho nodo y sus enlaces de nuestro modelo de redes.

1.2.3 PSEUDOCÓDIGO DEL ALGORITMO

Concretamente se exponen los pasos del algoritmo propuesto que simula la evolución del virus AH1N1 en nuestro modelo SEIR basado en redes aleatorias:

- Inicialización:

- Creamos un conjunto de **N** nodos
- Para la generación de contactos entre nodos, o aristas entre los vértices, realizamos los enlaces entre nodos de forma aleatoria. Para ello, escogemos un nodo aleatoriamente entre 1 y **N**, una vez escogido, volvemos a escoger otro nodo entre 1 y **N** de la misma forma y teniendo en cuenta que no puede ser el mismo nodo que el primero escogido. Una vez tenemos ambos nodos, comprobamos si están relacionados ya, en este caso se vuelve a escoger dos nodos nuevos, y en caso contrario, se relacionan. Este proceso se repite tantas veces como número de aristas totales tenga nuestro modelo (**k**·**N**/2), de esta forma ya tenemos nuestra red de contactos debidamente creada.
- Generamos distribuciones de Poisson de media **ω**, **ρ** y **k**.
- Se inicializan las variables correspondientes al acumulado de nacimientos y muertes por día, ya que estas variables serán incrementadas en cada paso de tiempo **Δt**.

- Evolución; para cada instante de tiempo **Δt** dentro del periodo total **T**:

- Primero comprobamos si hay infectados o latentes en nuestro sistema, en caso contrario, la simulación habrá terminado.

- Se incrementan los valores acumulados correspondientes a la natalidad y mortalidad de nuestro modelo. Dicho incremento es igual a la cantidad de nacimientos y muertes que hay en el periodo de tiempo **Δt**. En el momento que los valores acumulados superen la unidad, es decir, 1, nacerá una persona o morirá otra respectivamente. El nacimiento es similar a cuando se relaciona un contacto con el resto de la red, pero se usa en este caso la distribución de Poisson de media **k** para determinar el número de relaciones que tiene el recién nacido.

- Para cada individuo de nuestra red, se incrementa **Δt** el tiempo que lleva en el estado actual y se determina su estado de salud. Para cada estado de salud se realiza un procedimiento distinto:

    - **Susceptible**: Se comprueba si éste tiene contactos infectados. En caso positivo, para cada contacto infectado se genera un número aleatorio entre 0 y 1 (ambos incluidos). Si dicho valor es menor que el parámetro **β**, el nodo queda infectado, y pasa a tener el estado de salud de latente.

    - **Latente**: Tomamos el porcentaje acumulado de la distribución de Poisson de media **ω,** desde el principio de la distribución hasta el valor correspondiente al tiempo que lleva en el estado de salud de latente, y lanzamos un número aleatorio, operando de la misma forma que en el caso anterior. En caso que dicho número sea menor al porcentaje generado, el estado de salud del individuo cambia a infectado. Desde este preciso momento puede ser infeccioso para el resto de vecinos susceptibles que queden por analizar en esta iteración.

    - **Infectado:** Realizamos el mismo procedimiento como si se tratara de un individuo con el estado de salud latente, pero esta vez con una distribución de Poisson de media **ρ**. Si se cumple la condición, el individuo pasa al estado recuperado, y por tanto, inmune del virus.

    - **Recuperado**: En este caso no se hace nada.

- Todo este proceso se repite hasta que el tiempo total de nuestra simulación sea igual o haya superado el parámetro T, con el que consideramos que la simulación ha terminado.

Para cada paso de tiempo **Δt** de la simulación nos guardamos las estadísticas de la red para su posterior análisis, en el que se incluyen el número de susceptibles, latentes, infectados y recuperados de nuestro modelo.

1.2.4 DATOS REPORTADOS DEL AH1N1 Y SU ADAPTACIÓN AL MODELO EPIDEMIOLÓGICO

Los casos informados de la pandemia de la gripe AH1N1 representan aquellos casos que fueron dados como positivos del virus de la gripe AH1N1 mediante pruebas de laboratorio. Los datos fueron muestreados semana a semana y corresponden a las fechas en las cuales se tomaron las muestras. La información obtenida pertenece a instituciones de salud públicas, donde los pacientes presentaban síntomas agudos de la enfermedad. Las pruebas sobre el virus de la gripe AH1N1 fueron realizadas tomando muestras de la nariz o aspiraciones, a partir de las cuales se realizaba una reacción en cadena de la polimerasa con transcriptasa inversa (RT-PCR, del inglés reverse transcription polymerase chain reaction) o un cultivo viral de las mismas. En la tabla siguiente se muestran todos los casos positivos obtenidos en el estado venezolano de Nueva Esparta, obteniendo el número mayor de infecciones en la semana 34 de 2009.

Semana	24	25	26	27	28	29	30	31	32	33	34	35	36	37
Casos	2	3	0	0	1	1	3	0	2	1	13	9	8	1
Semana	38	39	40	41	42	43	44	45	46	47	48	49	50	
Casos	5	2	2	1	0	0	0	1	0	0	1	0	0	

Tabla de infectados del virus epidémico AH1N1 en el estado venezolano de Nueva Esparta (2009). Datos cedidos por el departamento de salud pública de Venezuela.

Los parámetros del modelo SEIR se deben ajustar correctamente, de forma que las soluciones de nuestro modelo concuerden lo mayor posible con los datos reales, sin embargo, la epidemiología del virus pandémico AH1N1 no se conoce de forma precisa o no actúa de la misma forma en cada persona. El valor de los parámetros siguientes se ha determinado a partir de los estudios realizados y de datos reales.

Los parámetros característicos de nuestra red de contactos lo forman **N** y **k**, explicados anteriormente. Sin embargo, no sabemos exactamente el valor del parámetro **k**, y tampoco podemos determinarlo a base de suposiciones. Aunque el periodo de simulación es suficientemente corto para no tener en cuenta la natalidad y mortalidad, se ha preparado el modelo para que registre este comportamiento. En este caso las tasas de natalidad (**nu**) y mortalidad (**d**) se han fijado a 17,16‰ al año.

El periodo de incubación del virus AH1N1, según los datos obtenidos, es del orden de 2 a 10 días con una media de 6 días. Por tanto, el tiempo medio en el estado de latente **E(t)** se ha asumido que es del orden de $\boldsymbol{\omega} = 5\ a\ 6$ días. El periodo infeccioso **I(t)** ha sido estimado entre cuatro y siete días, estableciéndose en el modelo con $\boldsymbol{\rho} = 7$ días. Sin embargo, en niños pequeños y pacientes con problemas inmunitarios o severamente enfermos, el periodo infeccioso puede ser mayor. Debido a la incertidumbre de estos valores, los parámetros **ω** y **ρ** pertenecerán también al

grupo de ajuste, pero en un rango bastante limitado con el fin de hacer el ajuste lo más fino posible.

Hay que destacar que cada uno de los parámetros **ω** y **ρ** pueden ser interpretados como la media del tiempo estimado del periodo de tránsito entre dos subpoblaciones. Por tanto, los valores numéricos calculados anteriormente no deben ser considerados como un tiempo constante o fijo para que el individuo transite al nuevo estado de salud, por esta razón usamos distribuciones de Poisson con una media igual a los parámetros calculados anteriormente.

Los datos semanales confirmados de infectados de la gripe pandémica AH1N1 no corresponden exactamente a la subpoblación **I(t)**, puesto que sólo una fracción **s** (factor de escala) de las personas que se sienten enfermas deciden ir al médico que informó de dicha infección. Adicionalmente algunas de las personas infectadas no muestran todos los síntomas. Este parámetro **s** puede ser determinado a posteriori en el análisis de resultados mediante el método de mínimos cuadrados.

El parámetro del que no se tiene ningún dato es el **β**, el cual modela la transmisibilidad del virus entre dos nodos en el periodo **Δt** de tiempo. Si el parámetro tiene un valor muy pequeño, el virus no llega a propagarse, y si es muy alto, termina infectando a toda la red en cuestión de poco tiempo. Por tanto, éste es uno de los parámetros principales a ajustar.

Las condiciones iniciales en t_0: **$S(t_0)$**, **$E(t_0)$**, **$I(t_0)$** y **$R(t_0)$** se reflejan en nuestro modelo con los parámetros **φ_S**, **φ_L**, **φ_I** y **φ_R** respectivamente, aunque se obvia el parámetro **φ_S,** puesto que puede ser calculado con **N** - (**φ_L** + **φ_I** + **φ_R**). Estas condiciones también son desconocidas, sin embargo generamos diferentes escenarios para el inicio de la epidemia y en todos ellos asumimos que **$I(t_0)$** > 0 y **$R(t_0)$** = 0, lo que quiere decir que una pequeña fracción de la población está infectada y que ningún individuo se ha recuperado de la gripe aún (la epidemia está comenzando).

Todos los parámetros definidos anteriormente son números reales, excepto **N**, **T**, **Δt** y los **φ**, que se tratan de números naturales (que incluyen el 0). También se ha considerado que el paso de tiempo **Δt** es de un día, puesto que los parámetros **ω** y **ρ** están en días, y tendremos más detalle en nuestro modelo al evolucionar nuestra red de esta forma.

Queda definir el método que se usará para la comparación e interpretación de cada solución. Al tratarse de un modelo de redes aleatorias, es bastante improbable que una solución dada por el modelo coincida exactamente con los datos reales. Como solución se ha decidido usar el error cuadrático medio para la comparación de resultados, cuya fórmula es la siguiente:

$$ECM = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2}$$

Donde **x** e **y** son el valor de la simulación y el valor reportado para **N** semanas totales de las que se tiene muestra, respectivamente.

Sin embargo, al utilizar el factor de escala **s** la ecuación queda de la forma siguiente:

$$ECM = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{s}x_i - y_i)^2} \text{ donde } \boldsymbol{s} = \frac{\sum_{i=1}^{N} {x_i}^2}{\sum_{i=1}^{N} x_i y_i}$$

También hay que considerar la conversión del número de infectados por día a infectados por semana. Para la conversión de los 7 valores de cantidad de infectados que tenemos en nuestra simulación, se hace la media aritmética directamente de los siete para cada semana, considerando el resultado como el valor a comparar con los valores reportados.

1.2.5 EL PROGRAMA CALCULADOR DEL AH1N1

Todas las características del modelo descritas en el apartado anterior deben de ser analizadas, desarrolladas y escritas en el programa que denominaremos calculador o simulador. El objetivo de dicho calculador es generar la red correspondiente a través de unos parámetros dados, y a partir de ahí evolucionar la red hasta que termine la simulación. Los parámetros de entrada tendrán unos valores definidos en un archivo que llamaremos ficha del problema, y los resultados se guardarán en un fichero solución.

El fichero solución recoge las estadísticas de la evolución del virus desde la creación de la red hasta el fin de la simulación. Los datos a almacenar son los parámetros iniciales con los que se ejecutó el modelo y el número de personas pertenecientes a cada clase del modelo SEIR para todos los días de la simulación, es decir, el número de personas que se encuentren en el estado de salud susceptible, latente, infectado y recuperado para cada día de la simulación.

Cada problema distinto tendrá su propia ficha del problema y su fichero solución, de esta forma se puede crear fácilmente un control sobre los problemas resueltos y los que queden por resolver, así como facilitar la gestión de ellos en caso de querer ejecutar el calculador en varios equipos simultáneamente.

1.3 COMPUTACIÓN DISTRIBUIDA

1.3.1 INTRODUCCIÓN

Cuando el tiempo de cómputo de una serie de trabajos realizados en una máquina resulta excesivo e inviable, se han de estudiar otras soluciones. Una amplia mayoría de problemas algorítmicos, científicos o matemáticos requieren gran cantidad de operaciones matemáticas, las cuales pueden durar del orden de días, meses o años, cuando son ejecutadas en una sola máquina. Un ejemplo práctico puede ser la previsión del tiempo del día siguiente. Si dicha previsión tarda más de un día en completarse, no sería interesante el uso de dicho algoritmo.

El objetivo principal de toda distribución de trabajos es la reducción del tiempo total de cálculo de nuestro problema. Existen varias formas de distribuir la carga de trabajo que debería de realizar un computador en varios de ellos, sin embargo siempre se busca la eficiencia a la hora de distribuir, a la vez que la integridad, puesto que solamente interesa el uso de un sistema distribuido si éste funciona correctamente. Es decir, que nos da soluciones idénticas a nuestro problema de la misma forma que nos las daría un sistema no distribuido.

Según la amplitud de un problema debemos de optar por varias soluciones. Si un problema tarda 8 años de ejecutarse en una máquina de cuatro núcleos de forma secuencial, no se puede paralelizar solamente entre los núcleos de dicha máquina, puesto que en el mejor caso dicho problema se habría resuelto en 2 años. En este caso se debería de optar por dividir el problema en más equipos, pudiendo obtener el resultado en meses o semanas.

Para la distribución de tareas tenemos como posibles opciones el uso de un sistema ORB o basado en llamadas de procedimientos remotos (RPCs), el cual se vuelve muy dependiente del tipo de problema, lenguaje de programación y de la configuración del sistema. También podemos utilizar un modelo de memoria distribuida con MPI, el cual requiere una configuración compleja, una interconexión eficiente, y resulta poco escalable, puesto que solamente sería viable en equipos de nuestra red.

Dentro de cada máquina se ha considerado la paralelización de nuestro simulador usando los cores internos mediante metodologías de creación de hilos, como pthreads o OpenMP. Analizando las dependencias entre instantes de tiempo y de contactos entre nodos, en ningún momento se obtendría un aprovechamiento máximo de todos los cores de la máquina. Debido a la naturaleza del programa, y a la cantidad de tareas que se tienen que procesar, resulta mucho más eficiente que el programa sea secuencial y se ejecute en un solo core, para poder ejecutar varias instancias de dicho programa en distintos cores con diferentes tareas a la vez.

Puesto que queremos repartir las tareas entre distintas máquinas, nos interesa buscar un modelo de distribución de trabajos que se pueda ampliar sin necesidad de una reconfiguración constante y que la localidad de los equipos no nos importe. También queremos que dicho sistema de distribución sea lo más universal posible, para su uso con varios simuladores o calculadores, y

no solamente para nuestro proyecto en cuestión. Todo apunta a que se debe pensar en un modelo cliente-servidor.

Mediante un sistema cliente-servidor se pueden unir tantos clientes como sean necesarios, independientemente si están en el mismo rack, ciudad o país. Solamente se requiere que el servidor tenga los puertos de escucha abiertos, el programa simulador y las tareas de dicho simulador, para que éste se encargue de distribuir dichos archivos a los clientes. Un cliente solamente debe de tener la dirección del servidor para poder ejecutarse, con la que trabaja de forma continua: Primero pide las tareas y el programa para resolverlas, se resuelve la tarea propuesta y por último se envían los resultados al servidor [22].

El sistema cliente-servidor más conocido para distribución de tareas científicas es el BOINC (Berkeley Open Infrastructure for Network Computing). Dicho sistema ganó gran popularidad con el programa SETI@Home, el que se encargaba de la búsqueda de inteligencia extraterrestre enviando datos de los satélites de Arecibo (Puerto Rico) para poder ser calculados en ordenadores de todo el mundo [23]. Actualmente BOINC tiene alrededor de 45 proyectos públicos activos disponibles en su wiki. No se cuentan los proyectos privados ni aquellos que hayan decidido publicarse en otro lugar. La cantidad de procesamiento de los computadores adheridos a proyectos BOINC superan al supercomputador más potente del mundo.

El uso de un sistema como BOINC está justificado si el proyecto es lo suficientemente grande para que varios usuarios de todo el mundo tengan que aportar su ayuda con el procesamiento de los problemas, esto requiere anunciar el proyecto debidamente y configurar los calculadores para soportar la interfaz de BOINC. Este trabajo puede llevar meses y en nuestro caso no necesitamos tanta capacidad de procesamiento, puesto que el problema del estado de Nueva Esparta es bastante reducido.

1.3.2 EQUIPOS UTILIZADOS

Para llevar a cabo el cálculo del problema propuesto, se ha tenido a disposición un sistema de clústeres del Instituto de Matemática Multidisciplinar de la Universidad Politécnica de Valencia para la ejecución de todas las pruebas de nuestro estudio. Dicho rack se compone de 15 equipos tipo clúster, que tienen el nombre de Neos, y un equipo adicional con el nombre de Frenchi, que tiene el papel de servidor de Sísifo, el sistema de distribución utilizado en estos equipos. Los equipos tipo clúster fueron cedidos gracias a Javier Villanueva Oller (CES Felipe II, Madrid)

Los equipos tipo clúster y Frenchi cuentan con un procesador Intel Xeon de 4 cores a 2.8GHz, con un total de 4GB de RAM. Todos los Neos tienen instalados una copia de Windows 7 y el servidor Frenchi una distribución Debian de Linux. En los computadores se ejecutan clientes BOINC y Sísifo, teniendo prioridad estos últimos para la ejecución de aplicaciones distribuidas.

Rack con los equipos en el IMM de la UPV

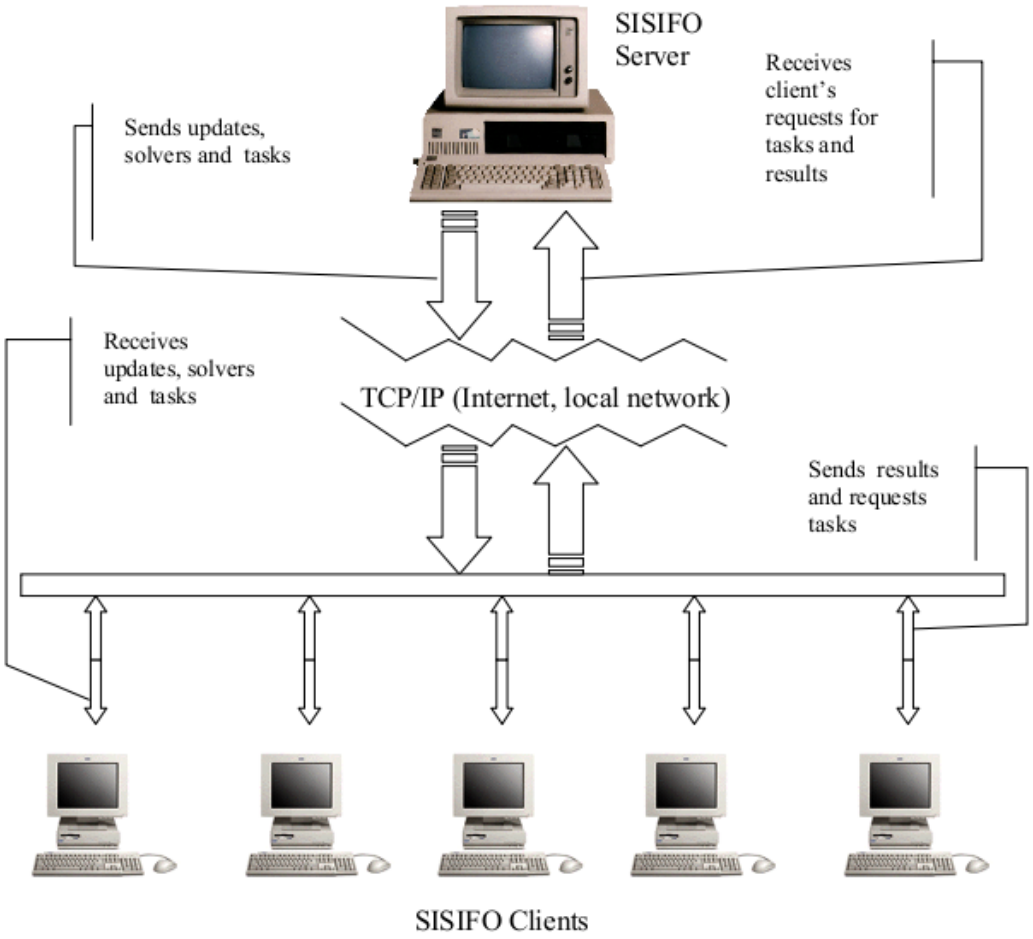
1.3.3 SISTEMA DISTRIBUIDO EMPLEADO: SÍSIFO

Al considerar BOINC demasiado grande para nuestras necesidades, se ha optado por utilizar un sistema propietario, el cual tiene la misma función que BOINC y cuya adaptación al simulador del virus AH1N1 resulta más sencilla. El programa de distribución utilizado se denomina Sísifo y fue creado por Javier Villanueva Oller (Universidad Complutense de Madrid, C.E.S. Felipe II) y Rafael J. Villanueva Micó (Instituto Matemático Multidisciplinar, Universidad Politécnica de Valencia). Dicho sistema es multiplataforma (Windows y sistemas basados en Unix) y está pensado específicamente para la distribución de aplicaciones científicas.

Dicho sistema es el ideal para poder distribuir los trabajos de nuestro modelo del AH1N1 en el rack disponible dentro del Instituto Matemático Multidisciplinar (IMM) y en otros equipos adicionales, en caso que se requiera un poco más de potencia de procesamiento. En el Anexo A se incluye la documentación necesaria para el funcionamiento y manejo del sistema de aplicaciones distribuidas Sísifo. Dicha documentación también incluye los estándares de la entrada y salida de los simuladores que deben cumplir para su correcta adaptación con Sísifo.

Con la configuración de este rack tenemos 15 máquinas de 4 núcleos cada una, lo que nos permite ejecutar 60 calculadores de forma simultánea. De esta forma reduciremos nuestro tiempo total de cómputo entre 60 aproximadamente, sin contar los retrasos producidos por el envío y recepción de tareas, que consideraremos casi despreciables.

Sísifo añade datos de ejecución en las soluciones de las tareas que nos servirán para calcular estadísticas de forma sencilla en nuestro apartado de resultados. Dichos datos incluyen el número de días totales simulados en la tarea del calculador y el tiempo de ejecución en segundos de nuestro calculador.



Esquema de funcionamiento de Sísifo

1.4 PROPÓSITO DE ESTE PROYECTO

1.4.1 PROBLEMAS PARA AJUSTAR MODELOS DE REDES CON DATOS REALES Y POBLACIONES GRANDES

Cuando se desean conocer los parámetros que caracterizan el modelo de una enfermedad, surgen varias dificultades que deben solucionarse. En un principio, no todos los parámetros son desconocidos, lo que nos ayuda a delimitar el problema. El problema se simplifica cuanto menor sea el número de parámetros a ajustar o determinar. No obstante, conocer un posible rango bastante delimitado de un parámetro o posibles valores cercanos también agiliza el ajuste.

Otro problema es la exactitud de los datos reportados. No se sabe con absoluta fiabilidad si las fuentes que contabilizaron las infecciones del virus AH1N1 fueron totalmente correctas. Anteriormente comentamos la posibilidad de algunos individuos infectados por el virus que deciden no ir al médico o cuya sintomatología era menos acusada. En este caso se decide usar un parámetro factor de escala (**r**). Tampoco se espera que nuestro modelo devuelva soluciones idénticas a los datos reportados, pero los resultados se deben poder justificar correctamente mediante intervalos de confianza con varias ejecuciones del simulador.

Adicionalmente, cabe incluir el tamaño de la población como el determinante del tiempo de ejecución del programa. Analizando el algoritmo, el coste computacional es de $(\mathbf{T}/\mathbf{\Delta t})\cdot\mathbf{N}\cdot\mathbf{k}$, con lo que depende linealmente del tamaño de la población y del grado de contactos. En cuanto a coste espacial, el más significativo depende de $\mathbf{N}\cdot\mathbf{k}$, ya que se trata de una estructura que almacena el estado de todos los nodos de la red y sus contactos. Por tanto, una red con demasiados nodos podría hacer que la máquina se quede sin memoria y que la simulación sea imposible de realizarse.

1.4.2 BARRIDO DE PARÁMETROS

Una vez conocemos cuales son los parámetros hemos de ajustar, debemos de generar fichas o problemas para que sean resueltas por el calculador y posteriormente sus resultados sean comparados con los datos reportados. Para cada parámetro tendremos pensado un rango posible en el que sabemos que está incluido el valor óptimo del ajuste.

Al tratarse de un simulador nuevo sin información de modelos anteriores, un posible barrido inicial puede proceder de la siguiente manera: Los parámetros de los que no tenemos ninguna información son **β** y **k**. Al tratarse **β** de una probabilidad, el rango a barrer será de 0 a 1. Sin embargo barreremos valores pequeños, porque si **β** es grande, como por ejemplo **β** = 0.5, la población se infectaría completamente en pocos días aunque **k** sea pequeña. en el caso de **k**, iremos probando valores entre 0 y 100, con valores altos puede bloquearse la máquina por falta de memoria, aparte que la población se infectaría en pocas iteraciones. Los parámetros **ω** y **ρ** pueden dejarse en los primeros barridos a sus valores medios estudiados para reducir el número

de pruebas resultantes. El hecho es inicializar el proceso de barridos sin dejarnos ningún rango por analizar.

Los barridos comienzan por distancias grandes entre valores, por ejemplo, para la **β**, generar 100 pruebas entre 0 y 0.5, por ejemplo, con lo que tendremos una distancia de 0.005 entre pruebas. Una vez se realizan las simulaciones y se comparan los resultados, se puede observar cuales pruebas han dado mejores resultados. Con los mejores resultados se pueden calcular nuevos subrangos, más limitados que en el barrido anterior, y así hacer un ajuste más fino. El objetivo es ir acotando los rangos entre barridos hasta encontrar aquel que sea suficientemente válido para nuestro ajuste.

Después de afinar los parámetros principales pueden entrar en juego otros parámetros que hemos dejado estáticos anteriormente para un ajuste más fino de ellos, como **ω**, **ρ** y $\boldsymbol{\varphi_L}$. Hay que tener en cuenta que al tratarse de redes aleatorias, hacer un ajuste muy fino de un parámetro no tiene siempre el resultado deseado. Incluso si se ejecuta una misma ficha con los mismos valores de parámetros varias veces devolverá soluciones cuantitativamente distintas.

Se observa que este método de ajuste es bastante rudimentario y requiere tiempo de análisis entre barridos para poder escoger los nuevos rangos. También se desperdicia mucho tiempo de cálculo, puesto que se calculan muchas pruebas que no se acercan al ajuste y hasta que no se hayan solucionado todas las pruebas de un barrido, no se pueden analizar.

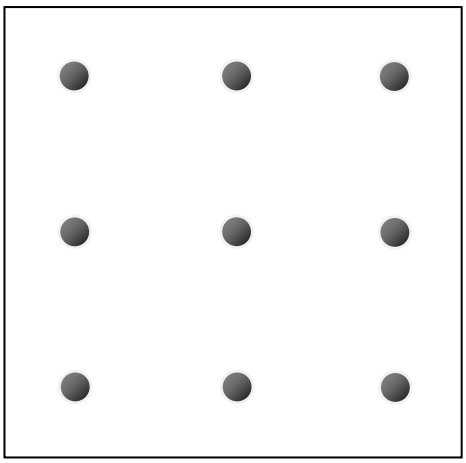
1.4.3 Algoritmo P.S.O. (Particle Swarm Optimization)

Una solución a los problemas relacionados a los barridos de parámetros es el uso de algoritmos de autoguiado. El propósito de estos algoritmos es reducir el número de pruebas totales que se generan en el ajuste y poder determinar automáticamente el mejor ajuste del modelo con la menor interacción posible del usuario y en menor tiempo.

Uno de estos algoritmos es el PSO (Particle Swarm Optimization), el cual actúa de forma similar al vuelo de una bandada de pájaros. Inicialmente se diseñó el algoritmo para simular el comportamiento de la sociedad [24]. Se trata de un método metaheurístico de inteligencia artificial que no garantiza encontrar una solución óptima global, pero los resultados obtenidos suelen ser satisfactorios.

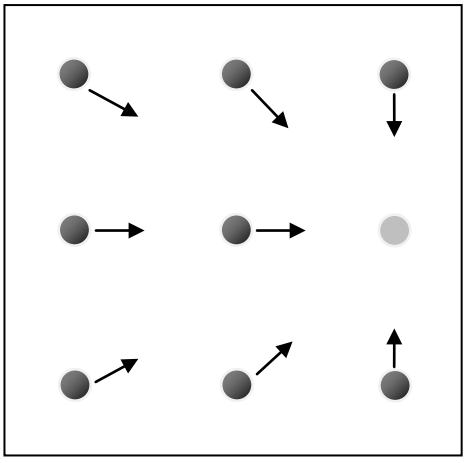
El funcionamiento es el siguiente: se elige un número de procesos o pájaros que forman nuestro algoritmo. Cada pájaro almacena dos problemas y resultados de nuestro calculador: el mejor resultado obtenido por el mismo pájaro y el último resultado calculado por el pájaro. Cada problema del pájaro contiene los parámetros variables que queremos ajustar y unas velocidades de vuelo para cada uno de estos parámetros. Inicialmente, todos los pájaros tendrán inicializados los valores de los parámetros y las velocidades con un valor aleatorio dentro del rango del que

buscamos nuestro ajuste. En la siguiente ilustración asumimos que los parámetros a ajustar por pájaro son solamente dos, para poder mostrarlo en un plano de dos dimensiones:

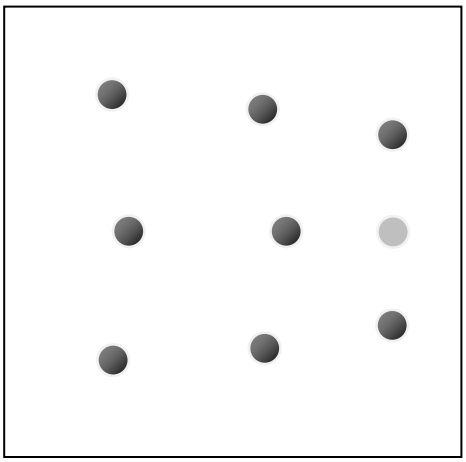


En la figura se encuentran los 9 pájaros distribuidos de forma regular en el espacio de búsqueda de los parámetros **x** e **y**, correspondientes a cada eje, pero se ha mencionado que en un principio la posición de cada pájaro es aleatoria. Cada parámetro también tiene consigo una velocidad, que inicialmente se asigna a un valor aleatorio.

Una vez se tienen todos los pájaros con los valores de los parámetros definidos, se ejecutan las pruebas de estos pájaros en el calculador correspondiente, dándonos unos resultados. Dichos resultados se interpretan y se les da una valoración numérica, para poder comparar como de buena es la solución respecto a las demás. En nuestro caso particular usamos el error cuadrático medio de la solución generada con la muestra de infectados de Nueva Esparta. Posteriormente se comprueba cual de todos los pájaros ha tenido el mejor resultado y éste se considerará el mejor de todos. Pongamos que en nuestro ejemplo es el derecho.



Todos los pájaros almacenan también su mejor solución local obtenida hasta el momento. A partir del mejor pájaro de todos y con la tasa de exploración, haremos que el valor de los parámetros de pájaros se acerquen a los del mejor. Esta tasa de exploración indica en cada iteración cuanto tienen que acercarse todos los pájaros al mejor de todos. Al tratarse de la primera iteración del algoritmo, solo utilizaremos esta tasa para el cálculo de la siguiente posición de los parámetros.



Una vez se tienen los nuevos valores de los parámetros se vuelve a lanzar el calculador y se obtiene la valoración de las pruebas de los nuevos pájaros. Cada pájaro comprueba si su nueva prueba es mejor que la anterior, si es así se guarda en la mejor solución local obtenida. También se actualiza el mejor resultado global como se ha hecho en la primera iteración del algoritmo.

Una vez tenemos los mejores individuales y el mejor global actualizado podemos calcular los nuevos parámetros para los pájaros usando la tasa de exploración, como hemos hecho antes. Sin embargo a partir de esta iteración se ha de agregar la información referente a la tasa de explotación, es decir, aquella tasa que se centra en el mejor resultado local y muestra su comportamiento autónomo. El pájaro tiene su mejor resultado hasta el momento y el resultado que acaba de calcular en la misma iteración. Con estos dos resultados y la tasa de explotación se obtienen las nuevas posiciones de los pájaros. A partir de este momento, el algoritmo se cierra en un bucle y los pájaros irán encontrando mejores soluciones.

Para el cálculo de las nuevas posiciones de los parámetros, primero se actualizan las velocidades de cada parámetro a partir del valor de velocidad anterior y los cálculos relacionados a la exploración y explotación. En el pseudocódigo siguiente se puede observar con detalle dicho proceso:

```
// Comprobamos si hemos mejorado localmente el resultado
ActualizaMejoresPájarosIndividuales();
// Se almacena en una variable el pájaro con el mejor resultado
MejorPájaroGlobal =  BuscaMejorPajaro();
Para cada p = pájaro de los recién ejecutados
       Para cada param = parámetro que queramos ajustar
              // El valor de explotación es nulo si el pájaro mejoró el resultado
              // anterior.
              explotación = tasaExplotación*
                            (p[param].mejorValorIndividual - p[param].valorActual)
              exploración = tasaExploración*
                     (MejorPájaroGlobal[param].mejorValorIndividual -
                     p[param].valorActual)
              // Calculamos la nueva velocidad del parámetro
              p[param].nuevaVelocidad = p[param].velocidad +
                                          explotación + exploración;

              // Calculamos el nuevo valor del parámetro
              p[param].nuevoValor = p[param].valorActual + p[param].nuevaVelocidad;
       finPara
finPara
ejecutaPruebasConNuevosValores();
```

Pseudocódigo del bucle principal del algoritmo PSO

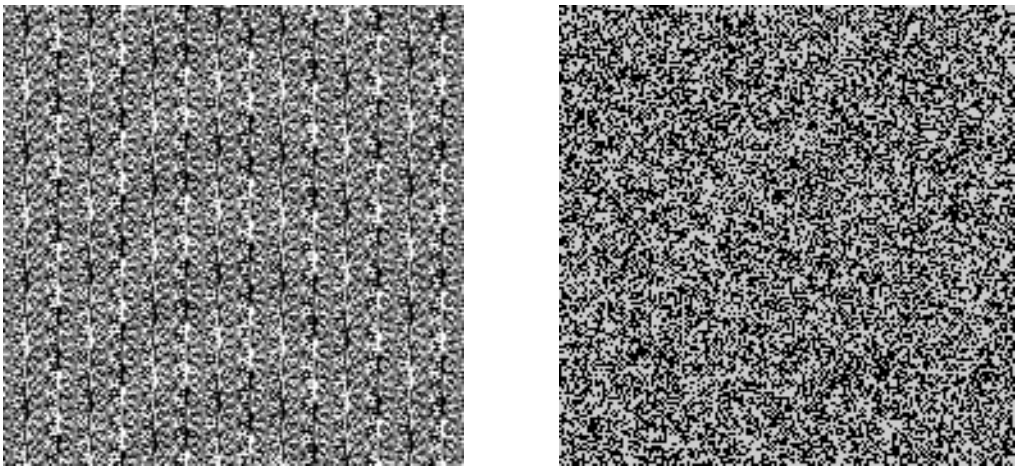
Además de las tasas de explotación y exploración ha decidido incluirse una tasa de aleatoriedad que nos puede ser útil para variar ligeramente las posiciones nuevas de los parámetros. Una pequeña aleatoriedad puede sacarnos de un bucle en el que un pájaro se sitúe solamente en dos posiciones. Si no se desea el uso de cualquier tasa, se declara a 0 y ésta no interaccionará en el algoritmo.

Particularmente este algoritmo nos puede facilitar el trabajo en el ajuste del virus AH1N1 en el estado de Nueva Esparta. Necesitaríamos inicialmente un barrido general para saber donde debemos de enfocar el algoritmo PSO. También hemos de definir las tasas de exploración y explotación según nuestras necesidades. Un valor alto de exploración respecto a explotación encontrará rápidamente soluciones interesantes, en el caso contrario, cuando la explotación es mayor, nos cercioramos de encontrar soluciones más óptimas si tenemos el tiempo suficiente. La estrategia y ajuste de tasas se cambia constantemente según los requerimientos y estado del problema.

1.4.4 Aleatoriedad intrínseca del modelo. Distribuciones probabilísticas (Poisson)

Al tratarse de un modelo de redes aleatorio, la generación de números intermedios debe de tomarse con especial atención. Una mala opción sería el uso de las librerías estándar de C o C++ para la generación de números aleatorios, puesto que está demostrado que sigue un patrón constante y en general es pseudoaleatorio. Este comportamiento es notable cuando se rellena un

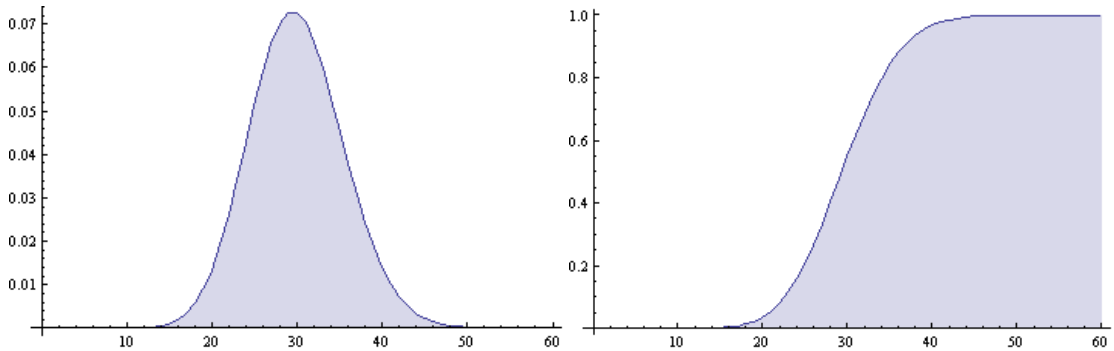
mapa de bits con los números generados por rand(). En nuestro caso se ha usado un rápido generador de números aleatorios de Tausworthe, cuya longitud del periodo es aproximadamente 2^88 y supera todos los test estadísticos [25] [26].



A la izquierda, una imagen generada con rand() de C [27]. *A la derecha con Tausworthe* [28].

La semilla del generador de números aleatorios se ha dejado a la elección del usuario, en el caso que se quiera recrear una simulación anterior. El usuario también puede dejar la semilla a un valor nulo, para que el simulador tome el tiempo del equipo encendido como referencia.

La distribución de Poisson se conoce como la distribución de los fenómenos raros. Ésta expresa la probabilidad que ocurra un determinado número de eventos durante un cierto periodo de tiempo [29]. Con ella se pueden modelar varios tipos de eventos, como el número de llamadas de teléfono en una centralita [30], los goles marcados en un partido de fútbol [31] o el número de documentos solicitados dentro de un servidor web [32].



A la izquierda una distribución de Poisson de media 30. A la derecha la misma pero acumulada.

El generador de números aleatorios se usa en conjunto con una distribución de Poisson acumulada para el cálculo de las probabilidades de salto entre estados de salud en nuestro modelo del virus AH1N1. Concretamente el paso de latente a infectado, y de infectado a

recuperado. También se usa para calcular el número de relaciones con otros nodos que tienen los recién nacidos a partir del parámetro **k**.

El generador de números aleatorios de Tausworthe y el creador de distribuciones de Poisson pertenecen a la librería propia del IMM de la UPV cedida por Javier Villanueva Oller.

2. EL MODELO DEL AH1N1

2.1 ANÁLISIS

2.1.1 INTRODUCCIÓN

En la introducción hemos explicado el modelo de forma comportamental, en este apartado deduciremos las estructuras de datos y algoritmos a utilizar.

En un principio necesitaremos una estructura que contenga los datos relevantes de una persona del modelo. Nos interesa saber el estado de salud de la persona, el tiempo que lleva en ese estado de salud y con qué otros habitantes se relaciona. Para representar los **N** nodos, necesitamos un vector o una lista que contenga todas las personas o nodos del modelo.

También hay que crear una situación inicial en la que los habitantes de la red se reflejen con la configuración de los parámetros φ de la ficha. Al tratarse de una red dinámica, hay que tener en cuenta que se deben poder eliminar y añadir nodos y aristas de la red con la menor penalización posible. La elección de los nodos que van a eliminarse de la red ha de ser totalmente aleatoria.

Se han usado librerías externas para el cálculo de las distribuciones de Poisson, generación de números aleatorios de Tausworthe, conversión de tipos, manejo de ficheros y funciones relacionadas con el reloj del sistema. Estas librerías fueron creadas por Javier Villanueva y son útiles para desarrollar aplicaciones científicas.

2.1.2 PSEUDOCÓDIGO DETALLADO DEL MODELO EN PASOS

Éste pseudocódigo se encuentra a un nivel más cercano al código final, tomando nota de todos los procedimientos necesarios para su ejecución, así como el uso de algoritmos y estructuras de datos necesarios para nuestro diseño. Para cada realización de la prueba, se reinicializa el modelo.

Inicialización:

- Inicializar estructuras de datos generales del modelo con los parámetros de la ficha.
- Creamos las distribuciones acumuladas de Poisson de media **ω**, **ρ** y **k**. Serán tablas acumuladas que corresponderán cada uno de los valores a los días que se llevan en un estado de salud en el caso de **ω** y **ρ**. En el caso de la **k** se refleja en la tabla el número de aristas correspondientes a un nodo recién nacido junto con su probabilidad.
- Inicializamos el generador de números aleatorios con una semilla indicada por el usuario o tomamos el tiempo en segundos desde que se inició la máquina.

- Primero se crea una lista de **N** nodos, a estos nodos se les inicializa su estado de salud e indicamos que el tiempo que llevan en ese estado de salud es de 0 días**.**
- Se inicializan a 0 las variables correspondientes al número de nacimientos y muertes que toca por día. Esta variable acumula los números decimales entre días de simulación.

Evolución:

- Para cada día de la simulación = 1 hasta **T**

- Se almacena el estado actual de la red. El número de susceptibles, latentes, infectados y recuperados en este día.
- Indicamos a Sísifo que el programa sigue funcionando (forma parte de los requerimientos para Sísifo).
- Comprobamos si quedan latentes o infectados en la red. Si no quedan, se acaba el bucle y la simulación.
- Sumamos al acumulado las muertes correspondientes al día actual de la forma:
  `muertesAcumuladas += (d / 1000)*numHabitantesVivos/365`
  Siendo **d** el parámetro correspondiente a la tasa de mortalidad.
- ```
  Mientras muertesAcumuladas >= 1
     matarNodoAleatoriamente()
     muertesAcumuladas -= 1
  FinMientras
  ```
- Sumamos al acumulado los nacimientos correspondientes al día actual de la forma:
  `nacimientosAcumulados += (μ/1000)*num_habitantes_vivos/365`
  Siendo **μ** el parámetro correspondiente a la tasa de natalidad.
- ```
  Mientras nacimientosAcumulados >= 1
     naceNuevoNodoSusceptible()
     numRel = calculaNumRelaciones() // Se obtienen de Poisson de media k
     relacionaNodosAleatoriamente(numRel)
     nacimientosAcumulados -= 1
  FinMientras
  ```
- Para cada nodo **n** de la red que no esté recuperado:
  - Se le aumenta en un día el tiempo del estado de salud.
  - Si es susceptible:
    - Comprueba todos sus vecinos que tengan estado infeccioso.
    - Para cada vecino se genera un número aleatorio entre 0 y 1, si el número generado es menor que el parámetro **β** el nodo **n** pasa al estado de salud latente y cambia su tiempo en el estado de salud actual a 0.
  - Si es latente:
    - Obtenemos el término de la distribución de Poisson de media **ω** correspondiente al tiempo que lleva en ese estado de salud. Se genera un número aleatorio entre 0 y 1. Si el número es menor que el término extraído de la distribución, se inicializa el tiempo de estado de salud a 0 y el nuevo de estado de salud del nodo es infectado.

- Si es infectado:
  - Se obtiene el término de Poisson de media **ρ** correspondiente al tiempo en días que lleva en el estado de salud infectado. Se genera un número aleatorio entre 0 y 1. Si el número aleatorio es menor que el término de Poisson obtenido, el nodo pasa al estado de salud recuperado.

Fin de realizaciones

Se hace la media aritmética de los valores de todas las realizaciones. Se escribe el fichero solución.

2.1.2 ENTRADA Y SALIDA DEL MODELO

El calculador lee la ficha del problema, que tiene toda la información y parámetros para crear y evolucionar el modelo y una vez ejecutado el cálculo se guarda como una solución. Tanto la ficha como la solución son ficheros que se podrán indicar como argumentos en el calculador.

El calculador ha de dar un estado detallado de lo que hace en todo momento sin repercutir exageradamente en el rendimiento de la ejecución. También se comunica con Sísifo para poder dar datos de ejecución e indicar que no se ha colgado el programa. Una información más detallada de la preparación y uso del ejecutable se encuentra en el Anexo B "Manual del Calculador A(H1N1)".

2.1.3 REQUISITOS DEL EJECUTABLE

Aparte de las librerías indicadas anteriormente, se plantea que el programa sea multiplataforma. Al ser un programa con un requerimiento de cálculo elevado, se desea que éste se pueda ejecutar en diversos sistemas. En este caso se desea compilar para Windows y Linux, porque son los sistemas utilizados en el Rack del IMM.

Esto nos plantea utilizar herramientas de código abierto y multiplataforma, en este caso hemos tomado Code::Blocks para la edición de código y el compilador de GNU. También se puede compilar el código en otros sistemas, como Mac OS X (con Xcode instalado), Solaris, BSD, etc...

2.2 DISEÑO

2.2.1 METODOLOGÍA DE DESARROLLO

En la parte de diseño se ha utilizado el lenguaje de descripción UML para la definición de las clases y sus relaciones en el modelo, con el fin de facilitar la comprensión de los módulos del proyecto y agilizar el desarrollo del código fuente.

Se han utilizado ciertos patrones de diseño en la escritura de los códigos fuente. El nombre de las clases comienzan siempre por "**C**" y los atributos o variables de cada una empiezan por **m_**, para que se sepa que son miembros de la clase en cuestión.

Cada fichero fuente contiene una cabecera en la que se almacena información y descripción del fichero fuente, el uso que tiene en el programa, requisitos especiales, versiones y cambios. A continuación se muestra un ejemplo:

```
/*
-------------------------------------------------------------------------------

Proyecto          : Simulador de modelo de redes para H1N1
Nombre del archivo: FICHA.H

Instituto de Matemática Multidisciplinar, Universidad Politécnica de Valencia

-------------------------------------------------------------------------------

Descripción: Funciones para procesar los ficheros de entrada y así conocer
las características del problema.

-------------------------------------------------------------------------------
Autor:            Javier Ruiz Baragaño
Fecha inicio:     01/08/2010
-------------------------------------------------------------------------------
Requisitos especiales: stdafx.h, windows.h, ficha.h, bibliotecas del proyecto

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
R E G I S T R O   D E   C A M B I O S
-------------------------------------------------------------------------------
Cambio número  : 1
Versión actual : 1.0
Autor          : Javier Ruiz
Fecha          : 01/08/2010

Descripción    : Primera versión
-------------------------------------------------------------------------------
Cambio número  : 2
Versión actual : 1.006
Autor          : Javier Ruiz
Fecha          : 05/10/2011

Descripción    : m_k será un long double para añadir ks reales. No afecta
a las simulaciones realizadas anteriormente cuando era int.
-------------------------------------------------------------------------------
*/
```

También se escribe una cabecera para cada función escrita en el código, la cabecera contiene el nombre y la información de las versiones de la función. Describe todos los cambios realizados de la función desde que se creó. Mostramos un ejemplo:

```
/* ------------------------------------------------------------------------
 * Función           : GuardaIdentificador Implementación
 * Paquete           : CFicha
 * Fecha             : 05/08/2010
 * Fecha Versión 1   : 05/08/2010
 *    Autor          : Javier Ruiz
 * Fecha Versión 2   : 05/10/2011
 *    Autor          : Javier Ruiz
 *    Cambios        : Se ha modificado la salida del fichero de acuerdo con
 *                     la nueva especificación de Sísifo
 * ------------------------------------------------------------------------*/
```

Para la documentación del proyecto, se incluyen comentarios siguiendo la sintaxis de Doxygen. Los comentarios cortos de las funciones y de las variables se encuentran en los ficheros cabecera .H y los comentarios grandes en los ficheros de código .CPP. Con Doxygen se ha exportado la documentación del código fuente a LaTeX y a una página web interactiva que facilita la navegación por el código desarrollado.

2.2.2 DESCRIPCIÓN DE CLASES PRINCIPALES

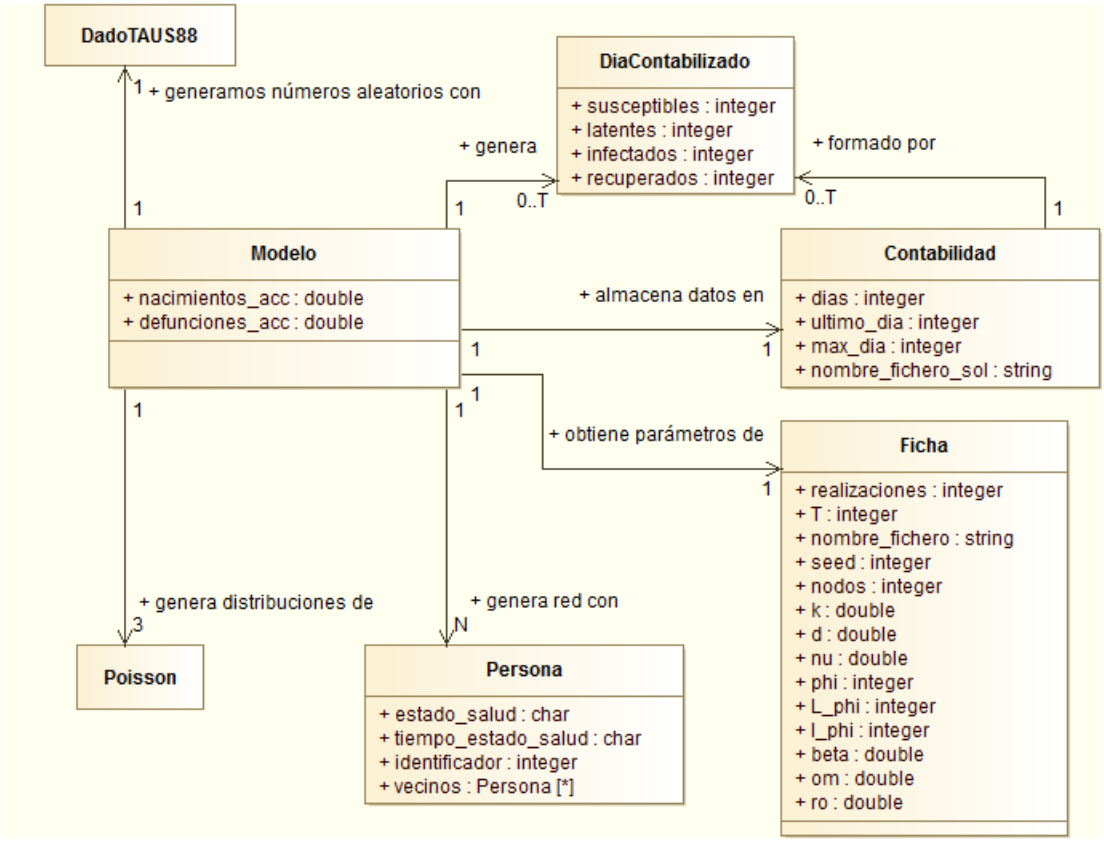


Diagrama de clases del calculador A(H1N1)

En al diagrama se muestran las clases principales y estructuras de datos del calculador. Concretamente, las clases son: Modelo, Contabilidad y Ficha. Las estructuras de datos utilizadas son Persona y DiaContabilizado. Por último se incluyen las clases externas Poisson y DadoTAUS88.

Modelo

Se trata de la clase principal del calculador, es la que tiene el código principal del algoritmo y se encarga de crear y evolucionar la red. Necesita obtener los datos y parámetros del problema a través de la clase Ficha y se almacenan los datos de la simulación con la clase Contabilidad.

Contabilidad

Almacena los datos de la simulación en el fichero .sol correspondiente. También calcula las medias de los valores de las realizaciones en caso de haberse ejecutado el modelo más de una vez.

Ficha

Se encarga de leer el fichero .pro y almacenar el contenido de éste. Una vez cargada la ficha, se usará durante toda la simulación para poder configurar la red a la elección del usuario.

DiaContabilizado

Estructura que almacena el número de susceptibles, latentes, infectados y recuperados de nuestra población en un día de la simulación. Esta estructura se utiliza para pasar la información del día entre clases.

Persona

Representa a cada nodo de nuestro modelo. Contiene la información de su estado de salud, cuánto tiempo lleva en el estado de salud y los habitantes con los que está relacionado.

Poisson

Crea distribuciones acumuladas de Poisson con el parámetro que se le indique. Luego puede consultarse para conocer la probabilidad por cada término de Poisson. Forma parte de la librería del IMM.

DadoTAUS88

Generador de números aleatorios de Tausworthe. Se puede inicializar con una semilla indicada por el usuario o por la predeterminada. Tiene funciones para obtener números en un rango deseado, tanto enteros como flotantes. Forma parte de la librería del IMM.

2.2.3 LIBRERÍAS UTILIZADAS

Aparte de las arriba indicadas (DadoTAUS88 y Poisson) y las librerías estándar de C y C++ se han usado las siguientes librerías adicionales del IMM:

Reloj

Contiene funciones relacionadas con el reloj del equipo. Se puede obtener la fecha y hora actual, medir tiempos de ejecución y calcular el tiempo restante de una operación.

DiskIO

Permite la lectura y escritura de ficheros, mapeado de directorios y otras operaciones de entrada y salida. Su objetivo es abstraerse de los entornos Windows y Linux, ofreciendo una interfaz genérica.

Utiles

Librería que contiene funciones para la conversión de tipos y relacionadas con Sísifo para poder comunicarse con el sistema distribuido.

2.2.4 FUNCIONAMIENTO DE LA APLICACIÓN

Toda la información sobre la configuración de la ficha, funcionamiento del calculador, solución de problemas y emparejamiento con Sísifo se encuentra en el Anexo B "Manual del Calculador A(H1N1)".

2.2.5 DOCUMENTACIÓN DETALLADA

La documentación de los ficheros fuente generada por Doxygen se encuentra en el Anexo D "Documentación del Calculador A(H1N1)".

3. DEMONIO PSO

3.1 ANÁLISIS

3.1.1 INTRODUCCIÓN

La metodología de análisis y de diseño del demonio PSO es idéntica a la usada en el calculador del virus A(H1N1), por lo cual, solamente se describirán las diferencias aplicadas al demonio.

Según hemos visto en el propósito de este proyecto, un pájaro se trata de una instancia que contiene los parámetros relativos a un problema. Cuando se ejecuta el calculador y obtenemos una valoración de la solución decidimos como deben de ser los nuevos valores de los parámetros, tomando como referencia los pájaros que mejor resultado han dado.

Con estos datos deducimos que necesitamos los valores de los parámetros dinámicos en todo momento y para cada pájaro. También necesitamos tener dos listas de pájaros: Una que contiene la iteración actual del algoritmo y otra que contiene la mejor posición individual del pájaro. Por último contaremos con un puntero que nos indique qué pájaro ha tenido la mejor posición global.

Teniendo en cuenta que el algoritmo PSO puede ser utilizado para cualquier tipo de calculador, se ha buscado una manera de valorar la solución de cada problema mediante un intérprete. Este intérprete ha de ser capaz de leer listas y realizar cálculos aritméticos sencillos para poder valorar correctamente cada solución y entregar la puntuación al demonio. Se ha decidido que el intérprete sea un programa externo y que el demonio tenga una interfaz para hacer las llamadas. De esta forma independizamos ambos programas y dejamos la posibilidad de reemplazar el intérprete por otro en un futuro.

Usamos las siguientes librerías externas: **Tausworthe** para la generación de números aleatorios, el **reloj** para estimaciones y medidas de tiempos, **hilos** para la creación de threads simultáneos en Windows y Linux, **utiles** para la conversión de tipos y **tokenizer** para la manipulación avanzada de cadenas. Todas estas librerías se encuentran en la biblioteca del IMM.

3.1.2 PSEUDOCÓDIGO DETALLADO DEL ALGORITMO EN PASOS

Inicialización:

- Inicializamos el generador de números aleatorios.
- Para cada pájaro **p**:
    - Para cada parámetro *dinámico* **x**:
        - `pajaros[p].params[x].valor = RandomEntre(parametro[x].minVal, parametro[x].maxVal)`
        - `pajaros[p].params[x].velocidad = RandomEntre(parametro[x].minVel, parametro[x].maxVel)`
- Lanzamos los pájaros: ejecutamos un problema con los parámetros de cada pájaro.
- Lanzamos el intérprete con la solución obtenida de cada pájaro y guardamos la valoración en `pajaros[p].ECM`
- Tomamos estos pájaros como los mejores individuales en una lista aparte (*elMejorIndividual*), y escogemos el mejor pájaro global (*elMejor*) con el menor valor obtenido de la variable `ECM`.

Evolución:

- Para cada iteración del algoritmo

- Creamos una lista nueva para las siguientes posiciones de los pájaros (`pajarosNuevos`).
- Para cada pájaro **p**:
    - Para cada parámetro *dinámico* **x**:
        - Calculamos la nueva velocidad y valor del pájaro con las tres tasas:
        - `aleatoriedad = tasaAleatoriedad* RandomEntre(parametro[x].minVel, parametro[x].maxVel)`
        - `explotacion = tasaExplotacion* (mejorIndividual[p].params[x].valor - pajaros[p].params[x].valor)`
        - `exploracion = tasaExploracion* (mejorGlobal.params[x].valor - pajaros[p].params[x].valor)`
        - `pajarosNuevos[p].params[x].velocidad = pajaros[p].params[x].velocidad + explotacion + exploracion + aleatoriedad`
        - Comprobar si la velocidad se ha salido de los límites, y si es así, truncar las componentes que se salen.
        - `pajarosNuevos[p].params[x].valor = pajaros[p].params[x].valor + pajaros[p].params[x].velocidad`
        - Comprobar si el valor se ha salido de los límites, y si es así, truncar aquellas componentes fuera de rango.

- `pajarosNuevos` pasa a ser la variable `pajaros`.
- Lanzamos los pájaros: ejecutamos un problema con los parámetros de cada pájaro.
- Lanzamos el intérprete con la solución obtenida de cada pájaro y guardamos la valoración en `pajaros[p].ECM`.
- Comprobamos la variable `ECM` de `elMejorIndividual[p]` y `pajaros[p]`. Si el resultado de la función de `pajaros[p]` es mejor, `elMejorIndividual[p] = pajaros[p]`. Se tiene también un control de las soluciones que se desechan, solo borrándose aquellas con un error superior a `ElMejor.ECM*(1+alpha)`.
- Una vez actualizados los mejores individuales, se guarda el mejor de todos los individuales en la variable `ElMejor`.

Durante el algoritmo se ha considerado que se usa el error cuadrático medio (ECM) como función de valoración de soluciones. Sin embargo, esta función es definida por el usuario en el script del demonio, y por tanto puede utilizarse otra cualquiera.

3.1.3 ENTRADA Y SALIDA DEL DEMONIO

Ficha

El demonio lee la configuración de un fichero que contiene información de tasas, rutas de directorios y los parámetros concretos del calculador que se va a utilizar. Los parámetros dinámicos son aquellos que variarán con el fin de encontrar el mejor ajuste y los estáticos mantendrán el valor indicado en todas las pruebas generadas.

Script

Para la valoración de las soluciones generadas por los calculadores necesitamos una función F que analice la salida del calculador. Dicho valor se considera el error de la solución con la muestra que buscamos en nuestro ajuste, por lo tanto dicho valor es mejor cuanto más pequeño sea. En resumen, el script debe de contener dos cosas: la muestra con la que se quiere comparar la solución del calculador y una función capaz de comparar ambos datos, devolviendo un error numérico.

El intérprete se encarga de leer primero el fichero solución y posteriormente el script para obtener el error numérico. La sintaxis del script guarda una similitud con el formato de Mathematica, puesto que es el programa escogido para el análisis de resultados y ficheros solución.

La configuración de los ficheros de entrada del demonio PSO, así como el manual de uso y algunos ejemplos se encuentra detallados en el Anexo C "Manual del demonio PSO".

Ficheros problema y solución

El demonio guarda los ficheros problema (.pro) y solución (.sol) en las rutas indicadas en la ficha. Sin embargo, solamente guarda aquellos ficheros que son útiles en el ajuste, que en realidad son los que mejor resultado han ofrecido al demonio y aquellos ficheros utilizados en el cálculo de nuevas iteraciones.

De no haber un control de borrado, se colapsaría rápidamente el sistema de archivos. No obstante, se tiene el parámetro **alpha**, con el que escogemos la cantidad de soluciones a guardar según su error numérico. En el mismo directorio del demonio se guardarán listas de aquellos ficheros que se encuentren activos en el algoritmo.

Consola

El demonio indica el estado del algoritmo mediante unos índices estadísticos por consola. La información se actualiza cada iteración, indicando la evolución del algoritmo.

3.2 Diseño

3.2.1 Descripción de clases principales

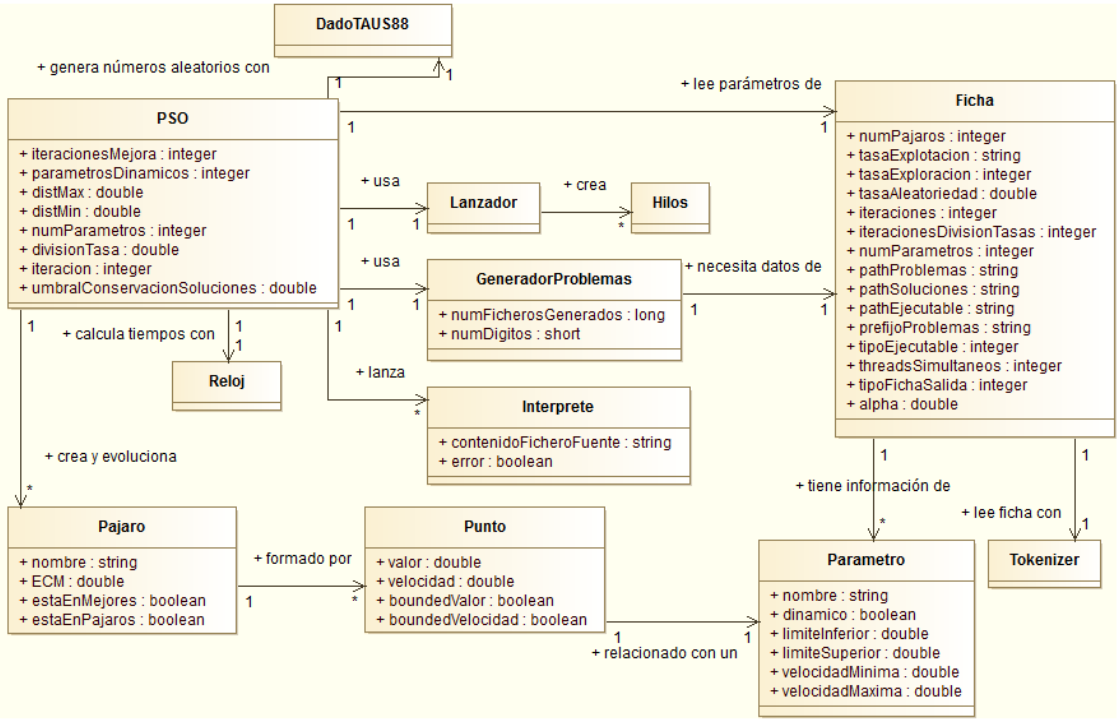


Diagrama de clases del demonio PSO

En al diagrama se muestran las clases principales y estructuras de datos del demonio. Concretamente, las clases son: PSO, Ficha, GeneradorProblemas, Interprete y Lanzador. Las estructuras de datos utilizadas son Pajaro, Punto y Parámetro. Por último se incluyen las clases externas Reloj, Tokenizer, Hilos y DadoTAUS88.

PSO

Se trata de la clase principal que contiene el algoritmo del PSO. Una vez leída la ficha, esta clase se encarga de llamar al resto según el estado del algoritmo. También se encarga de mostrar la información del progreso del algoritmo.

Ficha

Se encarga de leer la ficha con la configuración del demonio y almacenar el contenido de ésta. Una vez cargada la ficha, sus datos serán usados por las clases del demonio.

GeneradorProblemas

La clase crea ficheros .pro a partir de los parámetros de un pájaro para que puedan ser leídos por el calculador. Se tiene un control del número de problemas generados.

Interprete

Es la interfaz del programa con el intérprete externo. Debe de recibir un fichero solución y el script para poder lanzar el intérprete. En el caso de que se desee cambiar de intérprete, solamente se ha de modificar esta clase.

Lanzador

A partir del fichero .pro generado por la clase GeneradorProblemas se lanza el calculador a través de esta clase. Según la configuración del demonio, este lanzador creará hilos o threads locales para la ejecución de los calculadores o los dejará pendientes para su resolución mediante el sistema de aplicaciones distribuidas Sísifo.

Pajaro

Aparte de ser una estructura de datos que contiene la información de la posición del pájaro y el error, también contiene funciones que calculan índices estadísticos para la observación de la evolución del algoritmo.

Punto

Hace referencia al estado actual de un parámetro dinámico de nuestro pájaro. Contiene funciones adicionales para generar estadísticas de ejecución, al igual que con la estructura Pájaro.

Parametro

Representa cada uno de los parámetros del calculador. Existen dos tipos de parámetros, los estáticos y los dinámicos, siendo éstos últimos los que se variarán en el ajuste. Un parámetro estático contendrá toda la información (nombre del parámetro y valor) dentro de la variable nombre.

3.2.2 LIBRERÍAS UTILIZADAS

Aparte de las librerías estándar de C y C++ se han usado las siguientes librerías adicionales del IMM:

DadoTAUS88

Generador de números aleatorios de Tausworthe. Se puede inicializar con una semilla indicada por el usuario o por la predeterminada. Tiene funciones para obtener números en un rango deseado, tanto enteros como flotantes.

Reloj

Contiene funciones relacionadas con el reloj del equipo. Se puede obtener la fecha y hora actual, medir tiempos de ejecución y calcular el tiempo restante de una operación.

Hilos

Biblioteca multiplataforma (Windows y Linux) para la creación y manejo de hilos, con el fin de aprovechar los sistemas multicore.

Tokenizer

Clase útil para el manejo avanzado de cadenas de caracteres. Permite la separación de una cadena en tokens y así tratar los datos individualmente. Se utiliza para leer los datos de la ficha.

DiskIO

Permite la lectura y escritura de ficheros, mapeado de directorios y otras operaciones de entrada y salida. Su objetivo es abstraerse de los entornos Windows y Linux, ofreciendo una interfaz genérica.

Utiles

Librería que contiene funciones para la conversión de tipos y relacionadas con Sísifo para poder comunicarse con el sistema distribuido.

3.2.3 FUNCIONAMIENTO DE LA APLICACIÓN

Toda la información sobre la configuración de la ficha y del script, funcionamiento del demonio y solución de problemas se encuentra en el Anexo C "Manual del demonio PSO".

3.2.4 DOCUMENTACIÓN DETALLADA

La documentación de los ficheros fuente generada por Doxygen se encuentra en el Anexo E "Documentación del Demonio PSO". Aquí se encuentra de forma extendida el diseño real del demonio.

3.3 ADAPTACIÓN DEL CALCULADOR AH1N1 CON EL DEMONIO PSO

3.3.1 FICHA

La ficha del demonio ha de ser preparada con los parámetros del calculador A(H1N1). A continuación se muestra una ficha de ejemplo en la que se intentan ajustar los parámetros **β** y **k** en la región de Nueva Esparta. El número de pruebas necesarias para el ajuste es bastante elevado, por lo tanto se configura para el uso con Sísifo:

```
NÚMERO DE PÁJAROS
200
TASA DE EXPLOTACIÓN
0.03
TASA DE EXPLORACIÓN
0.09
TASA DE ALEATORIEDAD
0.005
NÚMERO DE ITERACIONES
6000
ITERACIONES SIN MEJORA PARA DIVIDIR LA TASA POR 2 (Si la función no mejora
en X iteraciones, dividir las tasas entre dos)
350
NÚMERO DE PARÁMETROS
14
--- PARÁMETROS: LOS DINÁMICOS SON:(nombre; límite inferior; límite superior;
Vmin; Vmax), LOS PARÁMETROS ESTÁTICOS (nombre; valor). OJO EN EL ORDEN.
realizaciones;1
nombre;H1NE
generador;0
t;195
nodos;450138
k;16;24;-0.5;0.5
nu;16
d;16
phi;0
L_phi;0
I_phi;1
beta;0.007;0.015;-0.001;0.001
om;5.0
ro;7.0
CARPETA DONDE ALOJAR LOS PROBLEMAS
./Problemas
CARPETA DONDE SE ANALIZARÁN LAS SOLUCIONES
./Soluciones
EJECUTABLE (En caso que se use el calculador)

FICHERO CON EL SCRIPT DEL INTÉRPRETE, PARA EL CÁLCULO DE LA FUNCIÓN
./scriptH1N1NuevaEsparta.txt
TIPO DE EJECUTABLE (1 si se trata de Sísifo, 2 si se trata de un calculador)
1
PROCESOS SIMULTNEOS DEL CALCULADOR A EJECUTAR
0
FORMATO DE FICHERO .PRO (1 para fracaso escolar o VRS, 2 para H1N1)
```

```
2
ALPHA (LMITE O UMBRAL PARA GUARDAR FICHEROS .PRO Y .SOL)
0.10
```

ficha.pro para el Calculador A(H1N1)

3.3.2 SCRIPT

El método de comparar las soluciones con la muestra es usando el ECM (Error Cuadrático Medio) con el factor de escala como se ha detallado en la introducción. Una de las formas para representarlo en el script es la siguiente:

```
datosMuestra = {2.0, 3.0, 0.0, 0.0, 1.0, 1.0, 3.0, 0.0, 2.0, 1.0, 13.0, 9.0, 8.0,
1.0, 5.0, 2.0, 2.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0};
Infinity = 999999999.0;
sumcuadradosdatos = 0.0;
summuestrapordatos = 0.0;

For[j = 0, j < Length[datosMuestra], j++,
   If[j*7 < Length[resultado],
      valorsemanal = 0;
      For[k=0, k<7, k++, (* Pasamos de días a semanas *)
         If[((j*7) + k + 1) <Length[resultado],
            valorsemanal = valorsemanal + resultado[[(j*7 ) + k + 1]][[3]];
         ];
      ];
      valorsemanal = valorsemanal / 7;
      sumcuadradosdatos = sumcuadradosdatos + valorsemanal^2;
      summuestrapordatos = summuestrapordatos +
                           valorsemanal*datosMuestra[[j + 1]];
   ];
];
If[summuestrapordatos == 0, (* Evitamos la división entre cero *)
  Return[Infinity];
];
r = sumcuadradosdatos / summuestrapordatos; (* Factor de escala *)
ECM = 0.0;
maxDatos = Length[datosMuestra];
If [maxDatos < Length[resultado],
   maxDatos = Length[resultado];
];
For[j = 0, j < maxDatos, j++,
   e = 0;
   valorsemanal = 0;
   For[k=0, k<7, k++, (* Pasamos de días a semanas *)
      If[((j*7) + k + 1) <Length[resultado],
         valorsemanal = valorsemanal + resultado[[(j*7) + k + 1]][[3]];
      ];
   ];
   valorsemanal = valorsemanal / 7;
   If[j < Length[datosMuestra],
      If[j*7 >= Length[resultado],
         e = datosMuestra[[j + 1]];
      ];
   ];
```

```
   If[j >= Length[datosMuestra],
       If[j*7 < Length[resultado],
          e = r*valorsemanal;
       ];
    ];
   If[j < Length[datosMuestra],
      If[j*7 < Length[resultado],
         e = datosMuestra[[j + 1]] - r*valorsemanal;
      ];
   ];
   ECM = ECM + e^2;
];

Return[Sqrt[ECM]]; (* No dividimos entre el número de elementos, no es necesario *)
```

3.3.3 OTRAS CONSIDERACIONES

Con la configuración indicada arriba, se precisa que Sísifo tenga los calculadores compilados en Windows y Linux para que puedan ser enviados a los clientes distribuidos. Sísifo y el demonio han de compartir los mismos directorios de problemas y soluciones, de lo contrario el demonio quedará bloqueado.

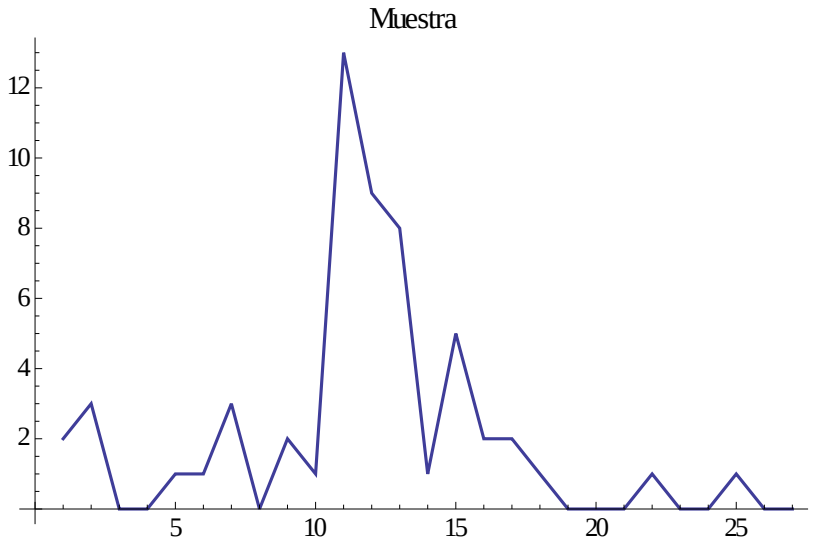
Antes de realizar un ajuste en el entorno distribuido se recomienda ejecutar el demonio en local para comprobar que no existe ningún error en el script y tampoco en la generación de ficheros .pro. Unos valores incorrectos en la ficha pueden causar que el calculador bloquee la máquina y en el caso de ser ejecutado de forma distribuida, paralizar todos los clientes.

4. RESULTADOS

4.1 MUESTRA

Una vez implementado el modelo redes aleatorias del calculador A(H1N1) se procede a su ajuste con los datos reportados del estado venezolano de Nueva Esparta. El caso epidemiológico de Nueva Esparta abarca las semanas 17 a 44 de 2009.

En este apartado se justifica el uso de distintos métodos para la generación de pruebas y obtención de resultados, con el fin de obtener el mejor ajuste.



Gráfica con los infectados reportados por semanas en el estado de Nueva Esparta entre las semanas 17 a 44 de 2009.

El tiempo medio de ejecución de una simulación es de 111,6 segundos contando la sobrecarga del sistema de distribución de problemas y resultados. Este tiempo depende principalmente del número de nodos y el parámetro **k**, propios del estado de Nueva Esparta. Contando con 62 núcleos disponibles en el IMM, se pueden calcular 33 pruebas por minuto, 2000 pruebas por hora o 48.000 pruebas al día.

4.2 BARRIDO

Los primeros ajustes del modelo de redes aleatorias con los datos reportados de Nueva Esparta (Venezuela) fueron realizados con barridos de parámetros. Los parámetros a ajustar son **β** y **k** con los rangos: $0.001 < \beta < 1.0$ y $1 < k < 50$. Los infectados iniciales quedan fijados a 1, los latentes a 0 y los parámetros **nu** y **d** a 17,16. Más adelante se delimitarían más los rangos y se

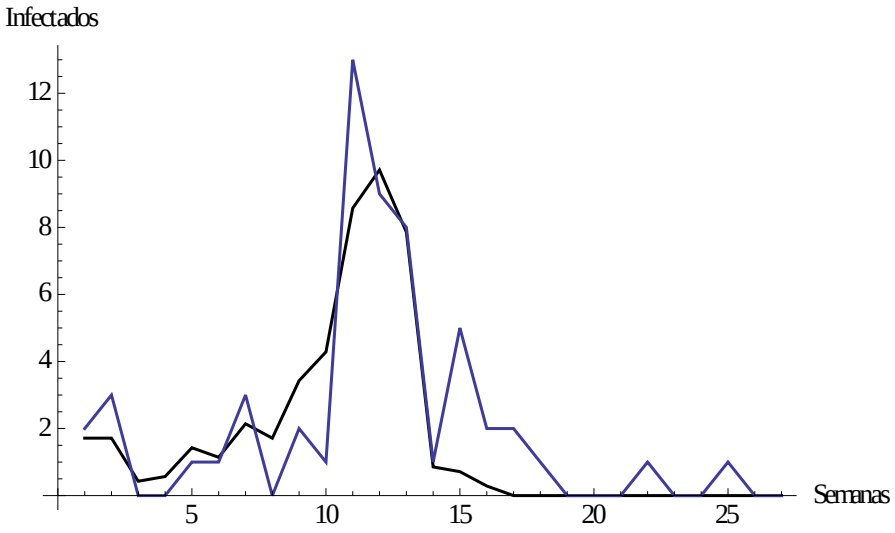
harían barridos de los parámetros **L_phi**, **ρ** y **ω**. Aproximadamente se han ejecutado un total de 480.000 pruebas con todos los barridos para el ajuste.

Una vez lanzadas varias tandas de barridos, cada una de ellas con más resolución con el fin de ajustar los parámetros, obtenemos los resultados y seleccionamos los 10 mejores en la tabla siguiente:

k	nu	d	L_phi	I_phi	beta	om	ro	s	MSE
13	17,16	17,16	3	1	0,014163	5	7	1,0616	1,56788
14	17,16	17,16	1	1	0,014806	5	7	1,11774	1,58128
13	17,16	17,16	4	1	0,014732	5	7	0,967603	1,71285
13,1244	17,16	17,16	0	1	0,013888	4,95446	7,27925	1,54328	1,73488
14	17,16	17,16	4	1	0,013862	5	7	0,855881	1,76113
11	17,16	17,16	1	1	0,013751	5	7	1,05293	1,77462
13,1944	17,16	17,16	0	1	0,013777	4,83175	7,18987	1,12437	1,81192
13,1142	17,16	17,16	0	1	0,013819	5,03037	7,26588	1,26839	1,83757
13	17,16	17,16	5	1	0,01398	5	7	0,936809	1,84052
12,8237	17,16	17,16	0	1	0,013972	4,86795	7,24629	1,01066	1,88991

Tabla con los 10 mejores ajustes del modelo

El primer resultado de la tabla de los mejores ajustes se muestra en la gráfica siguiente:



Mejor ajuste para la ciudad venezolana de Nueva Esparta.

El ajuste no resulta del todo convincente debido al error presente y tampoco se recoge el segundo pico. Se podría continuar realizando más barridos o repitiendo los mismos con el fin de encontrar un resultado mejor, pero esto requiere un tiempo bastante largo y sería ineficiente si podemos contar con algoritmos auto-guiados. El uso del algoritmo PSO queda justificado para ajustar el modelo con los resultados aquí obtenidos.

4.3 PSO

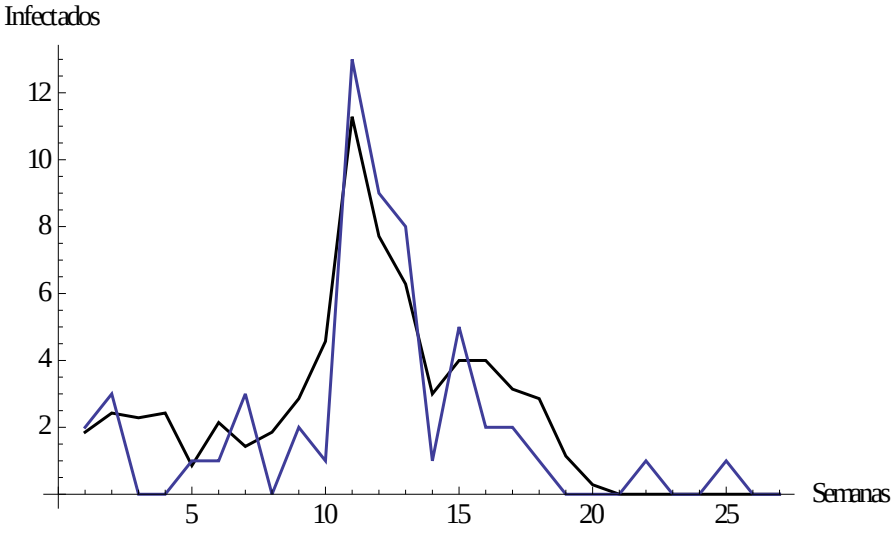
Partiendo de los resultados de los barridos se decidió utilizar el demonio PSO para encontrar un ajuste más fino enfocando el algoritmo de la misma manera que el barrido, primero tomando rangos amplios y más adelante cerrando los intervalos. Con el PSO se han detectado varios puntos distintos donde el ajuste mejoraba.

Adicionalmente se han ajustado los parámetros **L_phi** e **I_phi** que corresponden al número inicial de latentes e infectados respectivamente. **L_phi** ha sido ajustado en un rango entre 0 y 100 e I_phi entre 1 y 100. El objetivo de este ajuste adicional es la suposición de que la muestra que tenemos de Nueva Esparta sea solamente un porcentaje de los infectados reales.

k	nu	d	L_phi	I_phi	beta	om	Ro	s	MSE
21,1504	16	16	3	1	0,00849	5,9	6,66593	0,965375	1,44608
17,1685	16,	16,	2	1	0,012101	4,54818	6,57301	1,2915	1,46626
22,715	17,16	17,16	3	1	0,007748	5,9	6,66593	1,29447	1,48712
20,0275	16,	16,	2	1	0,009274	5,04268	6,4	0,855512	1,50027
17,4494	16,	16,	3	1	0,011686	4,60201	6,49985	0,934011	1,50434
19,9785	16,	16,	2	1	0,010418	4,94966	6,4	1,47961	1,53192
20,1783	16,	16,	7	1	0,008	5,5	6,69356	0,7827	1,56245
22,085	17,16	17,16	3	1	0,007363	5,9	6,66593	1,16283	1,60275
17,794	16,	16,	1	1	0,009464	4,61511	6,53786	0,784468	1,63218
22,355	17,16	17,16	3	1	0,008175	5,9	6,66593	1,05	1,64035

Tabla con los 10 mejores ajustes con el algoritmo PSO.

La gráfica del primer elemento de la tabla es la siguiente:



Mejor ajuste con el algoritmo PSO del estado venezolano de Nueva Esparta. Según el ajuste, el 96,5% de los casos fueron reportados.

5. CONCLUSIONES

Para la modelización de enfermedades epidemiológicas se requieren soluciones complejas que puedan darnos datos fiables en un tiempo limitado. El uso de redes aleatorias sigue este principio, simulando con la precisión deseada, el comportamiento de diversos eventos de nuestra naturaleza.

En este proyecto hemos podido diseñar un simulador de un virus con una evolución y características concretas, que es fácilmente extrapolable o comparable con otro tipo de enfermedades infecciosas. Al seguir el modelo epidemiológico SEIR, este calculador con pocas alteraciones puede ser usado para la simulación de otros virus pandémicos que sigan dicho modelo.

El objetivo principal de este tipo de simuladores, aparte de conocer las características internas de una enfermedad, como su probabilidad de transmisión o resistencia, es también posible simular nuevos escenarios hipotéticos en los que se introduzca un virus determinado en una ciudad o continente y así ver su evolución en esta red. Uno de los puntos más interesantes de los resultados de estas simulaciones es si dicho virus llega a ser transmisible en la red elegida y acaba expandiéndose con más o menos intensidad. O si por el contrario el virus acaba aislándose y desaparece de la red.

Otro tipo de aplicación de este simulador, por ejemplo, es implementar un código en el que se considere la vacuna del virus, conociendo los datos de efectividad e inmunidad de dicha vacuna, y así poder valorar como interaccionaría la aplicación de dicha vacuna a grupos de riesgo o a cierto porcentaje de la población. Con este tipo de simulaciones se pueden realizar estudios que estimen la viabilidad económica de una vacunación masiva del virus o por el contrario una hospitalización de aquellos individuos infectados. Este tipo de aplicación se encuentra ya implementada en el simulador del Virus Respiratorio Sincitial (en inglés RSV), también desarrollado por el Instituto de Matemática Multidisciplinar (IMM) de la UPV.

En este proyecto en concreto hemos podido ajustar el virus en el estado venezolano de Nueva Esparta con el fin de conocer los parámetros propios del virus y de la población. Concretamente se ha ajustado el parámetro de transmisibilidad (**β**), los tiempos medios de paso del estado de latente a infectado (**ω**), de infectado a recuperado (**ρ**) y por último el grado de relación de los habitantes dentro de Nueva Esparta (**k**). Un análisis post-ajuste nos ha permitido mejorar el error considerando un factor de escala (**s**) mediante mínimos cuadrados.

Para poder realizar el ajuste se ha necesitado hacer un barrido de los parámetros indicados aquí arriba. Este método es poco eficiente, puesto que se tienen que generar tandas de muchas pruebas según los rangos y las resoluciones deseadas. Si la resolución no es suficientemente fina puede no encontrarse el ajuste o encontrar uno equivocado. En cambio, si la resolución es muy alta, se pierde bastante tiempo calculando pruebas no interesantes para el ajuste.

Por este motivo se decidió incluir un método heurístico más eficiente con el fin de mejorar el ajuste obtenido de los primeros barridos, utilizando el algoritmo PSO en el ajuste del modelo. Este algoritmo añade cierta inteligencia artificial a la búsqueda del mejor ajuste, pero también incluye el riesgo de no encontrar la solución óptima. Sin embargo, es posible observar a tiempo real la evolución del algoritmo y así interactuar con él para poder rectificarlo.

Con el PSO se han podido obtener mejores soluciones descartando los caminos de aquellas pruebas que no tenían buenos resultados y ahorrándonos el tiempo de ejecución de todas ellas. De esta forma se ha conseguido obtener el ajuste en unos límites de tiempo aceptables con los recursos computacionales disponibles.

Si apareciera una nueva variante del virus más agresiva y se tuvieran que hacer ajustes urgentes, se podrían realizar por medio de sistemas como BOINC y obtener resultados concluyentes en cuestión de pocas semanas o días con las herramientas actuales, tomando mayor tiempo el análisis de los resultados.

ANEXO A: CONFIGURACIÓN DEL SISTEMA SÍSIFO (JAVIER VILLANUEVA)

A.1. COMPONENTES DEL CLIENTE DE SÍSIFO

Sísifo se compone de dos partes: el cliente y el servidor. El cliente (o clientes, son varios) es un programa que está en una carpeta de un PC cualquiera y se conecta al servidor para pedirle un programa ejecutable y un problema, ejecuta el programa sobre el problema, y devuelve la solución al servidor.

Actualmente tenemos de fijo los equipos del cluster Neo con 4 cores y 6 Gigas de RAM

neo01.imm.upv.es
...
neo10.imm.upv.es

mas dos equipos muy modestos con 1 GB de RAM y un solo core.

new01.imm.upv.es
new02.imm.upv.es

Hay otros más (Pio, Frenchi, equipos de la UCM) pero esos los pongo y quito yo dependiendo de las necesidades.

Los equipos ya están configurados y solo hay que ver de cuando en cuando que no están colgados o van mal. Cada PC con 4 cores tiene instalados 4 clientes Sísifo en el puerto 7000, uno en el puerto 7001 y otro en el puerto 7002. Estos clientes están en el programador de tareas y se ejecutan solos cada vez que se reinicia el PC, los puedes ver en el administrador de tareas.

A.2. COMPONENTES DEL SERVIDOR DE SÍSIFO.

El servidor está instalado en

xxxxxxxxxx.upv.es

y se accede a él a través del escritorio remoto usando TightVNC Viewer, que es gratuito y se puede descargar de la web. La clave del escritorio remoto es XXX y la del ordenador es ZZZ.

Frenchi tiene tres servidores distintos instalados, cada uno en una de las siguientes carpetas:

~\Escritorio\Sisifo
~\Escritorio\SisifoNeo7001
~\Escritorio\SisifoNeo7002

Cada servidor funciona independiente de los demás siempre que no les coincidan los puertos TCP/IP donde están funcionando, esto permite tener varios proyectos funcionando a la vez si se desea, eso lo detallaré más adelante. Habitualmente se usa solo el servidor de la carpeta ~\Escritorio\Sisifo.

Dentro de la carpeta del servidor hay un montón de archivos, los más importantes son los siguientes:

Servidor
Monitor
config.cfg
IP_filtradas.txt
.\Ejecutor\exe.exe
.\Ejecutor\exe.lin

A.2.1 ARCHIVO CONFIG.CFG

Tiene este aspecto:

config.cfg

```
PUERTO_DE_ESCUCHA
7000
CADUCIDAD
14400
CARPETA_EJECUTOR_WIN32
Ejecutor/exe.exe
CARPETA_EJECUTOR_LIN32
Ejecutor/exe.lin
RUTA_PROBLEMAS
/home/sisifo/Escritorio/Problemas
RUTA_SOLUCIONES
/home/sisifo/Escritorio/Soluciones
ARCHIVO_REGISTRO
./zlogeventos.txt
ARCHIVO_TIEMPOS
./zlogtiempos.txt
ARCHIVO_INFORME
./Terror.txt
CRC_INCLUIDO_EN_ARCHVO_DE_SOLUCION
1
```

PUERTO_DE_ESCUCHA

Es el puerto TCP/IP por el que espera que se conecten los clientes, es 7000 en el servidor habitual y no hay que modificarlo.

CADUCIDAD

Es el tiempo en segundos que el servidor va a esperar a un cliente para que le devuelva los resultados de un problema que le ha mandado. Depende del problema en cuestión, hay que dejar un buen margen por si hay cortes de red y similares, y que puede que ordenadores del cluster sean más lentos que otros.

CARPETA_EJECUTOR_WIN32

Es la ruta al programa calculador compilado para Win32, depende de cada proyecto y hay que compilarlo adecuadamente.

CARPETA_EJECUTOR_LIN32

Lo mismo para Linux de 32 bits.

RUTA_PROBLEMAS

Es la ruta de la carpeta donde están los archivos de problemas. Los problemas por solucionar tienen extensión .pro, los problemas que están ejecutándose en los clientes y se está pendiente del resultado tienen como extensión .pen y los problemas resueltos y cuya solución está en la carpeta de soluciones tienen como extensión .res

RUTA_SOLUCIONES

Es la ruta de la carpeta donde se van a ir copiando los resultados de cada problema, su extensión es .sol

ARCHIVO_REGISTRO

Es un archivo de texto donde el servidor apunta todos los eventos de ejecución (clientes que se conectan, hora, datos enviados, recibidos, etc). No se toca, solo se usa cuando se depura el programa.

ARCHIVO_TIEMPOS

Es un archivo de texto donde se guarda un registro del tiempo de CPU usado por cada cliente para cada problema. Habitualmente no se usa, ahora guardamos esa información en la propia solución.

ARCHIVO_INFORME

Es un resumen estadístico del tiempo de CPU y unidades procesadas. Lo usa la aplicación "Monitor".

CRC_INCLUIDO_EN_ARCHIVO_DE_SOLUCION

Este parámetro indica el tipo de solución que devuelve el cliente. Es para compatibilidad con proyectos viejos.

0 -> No hay checksum de control en el archivo de solución
1 -> Hay checksum de control binario en el archivo de solución
2 -> Hay checksum de control ASCII en el archivo de solución

Tu proyecto deberá usar el tipo 2 de checksum, más adelante se detalla cómo.

2.2 Archivo IP_filtradas.cfg

Este archivo contiene las direcciones IP de aquellas máquinas que queremos prohibir o permitir acceso al servidor. Habitualmente no se usa, pero es útil si algún cliente se vuelve "loco" y manda todo el rato resultados erróneos, por ejemplo, o si el servidor está en una red pública y quieres dar acceso solo a ciertos PC. Su formato es el siguiente:

EXCLUSIVO_IP
SI/NO
EXCLUSIVO_ID
SI/NO
[IP/ID]XXX.XXX.XXX.XXX
NO/SI

El primer parámetro indica si la lista es de exclusión o de aceptación. Si EXCLUSIVO_IP vale SI, entonces solo las IP's de la lista se tienen en cuenta y se rechazan todas las demás. Si vale NO, entonces solo las IP's de la lista se tienen en cuenta y las demás se aceptan.

El segundo parámetro es lo mismo pero para las ID de cada cliente

De ahí en adelante, el archivo tiene una entrada con el indicador de si el dato es una ID o una IP, y otra línea que indica si se le da acceso o no.

A.2.3 Archivo Ejecutor|exe.exe

Este archivo puede tener una localización y nombre arbitrario, pero lo normal es que esté en la carpeta "Ejecutor", es el ejecutable de Windows que resuelve el problema.

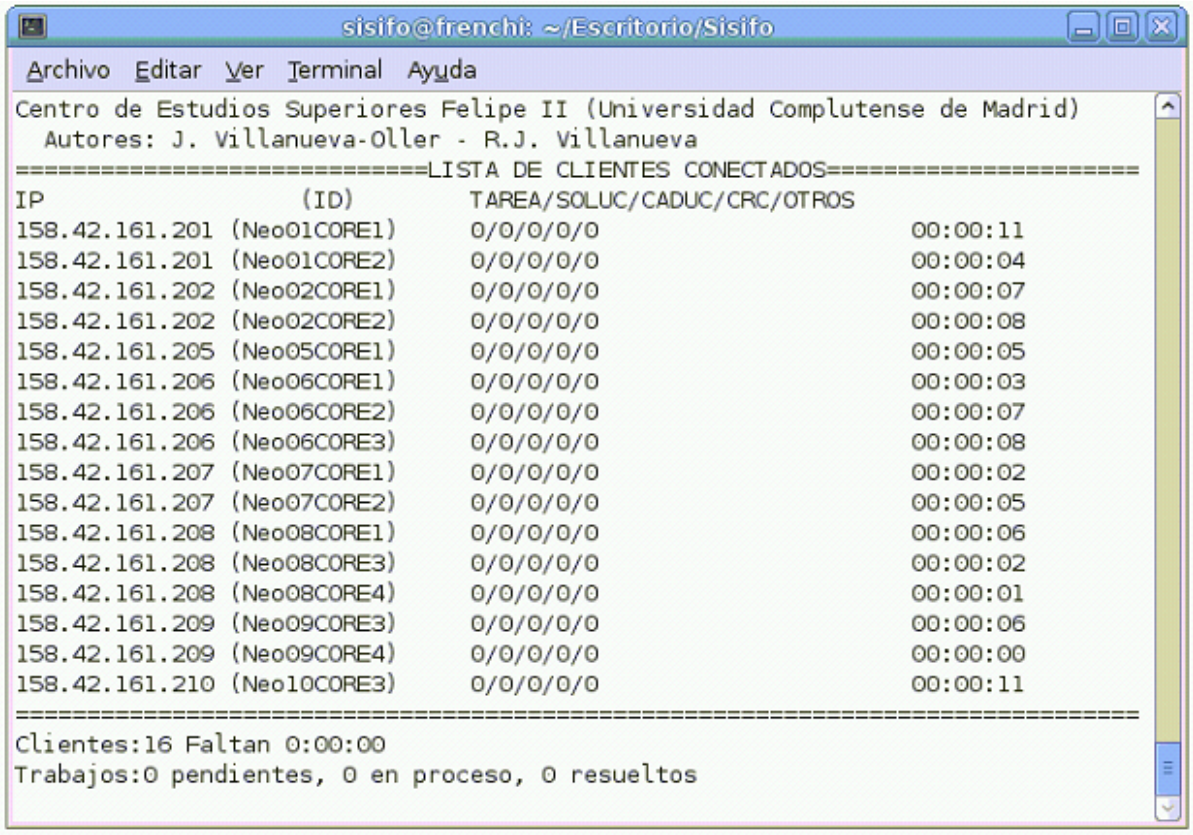
A.2.4 Archivo Ejecutor|exe.lin

Lo mismo para Linux.

A.2.5 ARCHIVO SERVIDOR

Es el ejecutable de Sísifo, se ejecuta desde una ventana terminal, lee el archivo config.cfg y muestra en pantalla la siguiente información:

De izquierda a derecha aparece la IP del cliente, la ID del cliente (puesto que puede haber varios clientes en un solo equipo y compartir IP), la cantidad de tareas enviadas, la cantidad de soluciones válidas devueltas, la cantidad de tareas que le han caducado, los errores de checksum que ha tenido y otros errores generales. El valor de más a la derecha es el tiempo que ha pasado desde la última vez que el cliente se comunicó con el servidor.



A.2.6 ARCHIVO MONITOR

Es un programa auxiliar que muestra en pantalla información sobre tareas pendientes, resueltas, y otras cosas como renombrar archivos de problemas, borrar soluciones, o comprobar la validez de las soluciones. Lo puedes ejecutar y parar a voluntad.

A.3. Reparto de memoria y de CPU

Cada cliente Sísifo ocupa un core de la CPU. A más cores, más clientes se pueden ejecutar simultáneamente. Sin embargo hay que tener en cuenta que la RAM también se comparte, y a veces no es posible que un PC ejecute 4 clientes a la vez si el consumo de RAM de los 4 excede al del ordenador.

Por ello cada PC de Neo tiene la siguiente configuración:

4 clientes al servidor en puerto 7000
1 cliente al servidor en puerto 7001
1 cliente al servidor en puerto 7002

Si el proyecto requiere poca memoria y caben 4 clientes a la vez, se usa el servidor del puerto 7000. Si ocupa mucha memoria y solo cabe un cliente a la vez, se usa el servidor del puerto 7001. El 7002 se usa para complementar al 7001 o para hacer pruebas.

A.4. Consideraciones de implementación

Puesto que Sísifo está pensado para ejecutar distintos tipos de aplicaciones, y además es intercambiable con nuestro otro sistema de cálculo distribuido Falúa, dichas aplicaciones (o proyectos) deben estar de acuerdo a una serie de normas, de forma que todas usen los mismos formatos de ejecución, archivos de entrada y archivos de salida.

A.4.1 Calculador

El proyecto a ejecutar en Sísifo necesita al menos un ejecutable que reciba un archivo con la definición del problema, y devuelva en un archivo la solución a dicho problema.

calcule calculador. este calculador normalmente tiene dos versiones, una para Windows y otra para Linux, si solo hay una entonces los clientes de la otra plataforma no podrán usarse para calcular. Neo usa Windows.

A.4.2 Definición del problema

Los problemas a calcular han de estar definidos en archivos de texto con el nombre

problemaXXXXXX.pro

siendo XXXXXX un número que va del 000000 al 999999.
El contenido de cada archivo de texto ha de ser **exactamente** el siguiente:

```
identificador_de_prueba
Pruebaxxxxxx
consumo_de_RAM
yyyyy
RESTO_DE_DATOS
xxx
```

La segunda línea ha de contener el nombre del problema, que ha de coincidir con el nombre del archivo pero sin la extensión.

La cuarta línea ha de contener la cantidad de RAM (en Kbytes) que va a ocupar la ejecución del problema. Si tus ejecuciones ocupan poco, puedes poner 512000 o una cantidad similar en todas. Si no, para cada problema deberás estimar e indicar en esta línea la RAM que necesitas.

De la quinta para abajo ya depende del problema del que se trate, pero te recomiendo fuertemente que mantengas el hábito de una línea de texto con el nombre del campo, la siguiente para el dato, y así sucesivamente.

A.4.3 Definición de la solución

El programa calculador ha de buscar, al ponerse en marcha, un archivo que tenga como extensión .pro, abrirlo, leerlo, resolverlo y generar una solución.

El archivo de solución generalmente tiene el mismo nombre que el del problema, solo que con la extensión .sol. Dicho archivo se ha de crear en el disco al principio de los cálculos con la palabra "ERROR" guardada en él, y se sobreescribe con la solución de verdad una vez terminas y justo antes de salir del programa. Eso se hace para detectar si el programa se ha interrumpido a medias o se ha quedado colgado, en cuyo caso la solución contendrá la palabra ERROR y se descarta.

La solución ha de contener el siguiente formato:

```
problema={Pruebaxxxxxx,param1,param2,param3,paramN};solucion={...};auxiliares={zzz,yyy};CHECKSUM
```

'problema' es una variable que contiene el identificador de la prueba y luego todos los parámetros que la definen separados por comas. Esto se hace para tener junto a la solución exactamente la información del problema que la generó.

'solucion' es una variable que contiene la solución, depende del problema.

'auxiliaries' es una variable que se usa para guardar información estadística. El primer valor es el número de turnos que se han ejecutado, y el segundo es el número de segundos que el calculador ha tardado en procesar el problema.

'CHECKSUM' es una cadena de 8 caracteres que refleja el checksum de 32 bits del contenido del archivo de solución. Este checksum lo calcula la función InsertaCRCASCII de la clase CRC32 de la biblioteca de funciones de Sísifo.

A.4.4 SISTEMA DE WATCHDOG DEL CLIENTE

El cliente tiene un sistema por el cual espera que el programa calculador le indique periódicamente si sigue vivo o no. Si pasado un tiempo no le responde, el cliente asume que se ha quedado colgado y mata la tarea.

Esto se hace a través de un archivo que se llama watchdog.txt. El calculador **ha de crear** ese archivo periódicamente (con un contenido cualquiera, puede estar vacío) y Sisifo mira si ese archivo existe. Si existe lo borra y vuelve a esperar a que lo creen de nuevo, y si no existe asume que el calculador se ha quedado bloqueado y entonces, como comentaba antes, lo mata.

El calculador puede hacer eso usando una de las funciones de la biblioteca compartida de Sisifo llamada "colea" (por lo de "vivo y coleando") dentro del bucle de simulación, por ejemplo, después de calcular cada turno. Esa función se encarga de crear el archivo sin tener que preocuparse de nada más.

A.5. OTRAS NOTAS

Por compatibilidad futura con Falúa, el programa calculador ha de admitir los siguientes parámetros:

```
calculador param1 param2
```

Si param1 y param2 no existen, entonces se asume como archivo de entrada 'problema.pro' y como archivo de solución el nombre que haya en la ficha de problema con la extensión .sol

Si param1 es "-boinc" o "-BOINC", entonces se asume como archivo de entrada 'problema.pro' y como archivo de solución 'problema.sol'.

Si param1 y param2 existen, entonces el archivo de entrada es param1 y el de salida es param2

Tanto Sísifo como Falúa se encargan de llamar al calculador con los parámetros adecuados dependiendo del caso.

ANEXO B: MANUAL DEL CALCULADOR AH1N1

B.0. INTRODUCCIÓN

Un programa calculador recibe unos datos de entrada con los que dedica un tiempo de cálculo para generar una posible solución. Este tipo de programas suelen ser propios de aplicaciones científicas y cuando se tratan de problemas grandes se requiere que se ejecuten en entornos distribuidos.

En el caso de este calculador, se trata de un modelo de redes aleatorias del virus AH1N1 que sigue el modelo epidemiológico SEIR. Con el simulador se han realizado ajustes en algunas ciudades, como Nueva Esparta (Venezuela) o Bogotá (Colombia). El tiempo de cálculo de una prueba puede durar de unos minutos a horas dependiendo del tamaño de la red y el grado de relación entre los nodos.

El programa puede ejecutarse en entornos Unix y Linux. Para la recompilación del ejecutable se necesita un computador con Windows o Linux, una versión reciente de Codeblocks y la suite de compliadores de GNU (g++, gcc, make, etc...). En el caso de Windows, se debe de descargar el paquete de Codeblocks con Mingw, que también incluye el compilador. En la carpeta de ficheros fuentes hay un fichero de proyecto para Windows y otro para Linux.

El objetivo de este manual es ayudar al usuario con el manejo del calculador del AH1N1 y poder familiarizarse rápidamente con él. Si se encuentra cualquier fallo, error u omisión durante la configuración del programa o en el manual mismo, por favor contactad al autor con el siguiente correo: jaruiba [@] googlemail.com

B.1. PARÁMETROS DE EJECUCIÓN

El programa soporta el sistema de aplicaciones distribuidas Sísifo, por tanto sigue unos patrones y requerimientos para su soporte mediante unas librerías. Ver "Configuración del sistema Sísifo" para más información de sus requerimientos.

Uso: `H1N1.exe [[<problema.pro> <solucion.sol>]|[-boinc]]`

El ejecutable puede ser compilado en Windows y en sistemas Unix, por lo que la extensión del ejecutable puede variar. Si se lanza el demonio sin parámetros de entrada, se da por hecho que la ficha del programa se encuentra en el directorio de trabajo del ejecutable y tiene el nombre de *problema.pro.*

En caso de querer indicar una ficha y un fichero de salida diferente, se pueden indicar ambos como parámetros. Por último, se ha agregado la opción -boinc en caso de que el calculador vaya a usarse con el sistema distribuido BOINC, con el cual intenta leer una ficha con el nombre "problema.pro" y escribe la solución en el fichero "problema.sol".

B.2. FICHERO DE CONFIGURACIÓN

En este apartado indicamos como es la estructura de una ficha de un problema para nuestro calculador. El formato que sigue la ficha es muy similar a la sintaxis de C, la definición de las parámetros son como asignaciones de C y permite comentarios con `//`. A continuación se muestra una ficha de ejemplo del caso de Bogotá (Colombia).

```
// Ficha para el simulador de redes del AH1N1
// ----------------------------------------

// Caso: Bogotá (Colombia)
// Comienzo: Lunes de la semana 17. Fin: Domingo de la semana 44.

realizaciones=1;    // Número de Repeticiones de la prueba.
nombre=Prueba;      // Nombre del fichero de salida (sin extensión).
generador=0;        // Semilla de la red, 0 para basarse en el tiempo actual.
t=195;              // Número de días en los cuales se corre la simulación

  // En este caso son 195 días porque se ejecuta durante 27 semanas y 6 dias.


// Características generales de la población

nodos=7609424;      // Número de nodos del modelo.
k=5;                // Grado de los nodos (a determinar).


// Índices de natalidad y mortalidad

nu=16;              // Índice de natalidad por 1000 habitantes al año.
d=16;               // Índice de mortalidad por 1000 habitantes al año.


// Reparto inicial de la población (en porcentaje sobre los nodos iniciales)

phi=0;       // Recuperados Iniciales
L_phi=0;     // Latentes Iniciales (a determinar)
I_phi=4;     // Infectados Iniciales (a determinar)

  // Los susceptibles se calculan de la forma: nodos-(phi+L_phi+ I_phi)*nodos


// Características del virus y propagación

beta=0.0147318;     // Probabilidad de paso de Susceptible a Latente por
                    // contacto con una persona infecciosa.
```

```
om=5.0;             // Media de días para el paso de Latente a Infeccioso.
ro=7.0;             // Media de días para el paso de Infeccioso a Recuperado.
```

El formato permite un orden de parámetros distinto al que se indica aquí, sin embargo se recomienda mantener un orden agrupado por categorías como se muestra en el ejemplo anterior. A continuación se muestra una tabla con todos los parámetros, el tipo de datos que requiere el parámetro y una descripción.

Parámetro	Tipo	Descripción
realizaciones	entero	Número de repeticiones de la misma prueba a realizar. Todas las soluciones se suman y se dividen entre el número de repeticiones, usando la media aritmética
nombre	cadena	Es el nombre del fichero de salida del problema. No se debe incluir la extensión .sol
generador	entero	Elige la semilla del generador de números aleatorios de Tausworthe. Al asignar un 0 se tomará el tiempo de ejecución en segundos de la máquina desde que se inició
t	entero	Tiempo en días total a simular con el modelo
nodos	entero	El número total de nodos de nuestra red aleatoria
k	real	El grado de relación de los nodos en la red
nu	real	Tasa de natalidad al año (por 1000)
d	real	Tasa de mortalidad al año (por 1000)
phi	entero	Número de recuperados iniciales. Si partimos de un escenario donde no ha aparecido nunca el virus, tendrá que ser 0
L_phi	entero	Número de latentes iniciales
I_phi	entero	Número de infectados iniciales
beta	real	Valor entre 0 y 1 que representa el porcentaje de transmisibilidad del virus en un contacto entre dos personas, en las que una de ellas esté en estado infectado y la otra sea susceptible.
om	real	Valor medio en días en los que pasa un individuo del estado de latente a infectado.
ro	real	Media de días en los que tarda un individuo de pasar del estado de infectado a recuperado.

El programa no muestra ningún mensaje error en caso que se haya saltado un parámetro especificado en la ficha, de modo que el usuario ha de observar cuidadosamente el contenido de la ficha. Puede ser útil basarse en la ficha de ejemplo aquí mostrada. También se incluyen varios ejemplos de fichas junto con el calculador.

B.3. FICHERO SOLUCIÓN

El fichero de salida del calculador tiene la extensión .sol y el nombre indicado en la ficha. La sintaxis que sigue el fichero para almacenar de los resultados es propio de Mathematica, que es el programa con el que se analizarán posteriormente las soluciones. Un ejemplo de fichero solución se incluye con el ejecutable.

En el fichero solución se almacenan tres listas en el orden siguiente. La primera tiene el nombre de **problema**. Esta lista contiene todos los parámetros de ejecución que también están en la ficha. La segunda lista tiene el nombre de **resultado**, con la que se almacenan el número de habitantes por estados de salud para cada día de la simulación. La última lista tiene el nombre de **auxiliares**, y ésta contiene datos estadísticos relacionados al tiempo de ejecución y a los días ejecutados.

Concretamente el formato de la lista "**problema**" es el siguiente:

```
problema = {"<nombreficherosalida.sol>", nodos, k, nu, d, phi,
            L_phi, I_phi, beta, om, ro, semilla_utilizada,
            num_realizaciones, T};
```

Todos las variables en negrita corresponden a los parámetros de la ficha. El nombre del fichero de salida se forma con el parámetro **nombre** de la ficha y la extensión .sol. La semilla utilizada puede ser el valor distinto de cero que se haya indicado en la ficha, o bien, si se ha elegido cero, se almacenará la semilla que utilizó el calculador.

El formato de la lista **resultado** es el siguiente:

```
resultado = { { S_dia1, E_dia1, I_dia1, R_dia1 },
              { S_dia2, E_dia2, I_dia2, R_dia2 }, ... };
```

Se almacena el número de habitantes por estado de salud de la red para cada día de la simulación, S representa a los susceptibles, E a los latentes, I a los infectados y R a los recuperados. Esta lista tiene una longitud variable porque la simulación puede terminar antes de lo previsto, por ejemplo, cuando no hay ni latentes ni infectados en nuestro modelo. Como mucho una simulación puede durar **t** días (depende del parámetro de la ficha).

El formato de auxiliares se recoge también en la configuración de equipos distribuidos de Sísifo, puesto que es obligatorio que tenga esta información en la ficha si se usa con el sistema de aplicaciones distribuidas Sísifo. El formato es el siguiente:

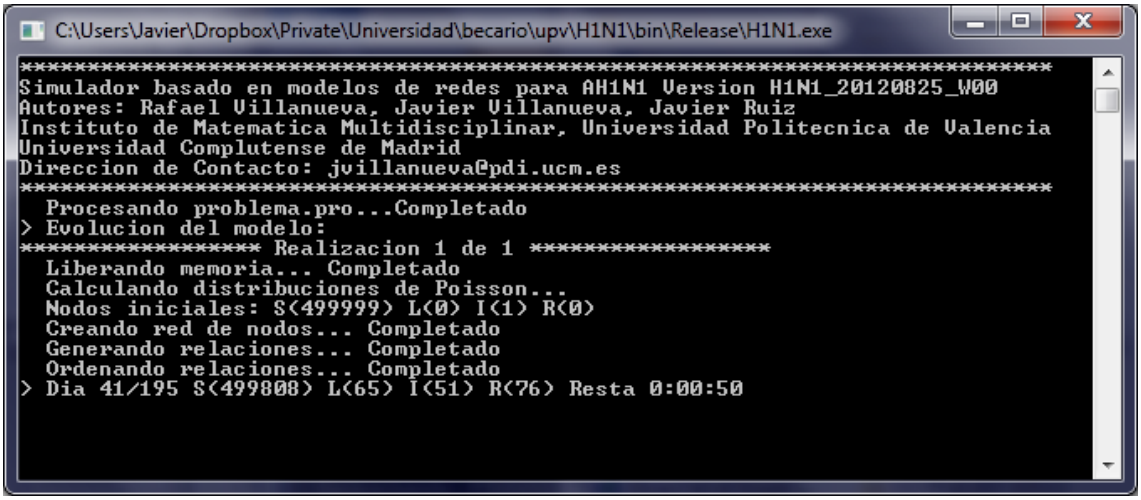
```
auxiliares = { rondas_ejecutadas, segundos_transcurridos };
```

Rondas ejecutadas significa el número de días simulados en total. Segundos transcurridos cuenta el tiempo desde que el programa se abrió hasta que terminó y devolvió los resultados.

Por último, un fichero solución contiene un valor de 8 caracteres hexadecimales al final del fichero que contiene el checksum CRC32 de los datos anteriores a este checksum. Forma parte también de la configuración impuesta por Sísifo para rechazar ficheros solución corruptos.

B.4. INTERFAZ

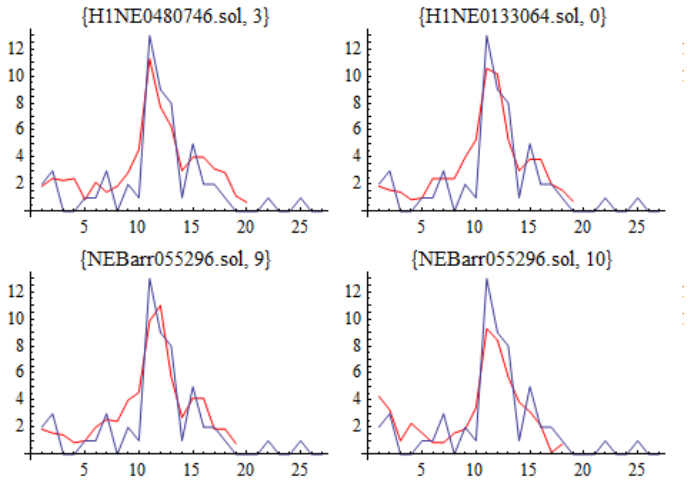
Los mensajes que muestra el calculador informan al usuario del estado actual de la simulación y contiene datos de la ejecución. En esta captura de pantalla se encuentra el calculador en el momento que está evolucionando el virus:



En la evolución se indica el día actual, el número de habitantes por estados de salud y el tiempo restante de la tarea en general. En caso de haber algún problema de ejecución se mostrará por la salida estándar.

B.5. ANÁLISIS DE SOLUCIONES

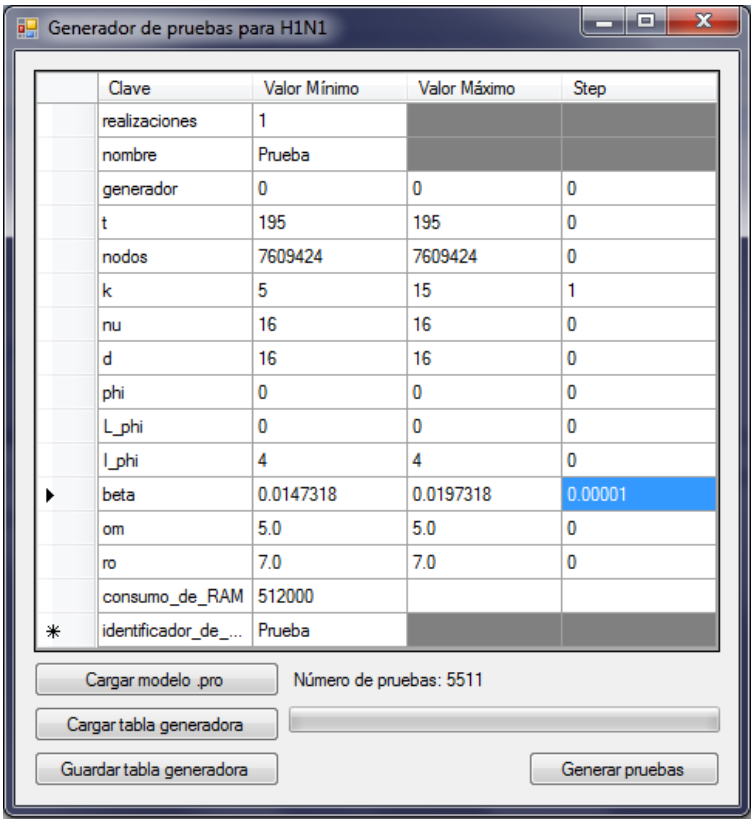
Para el análisis de ficheros solución se han preparado varios ficheros notebook de Mathematica. Con estos ficheros se han podido comprobar aquellas soluciones que mejor ajustaban el modelo a los datos concretos. Con el calculador se incluye también un fichero ejemplo de análisis de resultados para el caso de Nueva Esparta.



Captura de gráficas de Mathematica

B.6. HERRAMIENTAS ADICIONALES

Para la generación de barridos se ha creado un pequeño programa creador de fichas en Visual Basic .NET. El programa cuenta con una interfaz gráfica que facilita la entrada de los datos en la ficha. También se incluye con el programa principal.



El funcionamiento es sencillo. Se carga un fichero .pro del que se quiere tomar los valores de referencia y se modifican los valor mínimos, los máximos y los escalones de cada parámetro. A medida que se vayan modificando los valores el programa indicará el número de pruebas que generará en total. Si algún valor tiene un formato incorrecto se marcará en rojo.

Se pueden guardar y cargar tablas anteriores y por último generar las pruebas. Si es un número grande de pruebas puede tardar bastantes minutos en generarse, depende del sistema de archivos del ordenador.

B.7. SOLUCIÓN DE PROBLEMAS

La mayoría de problemas de ejecución del calculador provienen de la ficha. En ese caso, el programa lo indicará. La ficha debe de contener todos los parámetros indicados en el apartado de "Fichero de configuración", de lo contrario, el comportamiento del programa será inesperado.

Si el programa no se ejecuta en el sistema operativo por no estar compilado el binario para la máquina en cuestión, el proyecto deberá ser recompilado.

ANEXO C: MANUAL DEL DEMONIO PSO

C.0. RESUMEN

Este manual describe el programa tiene integrado el algoritmo PSO (Particle Swarm Optimization) para ayudar en el ajuste de problemas científicos distribuidos. El programa se puede usar de forma local o conjunta con el sistema de aplicaciones distribuidas Sísifo. Algunos calculadores con el que se pueden ejecutar el algoritmo son el modelo de redes del virus AH1N1, el simulador basado en redes del fracaso escolar, el simulador del virus respiratorio sincitial basado en redes (VRS) y cualquier otro calculador cuya entrada y salida cumpla con los requerimientos para Sísifo.

El objetivo de este manual es poder orientar al usuario a configurar el algoritmo para nuevos calculadores o nuevos ajustes de calculadores existentes. En un principio con ficheros ejemplo de ejecuciones anteriores es suficiente para guiarse y crear nuevos, este manual detalla aquellos aspectos que queden en ambigüedad. En caso de encontrar algún fallo, errata u omisión durante la configuración del programa, por favor contactad al autor con el siguiente correo: jaruiba [@] googlemail.com

C.1. PARÁMETROS DE ENTRADA

Uso: `PSODaemon.exe [[-interprete <ficheroscript.txt>]|[<ficha.pro>]]`

Si se lanza el demonio sin parámetros de entrada, se da por hecho que la ficha o configuración del programa está en un fichero llamado ***ficha.pro*** en la misma carpeta donde está el programa.

En caso de que se tengo un fichero de configuración con otro nombre, se ha de lanzar el programa con los argumentos:

```
> PSODaemon.exe ficheroConfiguracion.pro
```

Por último, si queremos usar el intérprete integrado del PSO para comprobar que funciona nuestro fichero de script, debemos de llamarlo de la forma:

```
> PSODaemon.exe -interprete script.txt
```

C.2. FICHERO DE CONFIGURACIÓN

A continuación se muestra un fichero de configuración ejemplo que se usaría con el calculador del virus AH1N1 de forma distribuida con Sísifo:

NÚMERO DE PÁJAROS
200
TASA DE EXPLOTACIÓN
0.1
TASA DE EXPLORACIÓN
0.2
TASA DE ALEATORIEDAD
0.005
NÚMERO DE ITERACIONES
5000
ITERACIONES SIN MEJORA PARA DIVIDIR LA TASA ENTRE 2 (Si la función no mejora en X iteraciones, dividir las tasas entre dos)
600
NÚMERO DE PARÁMETROS
14
PARÁMETROS: LOS DINÁMICOS SON:(nombre; límite inferior; límite superior; Vmin; Vmax), LOS PARÁMETROS ESTÁTICOS (nombre; valor). OJO EN EL ORDEN.
realizaciones;1
nombre;H1NE
generador;0
t;195
nodos;450138
k;7;25;-0.5;0.5
nu;16
d;16
phi;0
L_phi;0;100;-2;2
I_phi;1;100;-2;2
beta;0.008;0.100;-0.01;0.01
om;4.5;5.5;-0.1;0.1
ro;6.5;7.5;-0.1;0.1
CARPETA DONDE ALOJAR LOS PROBLEMAS
./Problemas
CARPETA DONDE SE ANALIZARÁN LAS SOLUCIONES
./Soluciones
EJECUTABLE (Solamente cuando se usa en modo local, si no dejar en blanco)
../H1N1/bin/Release/H1N1
FICHERO CON EL SCRIPT DEL INTÉRPRETE, PARA EL CÁLCULO DE LA FUNCIÓN
./scriptH1N1NuevaEsparta.txt
MODO DE EJECUCIÓN (1 si se trata de Sisifo, 2 en caso de ejecutarse localmente)
2
PROCESOS SIMULTÁNEOS DEL CALCULADOR A EJECUTAR

4
FORMATO DE FICHERO .PRO (1 para fracaso escolar o VRS, 2 para H1N1)
2
ALPHA (LÍMITE O UMBRAL PARA GUARDAR FICHEROS .PRO Y .SOL)
0.10

Cada parámetro se define cada dos líneas en el fichero de configuración. La primera línea contiene el nombre y la explicación del parámetro y la segunda línea el valor de dicho parámetro. Esta regla se cumple en todos los casos menos en la lista de parámetros del calculador.

Es **muy importante** que se mantenga el orden de los parámetros indicados aquí arriba, puesto que de otra manera no se leerán los parámetros de forma correcta.

Un error en la ficha suele devolver el mensaje "Error convirtiendo valor a tipo base" o similares. Se recomienda comprobar que los números reales separan su parte decimal y entera con una coma y que el valor de número de parámetros coincide con el número de líneas indicadas.

A continuación se comenta cada parámetro en detalle:

NÚMERO DE PÁJAROS
200

Es la cantidad de pájaros o pruebas simultáneas del algoritmo PSO que se calcularán en cada iteración del algoritmo. Conocemos el conjunto total de pájaros de nuestro algoritmo como bandada.

TASA DE EXPLOTACIÓN
0.1

La tasa modela el movimiento autónomo del pájaro. Esta tasa se utiliza junto al último y mejor resultado obtenido por el pájaro para el cálculo parcial de la nueva posición del pájaro.

TASA DE EXPLORACIÓN
0.2

Cuantifica el acercamiento de todos los pájaros hacia el pájaro que mejor resultado ha dado hasta el momento. Esta tasa se utiliza con la posición actual de cada pájaro con la posición del mejor pájaro que tenemos en nuestra bandada.

TASA DE ALEATORIEDAD
0.005

Además de las tasas anteriores, hemos agregado ésta para añadir una ligera inexactitud en caso que se desee. Ésta tasa se aplica al rango de velocidades del parámetro dinámico, por ejemplo, si la velocidad mínima y máxima de un parámetro es -1 y 1 respectivamente, y nuestra tasa de aleatoriedad es 0.05, el resultado será un número aleatorio entre los valores [-0.05, 0.05]. Dicho valor se suma a los resultados de las tasas comentadas anteriormente.

NÚMERO DE ITERACIONES
5000

Es el número de repeticiones del bucle del algoritmo para que termine el programa. Sin embargo, el programa puede ser detenido o cerrado antes.

ITERACIONES SIN MEJORA PARA DIVIDIR LA TASA ENTRE 2
600

Este parámetro ayuda a afinar ligeramente el ajuste de forma automática si el resultado de la función F no ha disminuido o mejorado en las iteraciones indicadas. Cuando la función no varíe en este caso durante 600 iteraciones, las tasas de aleatoriedad, explotación y exploración se dividirán entre dos. Si en las 600 iteraciones siguientes tampoco se ha mejorado el mejor valor de F, se volverán a dividir las tasas entre dos, y así consecutivamente. Una vez divididas las tasas, no volverán a los valores originales hasta que se cierre y vuelva a abrir el programa.

NÚMERO DE PARÁMETROS
14

Es el número a parámetros que vamos a especificar en el parámetro siguiente, uno por línea. Dichos parámetros son usados en las pruebas generadas por los pájaros para ser ejecutados en el calculador correspondiente.

PARÁMETROS: LOS DINÁMICOS SON:(nombre; límite inferior; límite superior; Vmin; Vmax), LOS PARÁMETROS ESTÁTICOS (nombre; valor). OJO EN EL ORDEN.
realizaciones;1
nombre;H1NE
generador;0
t;195
nodos;450138
k;7;25;-0.5;0.5
nu;16
d;16
phi;0

L_phi;0;100;-2;2
I_phi;1;100;-2;2
beta;0.008;0.100;-0.01;0.01
om;4.5;5.5;-0.1;0.1
ro;6.5;7.5;-0.1;0.1

En este ejemplo concreto se muestran los parámetros que se usarán para el modelo de redes del virus AH1N1. Es muy importante que el número de líneas indicadas coincida con el valor del parámetro anterior (NÚMERO DE PARÁMETROS). De lo contrario el demonio leerá la ficha incorrectamente.

Cada línea contiene un parámetro de nuestro calculador. Existen dos tipos de parámetros, los estáticos, que como su nombre indica, mantendrán su valor en todas las pruebas generadas por el PSO y los dinámicos, que serán aquellos parámetros que queremos ajustar en el modelo.

El formato del parámetro estático es del tipo: **nombre;valor**. Solamente se utiliza el punto y coma para la separación entre campos de un parámetro. El parámetro estático "nombre" es tratado de forma especial, porque se usará el valor indicado como prefijo del nombre de los ficheros .pro y .sol generados. En el ejemplo actual, al ser **nombre;H1NE**, los ficheros generados se llamarían H1NE000001.pro y H1NE000001.sol respectivamente en el caso de ser la segunda prueba generada.

Para los parámetros dinámicos, el formato es: **nombre;limInf;limSup;Vmin;Vmax**

Los valores limInf y limSup son los limitadores inferiores y superiores respectivamente del valor que se quiere ajustar. Si una iteración del PSO hace que el valor de un parámetro se salga de los límites indicados, el programa los acota. Con los valores de Vmin y Vmax se hace este proceso de forma análoga con las velocidades del parámetro de cada pájaro.

CARPETA DONDE ALOJAR LOS PROBLEMAS
./Problemas

Se indica la ruta donde se colocarán los ficheros .pro generados por este programa. Se ha de tener en cuenta que la especificación de las rutas varía si se ejecuta el programa en entornos Windows o Unix, usando \ para las carpetas en Windows y / en Unix. Esto se debe hacer también para todas las rutas indicadas posteriormente.

CARPETA DONDE SE ANALIZARÁN LAS SOLUCIONES
./Soluciones

Es la carpeta donde Sísifo dejará las soluciones o si se ejecuta en modo local, el sitio donde los calculadores dejarán los ficheros .sol.

EJECUTABLE
../H1N1/bin/Release/H1N1

Lugar donde tenemos el fichero ejecutable que llamamos calculador. Éste parámetro es solamente usado si estamos usando el PSO en modo local. En caso de ser distribuido con Sísifo, el contenido de este parámetro se ignora. Sin embargo, en caso de ignorarse, es **obligatorio** que este parámetro y un valor cualquiera estén en la ficha.

FICHERO CON EL SCRIPT DEL INTÉRPRETE, PARA EL CÁLCULO DE LA FUNCIÓN
./scriptH1N1NuevaEsparta.txt

Aquí se indica donde se encuentra el fichero que contiene el código necesario para calcular la función F. Dicha función se utiliza junto al fichero solución (.sol) de cada pájaro para evaluar "cómo de buena" es dicha solución. El fichero script debe de contener una sintaxis conocida por el intérprete, la cual se detalla en capítulos posteriores.

MODO DE EJECUCIÓN (1 si se trata de Sisifo, 2 en caso de ejecutarse localmente)
1

El parámetro solo puede tener los valores 1 o 2. Si se va a utilizar con Sísifo elegimos el 1. No importa el orden de ejecución de Sísifo o del PSO, si Sísifo se abre antes, éste se quedará esperando a que el demonio PSO coloque los ficheros .pro en la carpeta de problemas que **ambos programas** han de tener en común, al igual que la carpeta soluciones, donde el demonio PSO se quedará esperando los ficheros .sol que devuelva Sísifo.

En el caso de ejecutarse localmente, se debe de elegir el número 2. En este modo el demonio PSO se encarga de la ejecución del calculador y gestión de los ficheros .pro y .sol. Se pueden ejecutar varios calculadores simultáneamente para aprovechar las ventajas de los procesadores multicore.

PROCESOS SIMULTÁNEOS DEL CALCULADOR A EJECUTAR
4

Aquí se indican el número de hilos o threads simultáneos del calculador. Este parámetro sólo es usado cuando se usa el demonio PSO en modo local.

FORMATO DE FICHERO .PRO (1 para fracaso escolar o VRS, 2 para H1N1)
2

Aquí se le indica al demonio PSO cómo debe generar las pruebas .pro para ser leídas por el calculador. Si tenemos los parámetros siguientes:

realizaciones;1
nombre;H1NE
generador;0

en el modo 1 o propio del calculador del fracaso escolar o VRS se mostrarán de la forma:

realizaciones
1
nombre
H1NE
generador
0

en el modo 2 el formato será el siguiente:

realizaciones=1;
nombre=H1NE;
generador=0;

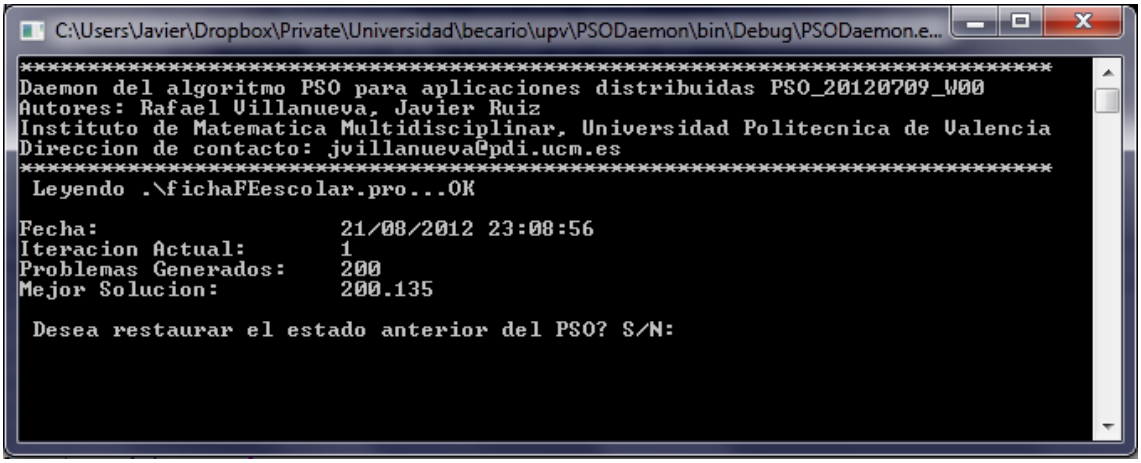
ALPHA (LÍMITE O UMBRAL PARA GUARDAR FICHEROS .PRO Y .SOL)
0.10

El parámetro selecciona un límite para almacenar aquellos ficheros de prueba .pro y solución .sol que ya no son usados por los pájaros. Aquellos pájaros por debajo del umbral se mostrarán en el fichero *lista_mejores.txt* en el mismo directorio que se encuentra el ejecutable del demonio PSO tras 2 iteraciones de su ejecución.

El funcionamiento es el siguiente: Si la mejor solución obtenida de todos los pájaros hasta el momento es **x**, aquellas soluciones que sean mayores que **(1 + α) · x** se eliminarán, o lo que es lo mismo, las soluciones menores o iguales a esta ecuación se quedarán en las carpetas correspondientes.

C.3. GUARDADO Y RESTAURACIÓN DEL ESTADO DEL ALGORITMO

El demonio guarda el estado actual del algoritmo en cada iteración en un fichero llamado estado.pso. Si el programa se cierra se puede volver al estado anterior ejecutándose de nuevo, la condición es que parte de la ficha no cambie y tampoco los problemas y soluciones dentro de sus respectivas carpetas.



Esta funcionalidad también puede ser utilizada para pausar el avance del algoritmo y cambiar ciertos parámetros de la ficha para reorientar a los pájaros. Sin embargo, no se puede cambiar el número de pájaros ni el número de parámetros, el resto sí se puede modificar.

C.4. EL INTÉRPRETE Y FICHERO SCRIPT

Para el cálculo de la función de evaluación F se debe de programar aparte un fichero script con el código correspondiente para evaluar cada solución. El intérprete incluido en el programa es bastante sencillo y tiene bastantes similitudes con la sintaxis de Mathematica. El código desarrollado para el intérprete debe poder ejecutarse también en Mathematica.

Después de ejecutar una prueba con el calculador se carga una nueva instancia del intérprete en memoria, primero cargando el fichero .sol generado por el calculador como código para el intérprete y posteriormente el script que se ha programado para la evaluación de la función F. El resultado de la función se retorna al demonio PSO con la función del intérprete Return[*valor*]; al igual que un método Module de Mathematica.

Al ser una variante del lenguaje muy reducida, se indicará a continuación con detalle aquellas operaciones que se permiten dentro del intérprete. También se incluye la gramática que sigue el intérprete en el Anexo F: Gramática y tokens del intérprete.

C.4.1 OPERADORES PERMITIDOS

Solamente está permitido el uso de los operadores aritméticos: `+, -, *, /, ^` y `Sqrt[]` y tienen el mismo efecto que en Mathematica. Por comodidad también se ha incluido el uso de los operadores: `++, --, +=` y `-=`. Sin embargo solamente están soportados los números enteros y reales y no se pueden hacer operaciones con listas enteras. Para ello se deberán usar bucles. También se permite el uso de todos los operadores con variables enteras y reales. Un ejemplo:

Código:

```
Print["1 + 2 * 2^2 = ", 1+2*2^2];
Print[Sqrt[81]];
```

Ejecución:

```
> 1 + 2 * 2^2 = 17
> 9
```

El uso de comparadores lógicos solamente se permiten en funciones condicionales, como `If` o `For`. Los comparadores integrados son: `==, <=, >=, <` y `>`. La concatenación de condiciones con operadores AND u OR no está soportada.

En las listas para acceder a sublistas o a un valor en concreto se usa el operador `[[ ]]`, los índices comienzan en 1 como en Mathematica. El uso de un índice 0 no está soportado por el intérprete. Un ejemplo puede ser el siguiente:

Código:

```
lista = {{5, 3}, 2};
Print[lista[[1]], " ", lista[[1]][[1]]];
```

Ejecución:

```
> {5,3} 5
```

C.4.2 VARIABLES

Se pueden crear variables durante todo el código, tanto enteras, reales como listas, sin embargo, en el caso de éstas últimas, no se podrán cambiar la cantidad de elementos de ellas. No hay operadores Append[] implementados ni similares. El funcionamiento de las variables es el mismo que el uso de variables globales en Mathematica. Un ejemplo:

Código:

```
a=1;
a+=a;
b=2.5;
c=a+b;
Print[a, " + ", b, " = ",c];
```

Ejecución:

```
> 2 + 2.5 = 4.5
```

Se pueden definir listas con varios elementos y cada elemento puede ser un entero, real, cadena de caracteres u otra lista. Se pueden tener tantos elementos como se desee, así como sublistas. No se puede modificar el contenido de una lista definida ni el número de elementos de ellas, solamente puede reemplazarse por completo.

C.4.3 FUNCIONES INTEGRADAS

El intérprete tiene unas funciones integradas de uso general, el funcionamiento de todas ellas es idéntico a como se aplica en Mathematica, excepto por el Return[]. La sintaxis de ellas se detalla a continuación:

Función	Descripción
Sqrt[]	Calcula la raíz cuadrada del valor en su interior.
Length[]	Indica el número de elementos de los que se compone una lista.
Print[]	Imprime un mensaje por pantalla. Se puede utilizar cuando el PSO está funcionando para mostrar mensajes por pantalla.
Return[]	Función que devuelve el valor de la función F al demonio PSO. No se ejecutará ninguna instrucción a continuación de esta.

C.4.4 CONDICIÓN IF

El funcionamiento es idéntico, tiene la estructura `If[condición, accionesTrue, accionesFalse];` Se pueden concantenar bucles y condiciones sin ningún problema.

C.4.5 BUCLE FOR

También funciona de la misma manera. La estructura es `For[inicialización, condición, actualización, acciones];` Sin embargo, solo se permite una instrucción en la inicialización, condición y actualización

C.4.6 OTRAS PUNTUALIZACIONES

- Para cada prueba se crea y se destruye un intérprete, así que se en un principio no tiene ninguna variable persistente en memoria.

- El lenguaje también permite comentarios mediante `(* *)`.

- Un error en el código puede volver el programa inestable y terminar indebidamente el demonio. Se recomienda la prueba del script cada vez que se modifique antes de lanzar el PSO ejecutando el demonio de la forma:

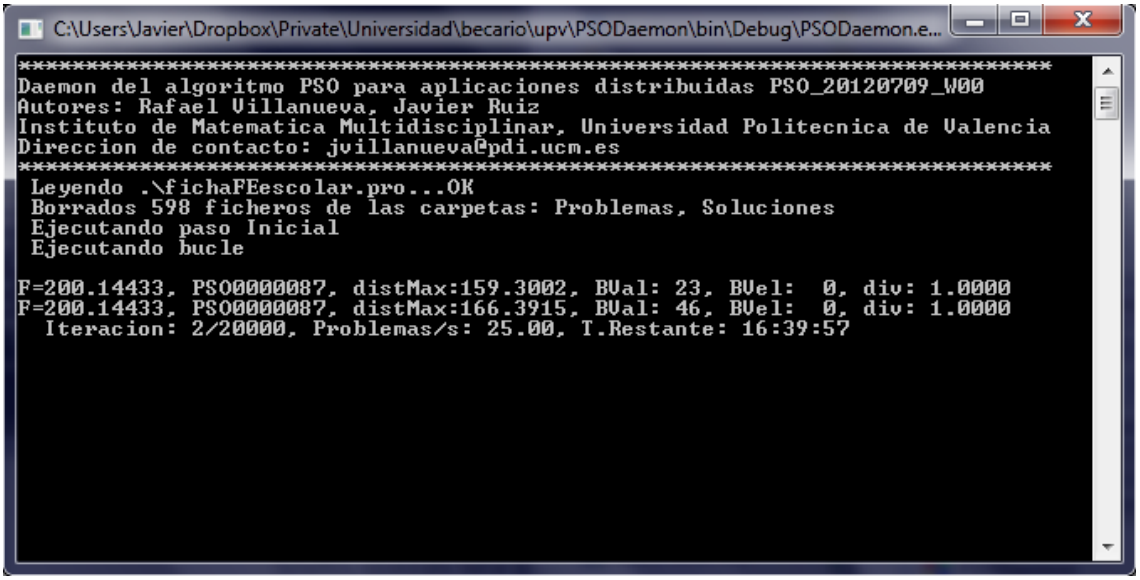
```
> PSODaemon.exe -interprete script.txt
```

Incluyendo al principio del script el contenido completo de un fichero .sol (Excepto el código CRC al final del fichero que obligatoriamente ha de escribir cada calculador). Se recomienda el uso del comando Print[] para la depuración. También se debería comprobar que los resultados coinciden con el resultado que nos devuelve Mathematica.

C.5. PUESTA EN MARCHA

Una vez se tiene preparado la ficha y el script se puede lanzar el demonio. Si se usa con Sísifo, quizás sea recomendable ejecutar el demonio PSO antes para limpiar las carpetas de problemas y soluciones.

En el caso de no haberse ejecutado anteriormente, se pondrá automáticamente a iterar, mostrando una pantalla similar a ésta:



Cada iteración devuelve una línea del estilo siguiente:

F=200.14433: Es el valor mínimo de la función encontrado
PSO0000087: El nombre de la prueba al que se le asocia el valor mínimo encontrado
distMax: Se trata de mayor distancia euclídea entre todos los pájaros del algoritmo. Se utiliza para saber como de separados están los más lejanos entre sí.
BVal: Contiene el porcentaje de parámetros de todos los pájaros que han tenido que ser acotados porque se salían de rango. Está en escala de 0 a 100
BVel: Contiene el porcentaje de velocidades que al igual que en el caso anterior, se salían de rango. También está en escala de 0 a 100.
div: Indica la división de las tasas aplicada actualmente. Ver parámetro ITERACIONES SIN MEJORA... de la ficha.

A partir de la segunda iteración se irán guardando las listas de mejores pájaros y mejores soluciones en el mismo directorio que se encuentra el ejecutable. Las listas incluyen el mejor F encontrado y el nombre del fichero.

Para comprobar que el PSO se ejecuta correctamente, se han de observar con detalle las dos primeras iteraciones, que es donde suele fallar en caso de haber un error en la configuración.

C.6. MENSAJES DE ERROR Y SOLUCIONES

Aquí se muestran los mensajes de error más frecuentes, su causa y posibles soluciones:

Mensaje	Leyendo *xxxx.pro*...ERROR abriendo el fichero
Causa	No ha encontrado la ficha o no tiene permisos de lectura
Solución	Comprobar que existe el fichero que intenta leer y también sus permisos.

Mensaje	Error! Unable to open file <*fichero*>
Causa	Error interno del intérprete, no ha encontrado el fichero script
Solución	Comprobar la ruta del fichero script

Mensaje	Error procesando linea *xxx*: syntax error, unexpected *yyy*, expecting *zzz*
Causa	Error sintáctico del intérprete, la sintaxis del fichero script es incorrecta o no está soportada por el lenguaje del intérprete.
Solución	Ejecutar primero el código en Mathematica. Si no hay errores, comprobar que se siguen las exigencias del intérprete viendo el capítulo 4

Mensaje	ERROR: Tipos incompatibles:... ERROR: Operacion no implementada:... ERROR: Operacion no implementada entre estos operandos:... ...
Causa	Error semántico del intérprete, el código intenta ejecutar operaciones ilegales o no están soportadas por el lenguaje del intérprete.
Solución	Ejecutar primero el código en Mathematica. Si no hay errores, comprobar que se siguen las exigencias del intérprete viendo el capítulo 4

Mensaje	Leyendo .*xxx.pro*...Error convirtiendo texto a tipo base
Causa	Existe un error en la ficha. Algún número real se ha definido con una coma en vez de un punto como separador entre enteros y decimales. También puede ser que las líneas estén incorrectamente desplazadas o no se siga el orden de los parámetros aquí indicados para la ficha.
Solución	Revisar ficha. Sobre todo los números reales, el número de líneas y comparar la ficha con otras fichas de ejemplo que se incluyen con el manual.

ANEXO D: DOCUMENTACIÓN DEL CALCULADOR AH1N1

ÍNDICE DE CLASES

LISTA DE CLASES

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

CContabilidad (Lleva la contabilidad de los ciudadanos afectados por estado de salud)
CFicha (Almacena los parámetros de la infección a simular)
CModelo (Clase con la que creamos el modelo de redes y lo evolucionamos)
CPersona (Atributos de una persona del modelo de redes)

DOCUMENTACIÓN DE LAS CLASES

REFERENCIA DE LA CLASE CCONTABILIDAD

Lleva la contabilidad de los ciudadanos afectados por estado de salud.

`#include <contabilidad.h>`

Métodos públicos

- CContabilidad (int dias, char *NombreFichero)
- ~CContabilidad ()

  Destructor de CContabilidad.
- void Inicializacion ()

  Pone a cero la contabilidad.
- void SumaDia (unsigned int dia, unsigned int *contabilidad_dia)
- void GuardaFichero (CFicha *ficha, CReloj *TiempoTranscurrido)
- void Media (int Realizaciones)

Atributos públicos

- unsigned int ** m_cuenta_personas_dias

  Variable que lleva la contabilidad de todos los días por estados de salud.

- unsigned int m_dias

  Variable que almacena los días totales de la simulacion.

- unsigned int m_ultimo_dia

  Variable que guarda cual ha sido el último día introducido.

- unsigned int m_max_dia

  Variable que indica hasta que día se puede hacer la media. (Es el menor día de todas las realizaciones).

- unsigned int m_dias_ejecutados

  Variable que almacena los días totales ejecutados de todas las realizaciones.

- char * m_NombreFichero

  Puntero al nombre del fichero.

Descripción detallada

Lleva la contabilidad de los ciudadanos afectados por estado de salud.

Esta clase permite la cuenta del estado de salud de las personas en un día concreto.

Definición en la línea 53 del archivo contabilidad.h.

Documentación del constructor y destructor

*CContabilidad::CContabilidad (int dias, char * NombreFichero)*`[inline]`

Constructor de CContabilidad.

Parámetros:

dias	Dias totales de la simulación. Parámetro T de la ficha
NombreFichero	Nombre de fichero .sol de salida

Definición en la línea 87 del archivo contabilidad.h.

Documentación de las funciones miembro

*void CContabilidad::GuardaFichero (CFicha * ficha, CReloj * TiempoTranscurrido)[inline]*

Vuelca toda la información de contabilidad en el fichero indicado, en este caso se trata del fichero solución (.sol). Se guardan el número de personas en un estado concreto de salud para todos los días de la simulación.

Parámetros:

ficha	Ficha del problema
TiempoTranscurrido	Tiempo de ejecución del problema

Definición en la línea 181 del archivo contabilidad.h.

Gráfico de llamadas para esta función:

void CContabilidad::Media (int Realizaciones)[inline]

La parte de contabilidad de eventos esta comentada, se deja para futuro uso.

Calcula la media de personas dividiendo por el número de realizaciones.

Parámetros:

Realizaciones	Número de realizaciones ejecutadas del mismo problema

Definición en la línea 262 del archivo contabilidad.h.

*void CContabilidad::SumaDia (unsigned int dia, unsigned int * contabilidad_dia)[inline]*

Suma los datos de contabilidad de un día de la simulación.

Parámetros:

dia	Día al que sumar la contabilidad
contabilidad_dia	Número de habitantes por estados de salud en el día actual

Definición en la línea 148 del archivo contabilidad.h.

La documentación para esta clase fue generada a partir del siguiente fichero:

- contabilidad.h

REFERENCIA DE LA CLASE CFICHA

Almacena los parámetros de la infección a simular.

`#include <ficha.h>`

Métodos públicos

- CFicha (string nombrefichero)

  Constructor por omision de la ficha.
- void LeerFicha (string nombrefichero)

  Lectura del fichero que contiene los parámetros a guardar en esta clase.
- void GuardaIdentificador (string *salida, CReloj *TiempoTranscurrido)

  Se guarda en el fichero de salida la información relativa a la salida.

Atributos públicos

- unsigned int m_realizaciones

  Tiempo en días para ejecutar la simulación.
- unsigned int m_T

  Tiempo en días para ejecutar la simulación.
- char m_Nombre_Fichero [512]

  Nombre del fichero de salida (con el .sol)
- unsigned int m_seed

  Generador de red.
- unsigned int m_Nodos

  Número total de nodos de la red.
- BASE_TYPE m_k

  Grado de los nodos.
- BASE_TYPE m_nu

  Tasa de natalidad al akño, nu por 1000.

- BASE_TYPE m_d

  Tasa de mortalidad al año, d por 1000.

- unsigned int m_phi

  Recuperados iniciales.

- unsigned int m_L_phi

  Latentes iniciales.

- unsigned int m_I_phi

  Infectados iniciales.

- BASE_TYPE m_beta

  Paso de Susceptible a Latente.

- BASE_TYPE m_om

  Paso de Latente a Infeccioso.

- BASE_TYPE m_ro

  Paso de Infeccioso a Recuperado.

- bool m_error

  Indica si no se ha leído el fichero correctamente.

Descripción detallada

Almacena los parámetros de la infección a simular.

Definición en la línea 59 del archivo ficha.h.

Documentación de las funciones miembro

*void CFicha::GuardaIdentificador (string * salida, CReloj * TiempoTranscurrido)*

Se guarda en el fichero de salida la información relativa a la salida.

Almacena el campo problema en el el fichero solución para poder ser leído por el intérprete de Mathematica. Dicho campo contiene toda la información necesaria de la ficha, incluyendo nuevos datos de la ejecución.

Parámetros:

salida	Fichero de salida en el que escribir los datos
TiempoTranscurrido	Contiene el tiempo transcurrido desde la ejecución del problema.

Definición en la línea 83 del archivo ficha.cpp.

Gráfico de llamadas a esta función:

void CFicha::LeerFicha (string nombrefichero)

Lectura del fichero que contiene los parámetros a guardar en esta clase.

Lee la ficha del problema, talla O(N) en peor caso, siendo N el número de parámetros totales.

Parámetros:

nombrefichero	Nombre de la ficha del problema, su extensión habitual es .pro

Definición en la línea 139 del archivo ficha.cpp.

Gráfico de llamadas a esta función:

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- ficha.h
- ficha.cpp

REFERENCIA DE LA CLASE CMODELO

Clase con la que creamos el modelo de redes y lo evolucionamos.

```
#include <modelo.h>
```

Métodos públicos

- CModelo (CFicha *ficha, CContabilidad *contabilidad)

  Constructor del modelo.

- ~CModelo ()

  Destructor del modelo.

- void Destructor ()

  Destructor auxiliar del modelo, para su posible reinicialización.

- void Inicializacion ()

  Inicialización del modelo de acuerdo con la ficha.

- void CierreContabilidad ()

  Cierre de contabilidad del modelo, hace la media de las realizaciones.

- const bool HabitantesRelacionados (int nodo1, int nodo2)

  Pregunta si dos habitantes están relacionados.

- void Relacionar (int nodo1, int nodo2)

  Se relacionan a dos habitantes.

- void Desrelacionar (int nodo1, int nodo2)

  Se dejan de relacionar a dos habitantes.

- void CambiaEstado (int nodo, char NuevoEstado)

  Cambiamos el estado y actualizamos contadores globales.

- void MuerePersona (int nodo)

  La persona indicada muere, haciendo los cambios oportunos en la red.

- void NacePersona ()

  Introducimos una nueva persona en la red, siendo susceptible.

- const unsigned int CuentaCuantos (char estado_de_salud)

  Cuenta el número de personas en la población con un determinado estado de salud.

- void Evolucion ()

  Corremos el modelo.

Descripción detallada

Clase con la que creamos el modelo de redes y lo evolucionamos.

Definición en la línea 56 del archivo modelo.h.

Documentación del constructor y destructor

*CModelo::CModelo (CFicha * ficha, CContabilidad * contabilidad)*

Constructor del modelo.

Constructor de la clase modelo.

Parámetros:

ficha	Ficha con los datos del modelo
contabilidad	Estructura para guardar los datos resultantes de la simulación.

Definición en la línea 67 del archivo modelo.cpp.

Documentación de las funciones miembro

void CModelo::CambiaEstado (int nodo, char NuevoEstado)`[inline]`

Cambiamos el estado y actualizamos contadores globales.

Cambia de un estado de salud a otro, actualmente sin uso.

Parámetros:

nodo	Habitante al que se le qure cambiar el estado de salud
NuevoEstado	Nuevo estado de salud que se desea que tenga el habitante.

Definición en la línea 464 del archivo modelo.cpp.

const unsigned int CModelo::CuentaCuantos (char estado_de_salud)

Cuenta el número de personas en la población con un determinado estado de salud.

Función que cuenta cuantos habitantes con un determinado estado de salud se encuentran en nuestro modelo. Sin embargo, esta función no se usa porque tiene coste N. Se lleva una cuenta del numero total de habitantes para cada estado salud y se almacena en la variable

m_cuenta_estado_salud. Esta función se usa en modo depuración para comprobar que ambos resultados coinciden y que la red sea consistente.

Parámetros:

estado_de_salud	Estado de salud solicitado para contar.

Devuelve:

El número de habitantes resultante de la cuenta.

Definición en la línea 436 del archivo modelo.cpp.

void CModelo::Desrelacionar (int nodo1, int nodo2)[inline]

Se dejan de relacionar a dos habitantes.

Dejar de relacionar dos habitantes dentro de nuestro modelo de redes.

Parámetros:

nodo1	Primer habitante
nodo2	Segundo habitante

Definición en la línea 194 del archivo modelo.cpp.

void CModelo::Evolucion ()

Corremos el modelo.

Esta función se encarga de la evolución del virus día a día por toda la población. También llama a las funciones necesarias para ir guardando resultados intermedios.

Definición en la línea 579 del archivo modelo.cpp.

const bool CModelo::HabitantesRelacionados (int nodo1, int nodo2)

Pregunta si dos habitantes están relacionados.

Indica si dos habitantes están relacionados.

Parámetros:

nodo1	Primer índice del habitante
nodo2	Segundo habitante

Devuelve:

Indica si están relacionados

Definición en la línea 153 del archivo modelo.cpp.

void CModelo::MuerePersona (int nodo)[inline]

La persona indicada muere, haciendo los cambios oportunos en la red.

Se muere la persona y se elimina el habitante de la red.

Parámetros:

nodo	Habitante a elminar

Definición en la línea 482 del archivo modelo.cpp.

void CModelo::NacePersona ()[inline]

Introducimos una nueva persona en la red, siendo susceptible.

Introducimos una nueva persona en la red, dicha persona se crea como un recién nacido y se le asigna un número de vecinos a través de una distribución de Poisson de media k. Se relaciona con todos ellos y se le asigna el estado de salud susceptible.

Definición en la línea 510 del archivo modelo.cpp.

void CModelo::Relacionar (int nodo1, int nodo2)[inline]

Se relacionan a dos habitantes.

Relacionar dos habitantes dentro de nuestro modelo de redes

Parámetros:

nodo1	Primer índice del habitante a relacionar
nodo2	Segundo habitante

Definición en la línea 177 del archivo modelo.cpp.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- modelo.h
- modelo.cpp

REFERENCIA DE LA ESTRUCTURA CPERSONA

Atributos de una persona del modelo de redes.

`#include <persona.h>`

Atributos públicos

- unsigned char m_estado_salud

  Estado de salud actual de la persona.

- unsigned short m_tiempo_estado_salud

  Días que lleva con el mismo estado de salud.

- unsigned int m_identificador

  Identificador de la persona.

- vector< unsigned int > m_vecinos

  Vecinos de esta persona.

Descripción detallada

Atributos de una persona del modelo de redes.

Definición en la línea 59 del archivo persona.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- persona.h

ANEXO E: DOCUMENTACIÓN DEL DEMONIO PSO

ÍNDICE DE CLASES

LISTA DE CLASES

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

CFicha
CGenProblemas
CInterprete
CPajaro
CParametro
CPSO
CPunto

DOCUMENTACIÓN DE LAS CLASES

REFERENCIA DE LA CLASE CFICHA

```
#include <ficha.h>
```

Métodos públicos

- void LeerFicha (string nombrefichero)
  Lectura del fichero que contiene los parámetros a guardar en esta clase.
- CFicha (string nombrefichero)
- ~CFicha ()
  Destructor de la ficha.

Atributos públicos

- unsigned int m_numPajaros
  Número de Pájaros.
- BASE_TYPE m_tasaExplotacion
  Tasa de explotación del algoritmo PSO.
- BASE_TYPE m_tasaExploracion
  Tasa de exploración del algoritmo PSO.
- BASE_TYPE m_tasaAleatoriedad
  Tasa de aleatoriedad del algoritmo PSO.

- unsigned int m_iteraciones
  Número de veces a repetir el bucle.
- unsigned int m_iteracionesDivisionTasas
  Cuantas iteraciones sin mejorar el ECM han de pasar para dividir las tasas.
- unsigned int m_nParametros
  Número de parámetros.
- CParametro ** m_vector_parametros
  Parámetros.
- string m_pathProblemas
  Path donde se encuentran los problemas.
- string m_pathSoluciones
  Path donde se encuentran las soluciones.
- string m_pathEjecutable
  Ejecutable que llamará el demonio PSO.
- string m_ficheroInterprete
  Fichero con el script del intérprete, que contiene la función de evaluación.
- string m_prefijoProblemas
  Es el prefijo que tendrán los ficheros .pro que genere el demonio, por defecto "Prueba".
- int m_tipoEjecutable
  Tipo de Ejecutable (1 si es Sísifo, 2 si es un calculador)
- int m_threadsSimultaneos
  Número de threads simultáneos a ejecutar en el caso del calculador.
- int m_tipoFichaSalida
  Formato de fichero .PRO a generar (1 si es tipo Fracaso Escolar, VRS, etc.., 2 si es tipo H1N1)
- BASE_TYPE m_alpha
  El porcentaje con el cual se conservarán los ficheros.

Descripción detallada

En esta clase se guarda la configuración que se lee del fichero .pro para todas las variables de nuestro algoritmo.

Definición en la línea 60 del archivo ficha.h.

Documentación del constructor y destructor

CFicha::CFicha (string nombrefichero)`[inline]`

Constructor por omision de la ficha.

Parámetros:

nombrefichero	Path completo o relativo a la ficha del PSO

Definición en la línea 136 del archivo ficha.h.

Documentación de las funciones miembro

void CFicha::LeerFicha (string nombrefichero)

Lectura del fichero que contiene los parámetros a guardar en esta clase.

Lector de la ficha del PSO, debe contener la información correspondiente al calculador, el modo de ejecución, los parámetros con sus rangos y velocidades para que el algoritmo funcione.

Parámetros:

nombrefichero	Ruta al fichero de configuración o ficha del PSO

Definición en la línea 52 del archivo ficha.cpp.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- ficha.h
- ficha.cpp

REFERENCIA DE LA CLASE CGENPROBLEMAS

`#include <genProblemas.h>`

Métodos públicos

- long getNumProblemasGenerados ()
- void setNumProblemasGenerados (long ficheros_generados)
- CGenProblemas (CFicha *ficha)
- void GeneraProblema (CPajaro *pajaro)

Descripción detallada

Esta clase genera los ficheros .pro internos para el algoritmo PSO

Definición en la línea 51 del archivo genProblemas.h.

Documentación del constructor y destructor

*CGenProblemas::CGenProblemas (CFicha * ficha)[inline]*

Constructor del generador de problemas

Parámetros:

ficha	Ficha con la configuración del PSO

Definición en la línea 90 del archivo genProblemas.h.

Documentación de las funciones miembro

*void CGenProblemas::GeneraProblema (CPajaro * pajaro)[inline]*

Crea un fichero .pro para ser ejecutado por el calculador deseado. Este fichero se genera con la información de la ficha y el pajaro de la ejecución del algoritmo PSO.

Parámetros:

pajaro	Pájaro que contiene el valor actual de los parámetros dinámicos

Definición en la línea 123 del archivo genProblemas.h.

long CGenProblemas::getNumProblemasGenerados ()[inline]

Número de problemas generados

Devuelve:

la variable solicitada

Definición en la línea 68 del archivo genProblemas.h.

void CGenProblemas::setNumProblemasGenerados (long ficheros_generados)[inline]

Guardar el número de problemas generados

Parámetros:

ficheros_generados	la variable a cambiar

Definición en la línea 76 del archivo genProblemas.h.

La documentación para esta clase fue generada a partir del siguiente fichero:

- genProblemas.h

REFERENCIA DE LA CLASE CINTERPRETE

`#include <interprete.h>`

Métodos públicos

- CInterprete (string ficheroFuente)
  Se guarda la dirección del fichero con el código a ejecutar a continuación.
- BASE_TYPE ejecutar (string ficheroSol)
  Ejecutamos primero el fichero Sol y luego el código del fichero intepretado.

Atributos públicos

- string m_contenidoFicheroFuente
  Variable en la que almacenamos el fichero fuente en memoria.
- bool m_error
  Si ha habido algún error, se almacenará aquí

Descripción detallada

Maneja el intérprete externo, preparándole la entrada y procesando la salida para ejecutar la función de evaluación, externa al programa. El intérprete está generado con flex y bison y forma parte de otro proyecto independiente, incluido en la misma carpeta del proyecto del PSO.

Definición en la línea 50 del archivo interprete.h.

Documentación del constructor y destructor

CInterprete::CInterprete (string ficheroFuente)

Se guarda la dirección del fichero con el código a ejecutar a continuación.

Constructor de la clase intérprete. Lee el fichero que contiene el script necesario para evaluar la solución de un pájaro.

Parámetros:

ficheroFuente	Ruta al fichero con el código del lenguaje interpretado

Definición en la línea 57 del archivo interprete.cpp.

Documentación de las funciones miembro

BASE_TYPE CInterprete::ejecutar (string ficheroSol)

Ejecutamos primero el fichero Sol y luego el código del fichero intepretado.

Ejecución del código del intérprete. Se incluye el fichero solución, cuyo código es ejecutado tal cual en el intérprete ANTES del script. El fichero solución debe de entregar código compatible con el intérprete y el fichero del script también, sino devolverá errores.

Parámetros:

ficheroSol	Fichero solución del problema resuelto

Devuelve:

Valor de la función F del script

Definición en la línea 85 del archivo interprete.cpp.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- interprete.h
- interprete.cpp

REFERENCIA DE LA CLASE CPAJARO

```
#include <pajaro.h>
```

Métodos públicos

- bool operator< (CPajaro &pajaro2)
  Función sobrecargada para comparar pájaros.
- CPajaro (CFicha *ficha, int id)
- void eliminaFichero (CFicha *ficha)
- BASE_TYPE sumaVelocidades ()
- unsigned int sumaBoundedValor ()
- unsigned int sumaBoundedVelocidad ()
- BASE_TYPE calculaDistancia (CPajaro *otroPajaro)
  1.

Atributos públicos

- CPunto * m_puntos
  Los puntos o valores de los parámetros correspondientes a este pájaro.

- BASE_TYPE ECM
  ECM o valor obtenido mediante el script del intérprete.
- string nombre
  Nombre de la prueba.
- bool estaMejores
  ¿Está incluído en la lista de los mejores?
- bool estaPajaros
  ¿Está incluído en la lista de pajaros?

Descripción detallada

Estructura que contiene toda la información de un pájaro del algoritmo PSO.

Definición en la línea 15 del archivo pajaro.h.

Documentación del constructor y destructor

*CPajaro::CPajaro (CFicha * ficha, int id)* `[inline]`

Constructor de pájaro

Parámetros:

ficha	Ficha del PSO
id	Identificador del pájaro a crear

Definición en la línea 56 del archivo pajaro.h.

Documentación de las funciones miembro

*BASE_TYPE CPajaro::calculaDistancia (CPajaro * otroPajaro)* `[inline]`

Calcula la distancia entre este pájaro y otro indicado por parámetro. Se realiza la distancia euclídea de ambos pájaros con todos los parámetros dinámicos de los pájaros. Cada parámetro representa una dimensión.

Parámetros:

otroPajaro	El otro pájaro con el que se quiere calcular la distancia

Devuelve:

Distancia obtenida

Definición en la línea 189 del archivo pajaro.h.

*void CPajaro::eliminaFichero (CFicha * ficha)[inline]*

Elimina el fichero correspondiente a este pájaro, se utiliza cuando este pájaro va a ser eliminado de memoria y la solución no nos resulta interesante.

Parámetros:

ficha	Ficha del PSO

Definición en la línea 97 del archivo pajaro.h.

unsigned int CPajaro::sumaBoundedValor ()[inline]

Suma la cantidad de parámetros cuyos valores se encuentran acotados por sobrepasar los límites indicados en la ficha del PSO.

Devuelve:

Cantidad de parámetros acotados

Definición en la línea 141 del archivo pajaro.h.

unsigned int CPajaro::sumaBoundedVelocidad ()[inline]

Suma aquellas velocidades que sobrepasan los límites de la ficha del PSO y por tanto, quedan acotadas.

Devuelve:

Número de velocidades acotadas del pájaro

Definición en la línea 164 del archivo pajaro.h.

BASE_TYPE CPajaro::sumaVelocidades ()[inline]

Suma todas las velocidades de los parámetros dinámicos del pájaro

Devuelve:

Resultado de la suma

Definición en la línea 118 del archivo pajaro.h.

La documentación para esta clase fue generada a partir del siguiente fichero:

- pajaro.h

REFERENCIA DE LA CLASE CPARAMETRO

`#include <ficha_parametro.h>`

Atributos públicos

- string name
  Nombre del parámetro.
- bool dinamico
  Parámetro dinámico o estático.
- BASE_TYPE lowerBound
  Valor mínimo.
- BASE_TYPE upperBound
  Valor máximo.
- BASE_TYPE minVelocity
  Mínima velocidad de cambio.
- BASE_TYPE maxVelocity
  Máxima velocidad de cambio.

Descripción detallada

Estructura que contiene la información de un parámetro a ajustar. Si el parámetro es dinámico, éste se usará en el PSO, en caso contrario mantendrá su valor original en todo el proceso.

Definición en la línea 15 del archivo ficha_parametro.h.

La documentación para esta clase fue generada a partir del siguiente fichero:

- ficha_parametro.h

Referencia de la Clase CPSO

`#include <PSO.h>`

Métodos públicos

- CPSO (CFicha *ficha)
- ~CPSO ()
  Destructor de la clase PSO.
- bool compruebaDependencias ()

Comprueba que el programa cumple todos los requisitos para funcionar.

- void ejecutar ()
  Arranca el algoritmo PSO.

Descripción detallada

Clase que implementa el algoritmo PSO, contiene todas las rutinas y las bibliotecas para funcionar de manera independiente.

Definición en la línea 58 del archivo PSO.h.

Documentación del constructor y destructor

*CPSO::CPSO (CFicha * ficha)* `[inline]`

Constructor de la clase que controla el algoritmo PSO

Parámetros:

ficha	Fichero con la configuración del PSO

Definición en la línea 258 del archivo PSO.h.

Documentación de las funciones miembro

bool CPSO::compruebaDependencias ()

Comprueba que el programa cumple todos los requisitos para funcionar.

Comprueba los posibles problemas que pueden ocurrir en ejecución por dependencias no resueltas. En caso de que se encuentra un problema, se detendrá la ejecución y se mostrará un mensaje de error.

Devuelve:

Devuelve true si no hay ningún problema, false en caso contrario

Definición en la línea 72 del archivo PSO.cpp.

void CPSO::ejecutar ()

Arranca el algoritmo PSO.

Inicializa el algoritmo y lanza la ejecución de éste. En caso de error, se devuelve el control a la función main y termina.

Definición en la línea 123 del archivo PSO.cpp.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- PSO.h
- PSO.cpp

REFERENCIA DE LA CLASE CPUNTO

`#include <punto.h>`

Atributos públicos

- BASE_TYPE valor
  Valor actual para la generación de fichero.
- BASE_TYPE velocidad
  Velocidad actual para la generación de fichero.
- bool boundedValor
  ¿Ha superado algún bound el valor?
- bool boundedVelocidad
  ¿Ha superado algún bound la velocidad?
- CParametro * parametro
  Referencia al parámetro.

Descripción detallada

Estructura que contiene la información actual de un parámetro dentro del algoritmo PSO

Definición en la línea 12 del archivo punto.h.

La documentación para esta clase fue generada a partir del siguiente fichero:

- punto.h

ANEXO F: GRAMÁTICA Y TOKENS DEL INTÉRPRETE

```
programa                → listaInstrucciones

listaInstrucciones      → ε
                        | instruccion PUNTOYCOMA_ listaInstrucciones
                        | comentario listaInstrucciones

comentario              → COMENTARIO_

instruccionOpcional     → ε
                        | instruccion

instruccion             → declaracionVariable
                        | instruccionEntradaSalida
                        | instruccionSalto
                        | instruccionBucle
                        | instruccionCondicion
                        | expresionIncremental

instruccionBucle        → bucleFor

bucleFor                → FUNC_FOR CORABR_ instruccionOpcional COMA_ expresion
                        COMA_ instruccionOpcional COMA_ listaInstrucciones CORCER_

instruccionCondicion    → condicionIf

condicionIf             → FUNC_IF CORABR_ expresion COMA_ listaInstrucciones
                        condicionElse CORCER_

condicionElse           → ε
                        | COMA_ listaInstrucciones

instruccionEntradaSalida→ funcionesInternas

instruccionSalto        → RETURN_ CORABR_ expresion CORCER_

declaracionVariable     → expresionIzquierda operadorAsignacion listaOExpresion
                        | VAR_ CORABR_ CORABR_ expresion CORCER_ CORCER_
                          operadorAsignacion listaOExpresion

listaOExpresion         → expresion
                        | LLAVABR_ listaVaciaOUnElemento LLAVCER_

elementosLista          → ε
                        | COMA_  listaOExpresion elementosLista

expresion               → expresionIgualdad
                        | expresionIzquierda operadorAsignacion expresion

funcionesInternas       → funcionPrint
                        | funcionLength
                        | funcionSqrt

funcionPrint            → FUNC_PRINT CORABR_ listaVaciaOUnElemento CORCER_

funcionLength           → FUNC_LENGTH CORABR_ expresion CORCER_

funcionSqrt             → FUNC_SQRT CORABR_ expresion CORCER_
```

```
expresionIzquierda      → VAR
                        |  varDimensiones

expresionIgualdad       → expresionRelacional
                        |  expresionIgualdad operadorIgualdad expresionRelacional

expresionRelacional     → expresionAditiva
                        |  expresionRelacional operadorIgualdad expresionRelacional

expresionAditiva        → expresionMultiplicativa
                        |  expresionAditiva operadorAditivo expresionMultiplicativa

expresionMultiplicativa → expresionPotencial
                        |  expresionMultiplicativa operadorMultiplicativo expresionUnaria

expresionPotencial      → expresionUnaria
                        |  expresionMultiplicativa operadorPotencial expresionUnaria

expresionUnaria         → expresionSufija
                        |  operadorUnario expresionUnaria
                        |  operadorIncremento VAR_

expresionSufija         → varDimensiones
                        |  expresionIncremental
                        |  VAR_ CORABR_ parametrosActuales CORCER_
                        |  PARABR_ expresion PARCER_
                        |  VAR_
                        |  CTE_
                        |  funcionesInternas

expresionIncremental    → VAR_ operadorIncremento

varDimensiones          → VAR_ dimension
                        |  varDimensiones dimension

dimension               → CORABR_ CORABR_ expresion CORCER_ CORCER_

parametrosActuales      → ε
                        |  listaParametrosActuales

listaParametrosActuales → expresion
                        |  expresion COMA_ listaParametrosActuales

operadorAsignacion      → IGUAL_
                        |  MASIG_
                        |  MENOSIG_

operadorIgualdad        → IGIG_
                        |  NOIGUAL_

operadorRelacional      → MAYOR_
                        |  MENOR_
                        |  MAYORIG_
                        |  MENORIG_

operadorAditivo         → MAS_
                        |  MENOS_

operadorMultiplicativo  → POR_
                        |  DIV_
```

```
operadorPotencial       → POW_

operadorIncremento      → MASMAS_
                        | MENOSMENOS_

operadorUnario          → MAS_
                        | MENOS_
```

Tabla de tokens activos:

Token	Carácter o descripción
MAS_	+
MENOS_	-
DIV_	/
POR_	*
POW_	^
CTE_	Un número entero, real, las cadenas 'True', 'False' y una cadena constante.
VAR_	El nombre de una variable, debe comenzar por una letra y posteriormente se permiten letras mayúsculas, minúsculas y números.
PARABR_	(
PARCER_	)
CORABR_	[
CORCER_	]
LLAVABR_	{
LLAVCER	}
UNDERSCORE_	_
PUNTO_	.
COMA_	,
PUNTOYCOMA_	;
DOSPUNTOS_	:
IGUAL_	=
IGIG_	==
NOIGUAL_	!=
MASIG_	+=
MENOSIG_	-=
PORIG_	*=
DIVIG_	/=
RETURN_	Return
COMENTARIO_	(* ... *)
MAYOR_	>
MENOR_	<
MAYORIG_	>=
MENORIG_	<=
FUNC_IF	If
FUNC_FOR	For
FUNC_LENGTH	Length
FUNC_SQRT	Sqrt
FUNC_PRINT	Print
MENOSMENOS_	--
MASMAS_	++

BIBLIOGRAFÍA

[1] G. González Parra, R. J. Villanueva Micó y J. Ruiz Baragaño, «Modelling influenza A(H1N1) epidemics using random networks in a distributed computing environment».

[2] CDC, «CDC - H1N1 flu,» [En línea]. Available: http://www.cdc.gov/h1n1.

[3] Novel Swine-Origin Influenza A (H1N1) Virus Investigation Team, «Emergence of a novel swine-origin influenza A (H1N1) virus in humans,» *New England Journal of Medicine,* vol. 25, nº 360, pp. 2605-2615, 2009.

[4] S. M. Tracht, S. Y. Del Valle y M. J. Hyman, «Mathematical modeling of the effectiveness of facemasks in reducing the spread of novel influenza A (H1N1).,» vol. 2, nº 5, p. e9018, 2010.

[5] S. Takeuchi and Y. Kuroda, "Predicting spread of new pandemic swine-origin influenza A(H1N1) in local mid-size city: evaluation of hospital bed shortage and effectiveness of vaccination.," *Nippon Eiseigaku Zasshi,* vol. 1, no. 65, pp. 48-52, 2010.

[6] G. F. Webb, Y.-H. Hsieh, J. Wu y M. J. Blaser, «Pre-symptomatic influenza transmission, surveillance, and school closings: Implications for novel influenza A (H1N1),» *Math. Model. Nat. Phenom.,* vol. 3, nº 5, pp. 191-205, 2010.

[7] S. Towers y Z. Feng, «Pandemic H1N1 influenza: predicting the course of a pandemic and assessing the efficacy of the planned vaccination programme in the Unisted States.,» *Euro Surveill.,* vol. 41, nº 14, pp. 1-3, 2009.

[8] J. B. Ong, M. I. Chen, A. R. Cook, H. C. Lee, V. J. Lee, R. T. Lin, P. A. Tamyah y L. G. Goh, «Real-time epidemic monitoring and forecasting of H1N1-2009 using influenza-like illness from general practice and family doctor clinics in Singapore.,» *PLoS ONE,* vol. 4, nº 5, p. e10036, 2010.

[9] D. Balcan, H. Hu, B. Goncalves, P. Bajardi, C. Poletto, J. Ramasco, D. Paolotti, N. Perra, W. Michele, M. Tizzoni, Broeck, V. Colizza y A. Vespignani, «Seasonal transmission potential and activity peaks of the new influenza A(H1N1): a Monte Carlo likelihood analysis based on human mobility,» *BMC Medicine,* vol. 1, nº 7, p. 45, 2009.

[10] G. Katriel, R. Yaari, A. Huppert, U. Roll and L. Stone, "Modelling the initial phase of an epidemic using incidence and infection network data: 2009 H1N1 pandemic in Israel as a case study," *Journal of The Royal Society Interface,* vol. 59, no. 8, pp. 856-867, 2011.

[11] E. Zulueta Guerrero, A. González González, J. M. Lopez-Guede y I. Calvo Gordillo, «Simulación basada en SMA de sistemas originalmente representados con EDO,» *Revista Iberoamericana*

de Automática e Informática Industrial RIAI, vol. 4, nº 9, pp. 323-333, 2011.

[12] L. A. Amaral and J. Ottino, "Complex networks - augmenting the framework for the study of complex systems," *European Physical Journal B,* vol. 2, no. 38, pp. 147-162, 2004.

[13] G. Witten y G. Poulter, «Simulations of infectious diseases on networks,» *Computers in Biology and Medicine,* vol. 2, nº 37, pp. 195-205, 2007.

[14] L. Acedo, J.-. A. Moraño, R. J. Villanueva, J. Villanueva Oller and J. Díez Domingo, "Using random networks to study the dynamics of Respiratory Syncytial Virus (RSV) in the spanish region of Valencia," *Mathematical and Computer Modelling,* vol. 78, no. 54, pp. 1650-1654, 2011.

[15] L. Acedo, J. A. Moraño y J. Díez Domingo, «Cost analysis of a vaccination strategy for Respitatory Syncytial Virus (RSV) in a network model,» *Mathematical and Computer Modelling,* vol. 78, nº 52, pp. 1016-1022, 2010.

[16] E. Massad, S. Ma, M. Chen, C. Jos Stuchiner, N. Stollenwerk y M. Aguiar, «Scale-free network of a dengue epidemic,» *Applied Mathematics and Computation,* vol. 2, nº 195, pp. 376-381, 2008.

[17] G. González Parra, L. Acedo, R. J. Villanueva Micó y A. J. Arenas, «Modelling the social obesity epidemic with stochastic networks,» *Physica A: Statistical Mechanics and its Applications,* vol. 17, nº 389, pp. 3692-3701, 2010.

[18] S. M. Firestone, R. M. Christley, P. M. Ward y N. K. Dhand, «Adding the spatial dimension to the social network analysis of an epidemic: Investigation of the 2007 outbreak of equine influenza in Australia,» *Preventive Veterinary Medicine,* vol. 0, nº 0, p. In press, 2012.

[19] C. Groendyke, D. Welch y D. R. Hunter, «Bayesian inference for contact networks given epidemic data,» *Scandinavian Journal of Statistics,* vol. 3, nº 38, pp. 600-616, 2011.

[20] S. Azaele, A. Maritan, E. Bertuzzo, I. Rodríguez Iturbe y A. Rinaldo, «Stochastic dynamics of cholera epidemics,» *Phys. Rev. E,* nº 2010, p. 051901, 2010.

[21] G. Gonzalez Parra, R. J. Villanueva Micó y L. Segovia, «Dinámica del virus pandémico AH1N1/09 en la población de Venezuela,» *Interciencia,* vol. 37, nº 4, Abril de 2012.

[22] «BOINC Wiki Site - User Manual,» [En línea]. Available: http://boinc.berkeley.edu/wiki/How_BOINC_works. [Último acceso: 8 Agosto 2012].

[23] «SETI@home - How SETI@home works,» [En línea]. Available: http://seticlassic.ssl.berkeley.edu/about_seti/about_seti_at_home_1.html. [Último acceso: 8 Agosto 2012].

[24] J. Kennedy, "A modified particle swarm optimizer," *Proceedings of IEEE International Conference on Evolutionary Computation,* pp. 69-73, 1997.

[25] P. L'Ecuyer, "Maximally equidistributed combined Tausworthe generators," *Mathematics of Computation,* no. 65, pp. 203-212, 1996.

[26] P. L'Ecuyer, Random number generation, chapter 4 of the Handbook of Simulation, Jerry Banks, Wiley, 1997.

[27] Scaevolus, «Codifies - Blogger,» 15 Mayo 2008. [En línea]. Available: http://cod.ifies.com/2008/05/php-rand01-on-windows-openssl-rand-on.html. [Último acceso: 15 Agosto 2012].

[28] Zevan, «ActionSnippet,» 19 Febrero 2009. [En línea]. Available: http://actionsnippet.com/?p=809. [Último acceso: 15 Agosto 2012].

[29] «Wikipedia - Poisson Process,» [En línea]. Available: http://en.wikipedia.org/wiki/Poisson_process. [Último acceso: 16 Agosto 2012].

[30] D. Willkomm, S. Machiraju, J. Bolot y A. Wolisz, «Primary user behaviour in cellular networks and implications for dynamic spectrum access,» *IEEE Communications Magazine,* vol. 3, nº 47, p. 88, 2009.

[31] A. Heuer, C. Müller y O. Rubner, «Soccer: is scoring goals a predictable Poissonian process?,» *EPL (Europhysics Letters),* vol. 3, nº 89, p. 38007, 2010.

[32] M. F. Arlitt y C. L. Williamson, «Internet Web servers: Workload characterization and performance implications,» *IEEE/ACM Transactions on Networking,* vol. 5, nº 5, p. 631, 1997.