UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

ALGORITMOS INSPIRADOS EN FENÓMENOS SOCIALES Y CIENCIAS SOCIALES COMPUTACIONALES

REPORTE DE INVESTIGACIÓN

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

SERGIO HERNÁNDEZ LÓPEZ

TUTOR
DR. JOSÉ ANTONIO NEME CASTILLO

2011

Hoja de Datos del Jurado

1. Datos del alumno
Hernández
López
Sergio
56 08 13 25
Universidad Nacional Autónoma de México
Facultad de Ciencias
Ciencias de la Computación
096504822

2. Datos del tutor
Dr
José Antonio
Neme
Castillo

3. Datos del sinodal 1
Dr
Carlos
Gershenson
Garcia

4. Datos del sinodal 2
L en CC
Victor
Mireles
Chávez

5. Datos del sinodal 3
Dr
José de Jesús
Galaviz
Casas

6. Datos del sinodal 4
Dr
José Luis
Gutiérrez
Sánchez

7. Datos del trabajo escrito
Algoritmos Inspirados en Fenómenos Sociales y Ciencias Sociales Computacionales
60p
2011

Índice general

Agradecimientos

Quisiera agradecer a las siguientes personas e instituciones por haberme apoyado para conseguir este logro.

Antes que todo quisiera agradecer a la Universidad Nacional Autónoma de México, la UNAM, así como toda la comunidad que la conforma por ser una institución cuya nobleza y compromiso social agradeceré siempre. Consecuentemente quisiera extender este agradecimiento al pueblo de México, responsables directos de la existencia de esta institución, y con quienes estoy en deuda.

A mi familia. A mi madre, cuya dedicación, cariño y confianza me impulsan diario a superarme. A mi padre por presentarme e interesarme en la ciencia, computación y matemáticas desde pequeño y por ser el modelo de persona que quiero ser y a mi hermana cuya solidaridad y amistad me mantienen (mas o menos) cuerdo.

A Yerye, Leo, Rodrigo, Paulo, Sele, Lu por estar conmigo en los momentos más difíciles, entrañables y también los más divertidos e interesantes, muchos de ellos en la Facultad de Ciencias.

A Victor, Bárbara y Natalia, por su entrañable amistad y apoyo así como por su interesante visión del mundo, sociedad y ciencia y por todas las amenas charlas alrededor de una cerveza y plato de sushi.

Particularmente quiero agradecer al reponsable del proyecto, tutor y amigo Antonio Neme Castillo por haberme incluido en un proyecto interesante y fundamental para mi formación profesional y personal.

De igual manera a la Academia de Dinámica No Lineal de la Universidad Autónoma de la Ciudad de México, en la cual llevo a cabo mis estudios de maestría y que constituyen la base teórica usada para abordar el proyecto.

Finalmente a los miembros de mi jurado, Carlos Gershenson, José Luis Gutiérrez, Victor Mireles y José Galaviz por sus valiosas observaciones a este trabajo.

Finalmente a todas las personas que han participado en seminarios, consultas y cursos derivados en el análisis del problema abordado.

A todos, gracias.

Capítulo 1

Introducción

Theory without data is myth: data without theory is madness.
Phil Zuckerman

1. Objetivo

El presente proyecto tiene como objetivo estudiar el estado del arte de las relaciones entre las Ciencias Sociales y las Ciencias Sociales Computacionales desde una perspectiva de las Ciencias Computacionales, así como proponer algunos modelos de las Ciencias de la Computación diseñados para explicar algún comportamiento social.

En la primer parte el proyecto consiste en conocer y analizar los algoritmos existentes que están basados en procesos sociales humanos que cumplen con la característica de que el proceso optimizado fuera un fenómeno emergente. También se hizo un análisis de la Complejidad Computacional de los algoritmos analizados.

En la segunda parte se desarrollaron modelos de sistemas sociales usando herramientas computacionales con el fin de reproducir y analizar la dinámica observada usando herramientas de modelación diseñadas dentro del ámbito de las Ciencias Computacionales.

2. Ciencias Sociales y Computación

El desarrollo de los algoritmos de optimización se fundamenta en buena medida en el análisis numérico y herramientas de tipo matemático. Los problemas son reexpresados en lenguaje matemático y así poder estudiarlos con una vasta gama de herramientas matemáticas que los científicos puedan echar mano.

Pero esta aproximación a la resolución y optimización de problemas requiere de la traducción del mismo en lenguaje matemático, aspecto que puede ser una tarea titánica si no es que imposible en la mayoría de los casos de manera que las soluciones con esta aproximación son generalmente limitadas.

Es por esto que las diversas disciplinas del quehacer científico e ingenieril hallaron en la naturaleza una vasta fuente de inspiración para buscar soluciones de tipo numérico a los diversos problemas con los que nos enfrentamos. Algoritmos basados estructuras sociales vistos en las colonias de hormigas dieron pie a la inteligencia de enjambre, tomar a la evolución como fuente de inspiración dio pie a los algoritmos genéticos, etcétera.

Esta fuente inspiración para encontrar algoritmos nuevos no quedó úicamente en metáforas naturales donde el ser ser humano no figuraba. Los sistemas sociales humanos también han sido abstraídos para desarrollar algoritmos de optimización, como ejemplo tenemos la aportación hecha por Robert Reynolds [17] quien desarrolló un algoritmo basado en la transferencia de experiencia observada en una comunidad oaxaqueña, este algoritmo se le denominó Algoritmo Cultural.

Los sistemas sociales, en su forma más general, son el objeto de estudio de las Ciencias Sociales. Disciplinas como la Historia, Antropología, Sociología, Economía, y muchas otras se ocupan principalmente de el comportamiento y la actividad humana, así como las relaciones entre los integrantes de los grupos sociales y de estos con su entrono. De forma genérica podemos decir que estas disciplinas se encargan del ser humano en algún contexto.

A diferencia de las Ciencias Naturales[1] donde la creación de modelos se basa en la habilidad para descartar y elegancia con la que se puedan descartar detalles observados con el fin de describir un fenómeno con una cantidad mínima de elementos (y no menos) es una tarea cotidiana y la polémica surge cuando lo observado no coincide con los resultados arrojados por el modelo, en las Ciencias Sociales, en cambio, una práctica común es hacer reflexiones sobre la realidad con toda la riqueza y sofisticación que el observador pueda percibir. Tal vez sea por esto que la creación de modelos, ya sean explicativos o predictivos, bajo esta perspectiva sea una tarea particularmente difícil.

Esto, encontrar modelos que describieran correctamente sistemas sociales, así como otros fenómenos naturales que no podían ser descompuestos nos plantea la necesidad de encontrar una nueva forma de estudio en la que se puedan incorporar las características y relaciones que los definen.

[1] Por Ciencias Naturales nos vamos a referir al concepto más común, es decir, aquellas donde la intervención del ser humano es marginal o bien nula dentro del objeto de estudio de la misma

2.1. La Computación como herramienta cognitiva

Edsger W. Dijkstra (1930-2002), renombrado investigador de la Ciencia de la Computación decía de esta que "no trata más sobre computadoras más de lo que la astronomía trata sobre telescopios". Esta reflexión resalta el concepto en el que comúnmente se tiene a la Ciencia de la Computación: como un conjunto de técnicas diseñadas para eficientizar el procesamiento (cálculo) de la gran cantidad de datos recabados a través de mediciones.

De manera informal y muy popular podemos entender que la Computación se encarga del procesamiento automático de la información. Pero el concepto de computación es mucho más profundo. Más formalmente el objeto de estudio de esta ciencia es analizar y encontrar procesos de mapeo entre cualquier par de conjuntos mediante una cantidad de pasos decidibles. Es decir, que después de ejecutar esta serie de pasos sea posible establecer la correspondencia entre elementos pertenecientes a ambos conjuntos. Usando esta definición los computólogos se encargan de buscar y diseñar algoritmos que lleven a cabo algún proceso de cómputo así como determinar lo que no sea computable.

El diseño de algoritmos computacionales inspirados en fenómenos característicos de las Ciencias Naturales comenzó en las décadas de 1960 y 1970, mismo que derivó en la formalización del Algoritmo Genético, utilizado principalmente como un método heurístico para encontrar soluciones óptimas. Este toma a la evolución como metáfora optimizadora y fue desarrollado por John Holland en 1975 [8]. Otro ejemplo que cabe citar es el de las redes neuronales o el de la inteligencia de enjambre. Estos son unos de los más conocidos modelos que son usados como herramientas usadas para clasificar u optimizar inspirados en fenómenos naturales.

Ha sido precisamente en este ámbito, el estudio de los fenómenos naturales, donde el uso de la computadora es muy socorrido, ya sea a través de análisis de simulaciones numéricos (*number crunching*) o mediante la implementación de simulaciones por computadora de modelos científicos, y que hoy en día se concibe como experimentación por computadora sin embargo en el resto de las disciplinas el uso que se hace de ésta es mas bien trivial y bien podría decirse que hasta marginal.

La modelación por computadora o *modelación in silico* es una técnica de modelado que ha adquirido popularidad desde 1970 básicamente dentro del ámbito de las Ciencias Naturales donde la implementación de modelos computacionales permitió el estudio de propiedades y características en menos tiempo o menos recursos comparándolos con la creación de un experimento o la construcción física de un modelo.

La validez de esta nueva forma de hacer ciencia recae precisamente en la definición de computación expuesta previamente, ya que no se requiere de re-

producir el fenómeno en su entorno, sino únicamente la forma en que los componentes de dicho fenómeno se relacionan entre sí. Es decir, la experimentación por computadora permitió separar el fenómeno del sustrato en el que ocurre.

Pero la cualidad de la experimentación por computadora que me interesa destacar es la capacidad de presentar un comportamiento complejo, cuyas características se describen en la sección 2, y el mejor ejemplo son los Sistemas de Autómatas Celulares que a partir de un conjunto de reglas de interacción sencillas muestran una evolución compleja, aspecto que quedó plasmado por Stephen Wolfram en su clasificación de las distintos comportamientos observados en su libro "A New Kind of Science".

Esta cualidad no representa únicamente un ahorro de tiempo o recursos, como comúnmente podría pensarse sobre la utilidad de los modelos hechos en computadora, ni tampoco a reproducir, en mayor o menor medida lo observado, sino algo conceptualmente más profundo que provee de robustez a los modelos computarizados y es la capacidad de que el modelo programado, iniciado con otro conjunto de propiedades iniciales, pueda exhibir comportamientos nuevos que hayan escapan a la observación inicial.

Esto permite explorar comportamientos nuevos en la simulación que emergen en buena medida al detalle de las reglas programadas en los componentes que conforman al sistema así como a las reglas que tienen estos con el entorno.

Aparte de los Sistemas de Autómatas Celulares existe otra herramienta emanada de las Ciencias de la Computación y que ha ganado popularidad desde hace 20 años, es la Modelación Basada en Agentes. Segun Gilbert Nigel [15]

> "Formalmente hablando, la modelación basada en agentes es un método computacional que permite al investigador analizar y experimentar con modelos compuestos por agentes en un entorno."

donde los agentes son generalmente considerados como pequeñas entidades autónomas que no están sujetas a un control central que dicte su dinámica.

Este paradigma de modelado resulta muy útil cuando se desean hacer simulaciones de sistemas que constan de una gran cantidad de componentes heterogéneos y autónomos que se pueden comunicar de los cuales se conocen en alguna medida las reglas que determinan su actuar. En el caso que nos interesa, la modelación de sistemas sociales, se puede aplicar con más facilidad y naturalidad que otros paradigmas computacionales. Este paradigma se analiza con más detalle en la sección 3.

En este sentido, las Ciencias Sociales Computacionales se presentan como una extensión en lo que al uso de la computadora se refiere, mostrando un

cambio cualitativamente distinto cuando se estudian a los grupos sociales mediante la observación, abstracción de mecanismos *esenciales* para su posterior implementación en modelos computacionales donde la exploración de hipótesis o generación de las mismas sea un proceso guiado por los resultados arrojados por dichos modelos.

3. Estructura

En la sección 2 se explica brevemente que es la Teoría de la Complejidad como un paradigma científico en el cual la dinámica y estructura de los sistemas sociales queda explicada más propiamente.

En la sección 3 se explican las herramientas computacionales que se usaron a lo largo del proyecto.

La primer parte del proyecto que consistió en hacer una investigación bibliográfica para documentar y analizar una amplia gama de algoritmos computacionales basados en procesos sociales junto con la modificación al algoritmo del SOM usando el dilema del prisionero como heurística está en explicada en la sección 4.

La segunda parte del proyecto, que consistió en desarrollar un modelo utilizando mapeos auto-organizados para explicar la influencia de de la publicidad y comunicación entre votantes en un proceso electoral, está desarrollada en la sección 5.

Los resultados obtenidos para las distintas etapas del proyecto se presentan en 6 y las conclusiones al trabajo están presentadas en la sección 7.

Capítulo 2

Sistemas Complejos

La metodología científica hasta ahora ha trabajado con los diversos problemas que se le han presentado mediante la separación de los mismos en problemas más sencillos con el principio de que el todo es la suma de las partes.

Sin embargo este principio es válido para muy pocos de los fenómenos en el mundo natural[1] y es precisamente por esto que desde mediados del siglo pasado se ha venido desarrollando un nuevo paradigma, el de los Sistemas Complejos así como las herramientas necesarias para abordar este problema.

1. Generalidades

Los sistemas complejos son aquellos que para describirlos se requiere saber no sólo la estructura y partes que lo componen, sino también como es que dichas partes interactúan entre sí, estos son estudiados por la naciente Teoría de la Complejidad. Esta es una de las razones por las que todavía no existe una definición ampliamente aceptada por la comunidad de investigadores en esta área pero la mayoría está de acuerdo en que todos presentan las siguientes características:

Colectividad Estos sistemas constan de una gran cantidad –mayor a cierto número crítico– de unidades que lo componen y cuyas reglas de comportamiento se caracterizan como sencillas o simples.

Auto-organización Los sistemas complejos, comúnmente, carecen de un control central que rija el actuar de las unidades constituyentes, en vez de esto, es el intercambio de información entre unidades vecinas que constituyen al sistema lo que da lugar al comportamiento y estructura organizada que

[1] Ver P. Miramontes. "Del maligno señor, defiéndeme...", *Ciencias* No. 46, 1997

podemos observar en ellos.

Emergencia Una de las características más sobresalientes dentro de los sistemas complejos es la capacidad de presentar propiedades emergentes, este proceso, derivado directamente de la dinámica y las relaciones entre las unidades que los conforman, es el responsable de presentar nuevas estructuras en el sistema. La descripción común de que en los sistemas complejos *el todo es más que la suma de sus partes*[2] se le debe, en una opinión personal, a esta característica[3].

Adaptabilidad Estos sistemas presentan la capacidad de adaptarse a su entorno mediante evolución o aprendizaje de las unidades que los conforman.

Es común que a las propiedades emergentes se les refiera como impredecibles, aunque una descripción más adecuada es el hecho de que no pueden ser previstas únicamente de analizar el sistema por separado, como la paradigmática consigna *divide y vencerás* lo establece y se aplica en la metodología clásica de la ciencia. Es a partir de observar la dinámica del sistema que estas propiedades pueden ser apreciadas.

Hasta este punto he descrito a los Sistemas Complejos como un tipo de sistema cualitativamente distinto a los Sistemas Simples pero algo bien conocido es que este fenómeno que denominamos como complejidad puede aparecer en cualquier sistema dependiendo del nivel de descripción requerido. Es decir, mientras más cerca nos fijemos en las partes interactuantes de un sistema vamos a poder notar las características arriba mencionadas. Un ejemplo de esto puede ser visualizar a la célula con muy poco detalle de tal forma que sólo pueda tener dos estados posibles o con mucha mayor resolución donde se pueda distinguir la interacción de los organelos que la constituyen.

Para estudiar estos sistemas se ha hecho un uso extensivo herramientas desarrolladas principalmente dentro de la Matemática, Física y Computación dentro de las que podemos encontrar la Teoría de Gráficas, Teoría de Juegos, Análisis de Series de Tiempo, la Teoría de Sistemas Dinámicos particularmente el Análisis de Estabilidad, Teoría del Caos, Teoría de Bifurcaciones, etcétera. Dentro de las herramientas computacionales están los Autómatas Celulares, las Redes Neuronales, la Lógica Difusa, etcétera.

[2]Este dicho, aunque paradójico, refleja bien el papel que juegan las reglas de interacción entre las partes que componen el sistema para poder determinar el estado final del mismo.

[3]Como complemento a los sistemas complejos están los sistemas simples, que son el tipo de sistemas que conocemos más ampliamentemente. Estos se caracterizan por una inherente separabilidad, es decir, el cómputo del sistema se encuentra perfectamente agrupado en distintas partes de este

La definición que Melanie Mitchell ofrece en [10] captura las características de los sistemascomplejos en el siguiente párrafo:

> "Un sistema es complejo si está compuesto de grandes redes de componentes que carecen de un control central y se rigen con reglas simples de operación son capaces de presentar un comportamiento complejo, sofisticado procesamiento de información y adaptabilidad vía aprendizaje o evolución."

2. Sistemas Sociales y complejidad

Esta definición viene a colación porque aparte de caracterizar de manera concisa lo que actualmente se entiende por sistema complejo también establece paralelismo claro con un sistema social.

Y es que los sistemas sociales presentan muchas de las características descritas dentro de la Teoría de la Complejidad. Están compuestos por una gran cantidad de componentes que gozan de autonomía e interaccionan dentro de un espacio acotado, estos componentes constituyentes (las personas) muestran un comportamiento colectivo auto-organizante y cooperativo que da lugar a la aparición de propiedades emergentes propias de sistemas sociales como ejemplo podemos poner al lenguaje, la política, cultura, economía, etcétera.

Este aspecto es interesante ya que la creencia común es que el rumbo de una sociedad está dictada por un control central mismo que, partiendo del principio de que este posea información global (y precisa) del sistema puede dirigir su destino [16], sin embargo una característica de los sistemas complejos es su dificultad para predecir estados del mismo [1].

Capítulo 3

Herramientas computacionales

Como se mencionó en la sección 2 existen diversos tipos de herramientas computacionales muy socorridas en el modelado de fenómenos complejos y también heurísticas. En este proyecto se usaron un tipo de red neuronal llamada mapeos autoorganizados (SOM por sus siglas en inglés), el dilema del prisionero que ha probado ser un paradigma muy socorrido dentro de la Teoría de Juegos.

1. El Dilema del Prisionero

Aunque en el sentido lógico el Dilema del Prisionero no es estrictamente un dilema[1] es una matriz de pagos muy popular dentro de la Teoría de Juegos. En esta situación se tienen a dos individuos (A y B) que no se pueden comunicar entre sí inmersos en la siguiente problemática: Si A coopera con B pero B no, entonces a A se la paga T y a B con S, esto aplica también de forma recíproca para el caso en el que B coopera y A no. Si A y B cooperan el uno con el otro entonces ambos reciben de pago D, y si ambos se traicionan reciben de pago T.

Los pagos guardan la siguiente relación:

$$S < D < C < T$$
$$2D < S + T$$

Lo paradigmático del Dilema del Prisionero es que el pago por traicionar es mayor que el de cooperación mutua así como también el costo por ser traicionado es el más fuerte. Esto ha hecho que el Dilema del Prisionero se convierta en la *E. Coli*[2] de la Teoría de Juegos ya que muchos consideran, que modela

[1] En mi opinión personal sería más apropiado que se llamara la *la Paradoja del Prisionero*

[2] La Escherichia Coli es probablemente la bacteria más estudiada así como la más usada en la biología experimental

muy bien el comportamiento real cuando se tiene que decidir si cooperar o no cuando se tiene únicamente información local.

2. Mapeos autoorganizados

Los mapeos autoorganizados son redes neuronales artificiales en las que el entrenamiento se lleva a cabo en un proceso autoorganizado de forma no supervisada utilizando la regla hebbiana[20] de aprendizaje. Son una poderosa herramienta para hacer clusters ya que permiten hacer mapeos entre espacios de dimensión indefinida (comúnmente de una dimensión alta) hacia espacios de dimensión baja (comúnmente de dimensión dos) preservando la topología del espacio original. El entrenamiento se una red SOM se da de la forma que sigue:

1. Se selecciona la neurona que mejor se aproxime al estímulo presentado. Este proceso puede ser conceptualizado como un proceso de competencia entre todas las neuronas dentro del mapa. La neurona ganadora es denominada como Best Matching Unit (**BMU**).
2. Se modifican a las vecinas de la BMU conforme lo dicten el parámetro de aprendizaje así como la función de vecindad, de esta forma el mapa se va alterando gradualmente.
3. Se actualizan los valores de la función de aprendizaje y vecindario.

La función que resume la actualización de las neuronas del SOM en forma iterativa es:

$$m_i(t+1) = m_i(t) + \alpha(t)h_{ci}(t)[x(t) - m_i(t)]$$

Misma que indica que cada BMU m_i en cada iteración i altera el peso de sus vecinas a través el parámetro de aprendizaje α y la función de vecindad h_{ci}.

3. Función de Información Mutua

Las herramientas para correlacionar datos son muchas, entre ellas se escogió a la Función de Información Mutua para analizar los datos derivados de los modelos realizados. A diferencia de otras funciones de correlación la función de información mutua permite encontrar correlaciones no triviales que pudiera haber entre un conjunto de datos, aparte de que los símbolos en los sistemas a correlacionar pueden ser usados en el lenguaje en el que originalmente hubieran estado expresados. La forma de la función de información mutua es:

$$I(X,Y) = H(X) + H(Y) - H(X,Y)$$

donde la entropía de un sistema (H) está dada por

$$H(X) = -\sum_j^N p_i log p_i$$

y la entropía conjunta de X e Y ($H(X,Y)$ por

$$H(X,Y) = -\sum_i^N \sum_j^N p(x_i, y_i) log(x_i, y_i)$$

La implementación de esta función se puede encontrar en el Apéndice C

4. Modelación Basada en Agentes

La modelación basada en agentes es una técnica de simulación computacional en el que comúnmente se busca estudiar la interacción de los agentes que componen un fenómeno con su entorno [15]. Esta naciente técnica de simulación por computadora ha ganado popularidad desde hace 20 años, empleada en una diversidad de ámbitos como lo son la Física, Química, Biología, Economía, Antropología e incluso en la Industria Militar [7].

Esta técnica guarda cierta similitud con el paradigma conocido como programación orientada a objetos, muy socorrida dentro de las Ciencias de la Computación, y consta de tres componentes principales, los agentes, el entorno y las reglas que rigen la interacción entra cada uno de ellos.

Este paradigma requiere, como en todo proceso de modelación científica, analizar detenidamente el fenómeno a simular para identificar a las diferentes clases de actores dentro del fenómeno así como las características de cada uno de estas clases. Estas entidades o elementos capaces de interactuar entre sí reciben el nombre de *agentes* y un rasgo distintitvo de esta técnica es que no sólo se analiza el comportamiento individual de cada agente, sino como es que cada uno de ellos interactúa (o se comunica) con los demás agentes así como con el entorno en el que están desarrollándose.

Esta es la característica por la que la Modelación Basada en Agentes (MBA), como también se le conoce, es tan popular en la modelación de sistemas complejos por computadora.

Un ejemplo que exhibe como pueden funcionar los modelos basados en agentes en contraste con los modelos dinámicos es el modelo poblacional de Lotka-Volterra cuyo dinámica está descrita por las siguientes ecuaciones:

$$\frac{dx}{dt} = x(\alpha - \beta y)$$
$$\frac{dy}{dt} = -y(\gamma - \delta x)$$

en la que x es la población de la presa, en nuestro caso las liebres, e y es la población del predador, los lobos en nuestro caso. Las ecuaciones modelan la tasa de crecimiento de ambas poblaciones. Los parámetros se ajustan para reproducir observaciones específicas, estos pueden ser la tasa de natalidad, mortandad, y el ritmo en el que unos son capturados por los otros.

Este sistema de ecuaciones diferenciales traducidas a un modelo basado en agentes funcionaría de la siguiente manera:

Habría dos tipos de de agentes, los lobos y las liebres. Los lobos serían programados para *perseguir* a las liebres y en caso de estar a una distancia arbitrariamente pequeña las podrían *comer*, eliminando así una liebre de la población total. Estas a su vez serían programadas para *huir* de los lobos y para *alimentarse* del pasto que *nace* con alguna tasa de crecimiento del suelo. En caso de que un lobo no pudiése alimentarse después de alguna cantidad de tiempo, este *morirá* y de igual forma reduciría en uno la población de lobos.

En ambos casos, si dos agentes de la misma población están arbitrariamente cercanos se puede crear un individuo de la población correspondiente, para de esta forma aumentar la población de ambas especies. La reproducción de los agentes pueden estar ponderados con alguna distribución de probabilidad para regular el aumento y decenso de la población.

Cuando se tiene un fenómeno modelado por un sistema de ecuaciones y por un modelo computacional lo primero es esperar que los resultados que arrojen sean los mismos para ciertas condiciones iniciales. Pero la gran ventaja de los modelos basados en agentes sobre los sistemas de ecuaciones es la de encontrar comportamientos no capturados por e modelo matemático, ya que estos últimos se vuelven intratables dependiendo de la cantidad de detalle incorporado al sistema, es decir, son construidos con base en grotescos promedios [14] mientras que los realizados con agentes pueden funcionar a diversos niveles dependiendo del nivel de detalle con el que estén programados [7].

Pero aunque el campo del modelado basado en agentes tiene aproximadamente 20 años todavía no existe una definición ampliamente aceptada de lo que es un agente, de hecho no es raro encontrar definiciones muy concretas como la que ofrece Wooldrige [9]:

> "Un agente es un sistema computacional situado en algún entorno capaz de exhibir acciones independientes (autónomas) en este entorno ya sea en nombre de su usuario o propietario (determinando lo que se necesite llevar a cabo para satisfacer los objetivos definidos en su diseño, en vez de constantemente recibir instrucciones)".

A pesar de esto dentro de la comunidad de la inteligencia computacional existe un conjunto de características generalmente aceptadas que todos los agentes

deben cumplir [15],[2]:

Política Tienen definidos un conjunto de reglas o estrategias que determinan su actuar. Es común escuchar que a este conjunto de reglas se les refiera como *simples*[3]

Percepción Son capaces de percibir las características de su entorno, ya sea midiendo lo que sea que se encuentre en él o la existencia de otros agentes en una vecindad finita.

Autonomía No se estila que los agentes dependan de un control central para decidir como accionar e interactuar con el entorno u otros agentes. A cambio los agentes son capaces de alterar estados de variables internas definidas en su diseño para poder adaptarse al entorno.

Interacción Los agentes deben ser capaces de comunicarse con otros agentes mendiente el envío y recepción de mensajes, estos son usados para computar los objetivos que los agentes tengan definidos.

Es importante resaltar el hecho de que por estas características los modelos basados en agentes son muy socorridos para simular y así analizar fenómenos complejos así como las propiedades emergentes relacionadas con estos, esto se logra incorporando en la simulación una gran cantidad de agentes, al punto que superen cierta masa crítica, para después observar el comportamiendo de esta.

Uno de los modelos basados en agentes mas afamados y estudiados es el modelo de segregación de Schelling, este modelo exhiba la formación de agrupaciones humanas a partir de decisiones individuales muy sencillas.

Para esta parte del proyecto impartí un curso en el Posgrado de Contaduría para modelar sistemas complejos usando NetLogo[4] que constó de cuatro sesiones del 10 de Octubre al 25 de noviembre del presente.

4.1. El modelo de segregación de Schelling

Dentro de los modelos desarrollados con agentes[5], computarizados o no, hay una gran variedad muy interesante, muchos de ellos de investigadores con gran renombre como Robert Axelrod, John Holland, Robert Axtel, pero es el trabajo

[3] Ver Bonabeau, Eric, Agent-based modeling: Methods and techniques for simulating human systems, PNAS, 2002

[4] Ver `http://ccl.northwestern.edu/netlogo/`

[5] El concepto de agente ha venido evolucionando desde disciplinas como la Economía y Sociología

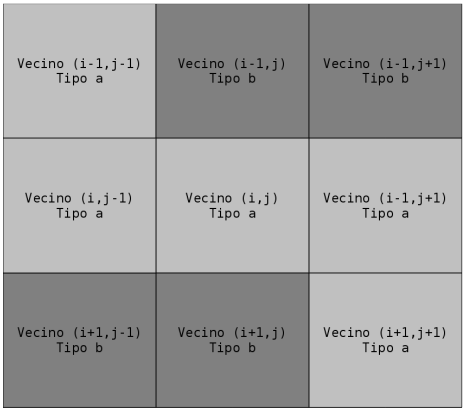


Figura 3.1: Vecindad de Moore de un vecino en el modelo de Schelling. Cuatro de sus vecinos pertenecen al mismo grupo, si su umbral de homofilia fuera mayor al 50 % se mudaría

de Thomas C. Schelling el que voy a destacar.

Thomas Schelling propuso en 1971 un modelo [18] para mostrar la segregación de grupos humanos. Este modelo está dentro de los primeros en los que se puede identificar la metodología de agentes.

El modelo, expuesto en su trabajo seminal *Dynamic Models of Segregation*, se basa en el principio de homofília como razón subyacente de segregación racial o concretamente para la formación de grupos sociales. Este simplificaba las razones por las que se forma barrio a partir de decisiones personales muy sencillas: Si la mayoría de mis vecinos pertenece a mi grupo entonces me quedo a vivir aquí, si no me mudo. El modelo se describe a continuación de una forma más explícita:

Las personas van a estar representadas por dos tipos de agentes los cuales debían *habitar* en una malla rectangular, donde cada uno constaba de ocho vecinos alrededor de él[6]. Cada agente va a procurar vivir rodeado por la mayor cantidad posible de agentes que también pertenezcan a su grupo. Es decir, en cada instante del programa cada agente va a contar la cantidad de vecinos que pertenezcan a su grupo, si esta cantidad sobrepasaba un cierto umbral predefinido de *satisfacción* entonces el agente opta por ya no mudarse más, si este umbral no se rebasa entonces el agente decide *mudarse* a otro punto en la malla. Una versión de este modelo en pseudocódigo puede verse en 5. Las cursivas en la explicación anterior tienen la intención de resaltar las analogías dentro del modelo.

En la figura 3.2 podemos ver como fueron disminuyendo la cantidad de ve-

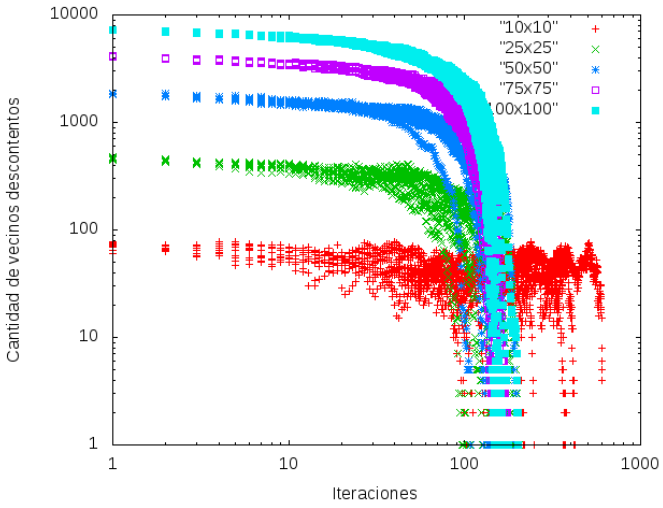
[6]En computación este tipo de vecindad recibe el nombre de vecindad de Moore

cinos insatisfechos para diferentes simulaciones del modelo de Schelling. Para dichas simulaciones se consideró un umbral de homofilia del 75 % y una densidad vecinal del 84 %. La cantidad de vecinos de cualquiera de los dos tipos era aproximadamente la mitad del total de vecinos y cada simulación se corrió 10 veces. Se fue variando la cantidad de la malla en tres tamaños distintos, 10x10,25x25, 50x50, 75x75 y 100x100. La implementación del código puede verse en el Apéndice A

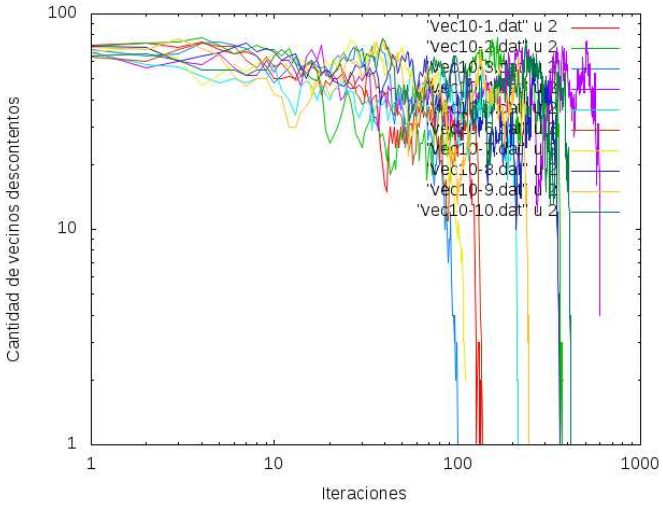
Hay varios aspectos que resaltar de este modelo. El primero es que este trabajo es un modelo basado en agentes. Es decir, este modelo ejemplifica el concepto y uso de los agentes como entidades que encapsulan un conjunto de propiedades así como un conjunto de reglas de interacción, y como es que, a partir una masa crítica de ellos interactuando, pueden dar lugar a un comportamiento complejo que no era predecible a simple vista.

Pero el aspecto que considero más importante y que tiene una gran importancia para la Sociología es el hecho de que este modelo a partir de una simplificación extrema de lo que puede ser una persona cualquiera se puede reproducir un comportamiento complejo observable a gran escala. A Schelling este modelo le valió el premio Nobel en Economía en 2005[7].

[7] Ver "The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel 2005". Nobelprize.org. 25 Jun 2010 `http://nobelprize.org/nobel_prizes/economics/laureates/2005/`.

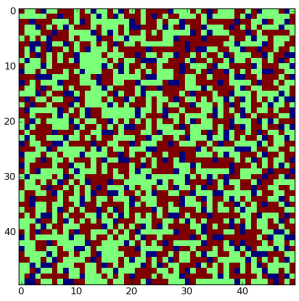


(a) Evolución de vecinos descontentos

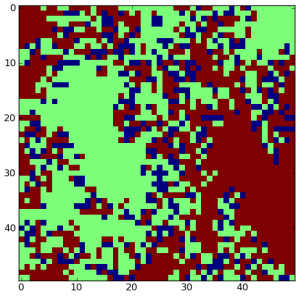


(b) Evolución de vecinos en 10x10

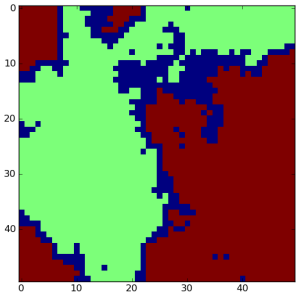
Figura 3.2: Evolución de la cantidad de vecinos descontentos en varias mallas. En (a) podemos ver la evolución para distintos tamaños de la malla, que van desde 10x10 hasta 100x100. En (b) vemos el caso exclusivamente para una malla de 10x10.



(a) Condición incial



(b) Etapa media



(c) Condición final

Figura 3.3: Diferentes etapas en la segregación vecinal en el modelo de Schelling en una malla de 50x50

Capítulo 4

Algoritmos Inspirados en Fenómenos Sociales

Como ya se mencionó en la introducción de este proyecto, la mayoría de los algoritmos de optimización han sido tomado de fenómenos en donde el ser humano está ausente. Pero esto no tiene porque ser así si consideramos la riqueza y adaptabilidad del comportamiento humano así como la sofisticación de los grupos sociales que este forma.

En esta sección se hizo una revisión de algoritmos inspirados en procesos sociales donde la optimización de la variable optimizada fuera una propiedad emergente derivada de la interacción no guiada de los agentes involucrados y también se presentó una modificación al algoritmo del SOM en su etapa de cooperación donde en vez de tener una cooperación irrestricta la estrategia de cooperación va a ser conforme lo dicta el paradigma del dilema del prisionero.

1. Aspectos generales sobre optimización

Una de las tareas principales dentro de las Ciencias de la Computación es la optimización de procesos, esta tarea en general tiene que ver con encontrar la mejor solución a un problema dado o visto de otra forma, por optimización se puede entender, desde el punto de vista matemático la minimización o maximización de alguna número variables sujetas a ciertas restricciones.

Desde el punto de vista ingenieril o computacional se entiende por optimización al proceso de maximizar el desempeño de un sistema o aplicación con la menor cantidad de recursos. Aunque ambas definiciones tienen que ver con el trabajo desarrollado en esta sección, la segunda está mucho más cercana a lo que entendemos por optimización.

Podemos decir, sin perdida de generalidad que optimización es encontrar el mínimo global de alguna función

$$f : \mathcal{X} \mapsto \mathcal{Y}$$

El nombre que recibe $\mathcal{X}$ es el de espacio de búsqueda y f está sujeta a un conjunto de restricciones que deben cumplirse simultáneamente, es decir, queremos encontrar un conjunto de puntos en el espacio de búsqueda para los cuales se cumpla

$$f(x^*) < f(x) \forall x \in \mathcal{X}$$

Entre las técnicas de optimización por computadora existen por lo menos dos categorías: exactos y estocásticos. Los métodos exactos, como A*, programación dinámica, etcétera, pueden ser muy útiles para problemas pequeños. Otra forma que pueden adquirir los métodos exactos puede ser la reexpresión del problema en un conjunto de ecuaciones para, que de forma analítica, encontrar la solución de la ecuaciones. Esta técnica ha derivado en una gran cantidad de algoritmos de optimización numérica [19].

Para problemas más complejos y grandes, que pueden ser del tipo NP-completos, la alternativa es únicamente con métodos estocásticos. Entre estos se encuentran la Inteligencia de Enjambre (Swarm Intelligence), el Recocido Simulado (Simulated Annealing), y los Algoritmos Genéticos. La teoría desarrollada alrededor de estos últimos es particularmente útil cuando se quiere entender la dificultad de buscar una solución en un espacio de soluciones muy grande e inexplorado, son precisamente los conceptos como *Epistásis*, *Paisaje Adaptativo*, lo que nos puede permitir comprender formalmente que tan difícil puede ser la búsqueda de soluciones de un problema.

En el contexto evolutivo se usa el concepto de paisaje adaptativo (*fitness landscape*) introducido originalmente por Sewall Wright en 1932 como una metáfora para describir dominios de atracción en dinámicas evolutivas, la idea de Wright era usarlos como una ayuda para visualizar el comportamiento de los operadores de selección y variación [3].

En el contexto de la optimización se usan para presentar una relación entre individuos (fenotipos) asociados con cierto éxito adaptativo (fitness). Cada uno de estos fenotipos representa a un candidato a solución de nuestra a función a minimizar y el valor del éxito adaptativo es que tan cercana u óptima es dicha solución. Entonces el problema de encontrar una solución óptima a nuestro problema se replantea como la búsqueda dentro de este paisaje al individuo que mejor adaptación presente [19].

A partir de varios escenarios planteados con el uso de los paisajes adaptativos se puede mostrar los distintos problemas con los que nos podemos encontrar cuando buscamos una solución en un espacio de búsqueda, sobre todo cuando

este puede ser muy grande y porque el problema de optimizar es en general una tarea técnicamente difícil [6].

De los problemas que pueden surgir cuando se busca una solución óptima y que quedan evidenciados a través del análisis de los paisajes adaptativos están el de converger a mínimos locales, esto es, que el algoritmo haya encontrado un fenotipo mejor en comparación con los que estuviese examinando y que exista otro con una adaptabilidad mejor en otro punto del espacio, es decir, que el algoritmo haya convergido prematuramente a un fenotipo que no es solución.

Otro problema puede ser el tener un espacio de búsqueda neutral, es decir, que todos los individuos en el espacio de búsqueda tengan la misma adaptabilidad asociada, por lo que la el seguir buscando individuos mejor adaptados parezca una tarea inútil.

Uno más es el tener un espacio de búsqueda corrugado donde el gradiente en el espacio de búsqueda exhiba valores muy cambiantes y esto hace que al algoritmo le sea muy difícil decidir hacia que regiones explorar para obtener una solución óptima [19].

El otro concepto que nos muestra porque es difícil la búsqueda de solución óptimas es el de *epistásis*. De modo muy general, la epistásis es la correlación que puedan tener varios genes en la expresión de algún rasgo fenotípico [3].

También es importante resaltar que la naturaleza estocástica de este tipo de algoritmos de optimización no garantizan encontrar la solución óptima como idealmente podría desearse por lo que únicamente podemos aspirar una solución cuasi-óptima en una cantidad razonable de tiempo.

Estas reflexiones inducen una pregunta ¿qué algoritmo de optimización usar para un problema dado? Es decir, parece implícita la existencia de un compromiso entre rapidez y exactitud con el tamaño del problema como parámetro de decisión, de hecho podría concluirse que no existe un método mejor que cualquier otro para todos los espacios de búsqueda. Wolpert y Macready formalizaron esto con sus teoremas No-Free Lunch[1] para búsqueda y optimización.

Lo que ellos encontraron es que sí un algoritmo a_1 califica mejor que a_2 para alguna función objetivo f_1 entonces existe una función f_2 en la que a_2 califica mejor que a_1. Más aún, en promedio, todos los algoritmos califican igual sobre todos los espacios de búsqueda [5].

Pero las estructuras sociales no humanas, como las de abejas, termitas u hormigas no han sido las únicas que han servido de inspiración para diseñar algoritmos que optimicen algo. También las estructuras sociales humanas también

[1] `http://www.no-free-lunch.org`

Estrategia	Descripción
C	Cooperativa, la neurona coopera siempre con sus vecinas
D	Desertora, la neurona nunca coopera con sus vecinas
T	Tit-for-tat (TFT), la neurona coopera con sus vecinas de la misma forma como las neuronas hayan cooperado con ella
R	Aleatoria, la neurona coopera o desierta con la misma probabilidad
A	Alternadora, la neurona coopera en la iteración i y desierta en la iteración $i+1$
M	La neurona interactuará con la interacción más frecuente
N	La neurona interactuará con la respuesta menos frecuente

Cuadro 4.1: Estrategias seguidas por las neuronas en SOM-NC

han sido fuente de inspiración de algoritmos de optimización, aunque no son las técnicas más populares en el campo de la computación.

2. SOM con estrategias no cooperativas

En el modelo desarrollado se hizo una modificación al algoritmo del SOM usando el Dilema del Prisionero como paradigma. En el SOM original se considera que la cooperación con las neuronas vecinas es irrestricta y con un decrecimiento constante.

Por cooperación estamos entendiendo la modificación local del mapeo realizado por la BMU con sus vecinas, pero ¿qué mapa se formaría si esto no fuera así? La situación más común en situaciones reales es de tal forma que no todos los vecinos cooperan irrestrictamente entre ellos, la dinámica de cooperación es mucho mas variada y sujeta a cambios por las condiciones en las que se encuentren los agentes.

En este modelo (SOM-NC) cada neurona cuando adopte el rol de BMU va a tener una estrategia asignada derivada del Dilema del Prisionero y que va a ser la guía para determinar de que forma coopera con sus vecinas. Es decir, la estrategia que siguen las neuronas en el algoritmo del SOM original es la cooperativa, es decir, en cada iteración modifican sus pesos de acuerdo con las neuronas vecinas.

El caso extremo a este sería la estrategia desertora donde las neuronas no van a cooperar en ninguna iteración con sus vecinas. Se definieron cuatro estrategias más explicadas en el Cuadro 4.1.

El experimento consistió en asignar a una red de 20x20 las estrategias del Cuadro 4.1 de manera aleatoria y se corrió el algoritmo SOM-NC para tres conjuntos de prueba. El proceso de mapeo tenía una salida de las épocas generadas

por el algoritmo donde cada entrada de la red correspondía a la estrategia seguida por cada neurona.

Para poder visualizar como evolucionaban las estrategias hice un script en python y matplotlib para presentar gráficamente las diferentes épocas generadas durante el entrenamiento del SOM, el programa puede ser visto en el Apéndice B. Este trabajo fue publicado en [11].

3. La sociedad como optimizadora

El ser humano se caracteriza, a diferencia de otros animales, por su gran capacidad para plantear de forma analítica los problemas que pueda tener y optimizarlos. De esta manera se han formalizado una gran cantidad de problemas en algoritmos que determinan la mejor solución (bajo algunos parámetros) que este pueda tener. Pero no son estos los algoritmos que se estudiaron en esta sección.

Se estudiaron todos aquellos donde la búsqueda no fuera un proceso guiado. Las razones para condicionar los algoritmos estudiados al hecho de que la optimización fuera una propiedad emergente es que de la interacción de agentes tan sofisticados como lo puede ser una persona (a diferencia de una hormiga o una suricata) puede dar pie a soluciones nuevas o simplemente mejores que las ya existentes al problema que este enfrentando.

Por otro lado se puede decir que es una aproximación más realista a la solución de problemas ya que generalmente no contamos con información global para solucionarlos sino que únicamente contamos con información de tipo local para computar, un ejemplo de esto es encontrar el precio más bajo en algún mercado.

También puede pensarse en términos técnicamente prácticos ya que puede resultar más sencillo abstraer e implementar el funcionamiento de cada agente para después dejar que la optimización surja por medio de la interacción entre una gran cantidad de estos contrario a lo complicado y específico que puede ser la implementación de alguna solución previamente conceptualizada.

4. Cómputo Emergente

Como ya habíamos mencionado, en esta parte del proyecto se hizo una investigación bibliográfica de aquellos algoritmos que estuvieran basados en fenómenos sociales y hacer un análisis de la complejidad computacional de estos. El trabajo fue reportado en [12] así como en la X Escuela de Biología Matemática llevada a cabo en Mazatlán, Sinaloa en octubre de 2008.

Los algoritmos inspirados en fenómenos sociales estudiados publicados en [12] fueron divididos en los siguientes temas: *Liderazgo*, *Formación de Alianzas*, *Optimización mediante etiquetado social*, *Delimitación y segregación de barrios*

4.0.1. Liderazgo

El liderazgo ha servido de inspiración para algoritmos de computación de la siguiente forma. En todas las estructuras sociales hay individuos que destacan en comparación a los demás integrantes del grupo, estos individuos muchas veces son modelo a seguir para los demás, tienen la característica de influir en el resto y su opinión en general tiene más peso.

Los líderes en un grupo atrapan la atención de algunos de los demás miembros del grupo,dependiendo de que tan bien les vaya en las tareas que se especialicen en realizar. Es común que estos seguidores copien opiniones o incluso algunos, si no es que todos, aspectos en el actuar del líder del grupo.

Es decir, la abstracción que se puede hacer de este fenómeno es suponer que los *líderes* sean la mejor solución (o por lo menos la más aproximada) en el espacio de soluciones de un problema dado y a través de compartir su posición en dicho espacio permitan a otros *individuos* migrar hacia la zona en la que se encuentren ubicados.

Algoritmo 1 Algoritmo de optimización por liderazgo

$t = 0$
Generar civilización $C(t)$ con N individuos uniformemente distribuidos en el espacio de parámetros.
Evaluar función objetivo en cada individuo junto con las condiciones a las que estén sometidos.
Construir *clusters* $S(t)$ como sociedades de $C(t)$.
Identificar al líder en cada sociedad $S(t)$
Migrar individuos en cada sociedad hacia la ubicación del líder más cercano.
Identificar líderes en la civilización $C(t)$ de los líderes de las sociedades $S(t)$
Migrar a los líderes de las sociedades hacia la ubicación de los líderes de la civilización
$t = t + 1$
if condición de paro **then**
  detenerse
else
  ir al paso 3.
end if

Otro algoritmo en el que la cooperación y la habilidad de seguir a individuos destacados quienes a su vez comparten su conocimiento con el resto de la pobla-

ción son las metáforas principales son los Algoritmos Culturales desarrollados por Robert Reynolds quien hizo una abstracción de como las culturas generan conocimiento, el algoritmo está dividido en tres etapas principales: la primera en la que se define de una forma general las zonas del espacio a explorar, la segunda en la que cada una de estas zonas se explora de forma mas detallada y finalmente una etapa cuando la búsqueda se queda estancada. Este algoritmo cuenta con una estructura denominada espacio de conocimiento.

En la investigación se presentó una modificación del Algoritmo Cultural combinado con Programación Evolutiva que ha mostrado resultados sobresalientes para optimización multiobjetivo.

Algoritmo 2 Algoritmo para optimización por liderazgo

Generar k individuos como población inicial.
Se evalua a la poblacion inicial.
Se inicializa el espacio de creencias.
while No se alcancen las condiciones de paro **do**
  Aplicar mutacion para generar a p hijos.
  Evaluar cada hijo.
  Obtener el desempeno relativo de cada solucion mediante mutaciones aleatorias
  Seleccionar q individuos con la mayor cantidad de victorias para generar una otra generacion.
  Agregar a los individuos no-dominados a una memoria externa.
  Modificar el espacio de creencias con individuos en la memoria externa.
end while

4.0.2. Formación de Alianzas

Las metas y objetivos que como seres humanos nos ponemos usualmente rebasan nuestras propias capacidades para alcanzarlas, precisamente por esto si tomamos como principio que estas metas y características las comparten otros conjunto de individuos podemos entonces conformar una alianza con aquellos que las compartan. Estas se caracterizan por tener una meta predefinida que justifica su formación así como una validez temporal. Por su puesto que esta formación de alianzas no queda únicamente en el nivel personal, sucede lo mismo para casi cualquier escala de grupo social (sino es que todos), desde familias, corporaciones, países, etcétera.

El problema puede ser visto en como encontrar un conjunto de particiones con los agentes que participan en el sistema de tal forma que todas la suma de los pagos a todas estas particiones sea máximo.

Algoritmo 3 Algoritmo para formación de alianzas

Construir todas las posibles alianzas $S_i(q)$ con q agentes.
while $S_i(q)$ no se vacíe **do**
  Contactar a los agentes $A_j \in S_i(q)$.
  Evaluar el beneficio de aliarce con A_j sujeto a preferencias y restricciones.
  Extraer q de $S_i(q)$ y compartir tareas.
  Si alguien es contactado por el agente A_k extraer a q así como las tareas comúnes.
end while

4.0.3. Optimización por etiquetado social

El proceso de clasificación es tan común en las Ciencias de la Computación como en en cualquier grupo social. En cualquier sociedad ocupamos etiquetas para clasificar a los individuos que pertenezcan a esta, no importa si estos reales o no el hecho es que ahí están e influencian y regulan la interacción que podamos tener con el poseedor de estas etiquetas. Particularmente estamos buscando el caso en el que podamos establecer una relación de cooperación con otro individuo.

En esta sección se presenta un algoritmo general basado en este fenómeno de etiquetado y que servirá de guía para determinar si podemos establecer una relación mutuamente cooperativa con algún otro agente al que no conocemos personalmente. Este algoritmo está pensado para un entorno como el de las redes P2P donde el esquema en el que uno coopera mientras que la contraparte muy frecuentemente opta por no hacerlo. Es decir, a través de un algoritmo basado en teoría de la cooperación se busca optimizar la eficiencia de una red P2P.

Algoritmo 4 Algoritmo para etiquetado social

Definir K número de generaciones
while No se alcance el número de generaciones **do**
  para cada agente i en la población:
  seleccionar agente adversario j con una etiqueta similar (siempre que sea posible).
  Las parejas i y j se enfrentan usando sus estrategias y obtienen su pago.
  Reproducir a estos agentes proporcionalmente a su pago
  Mutar etiquetas y estrategias en los agentes creados.
end while

4.0.4. Delimitación y segregación de barrios

La segregación y formación de ghettos es un fenómeno de una riqueza muy amplia en los grupos sociales humanos. Thomas Schelling desarrolló un mode-

lo ampliamente estudiado y usado que explica el fenómeno en que los agentes, originalmente distribuidos en una malla, determinan su permanencia o traslado dentro de la malla a partir de un sencillo conjunto de reglas que computan información extraída de los primeros vecinos de este mostrando un proceso de auto-organización.

Este algoritmo, con ciertas modificaciones, ha sido implementado para modificar dinámicamente la topología de redes P2P con el fin de mejorar el ancho de banda de estas. También, en un caso muy similar pero con distintas condiciones, ha servido para prevenir la aparición de concentradores en redes P2P y aumentar la cohesión de los usuarios de estas redes.

Algoritmo 5 Algoritmo de optimización por segregación

Determinar la fracción de vecinos que no son del mismo color al propio. (i.e. distancia al vector de rasgos F_i).
Si el agente esta conforme con los vecinos entonces se queda en su ubicación actual.
En caso contrario busca una posición libre más cercana que cumpla con sus requerimentos y se muda ahí.

Capítulo 5

Ciencias Sociales Computacionales

Como se mencionó en la sección 4 los usos más comunes de la computadora van desde máquina de escribir, una calculadora o una bodega de datos. Parece injusto, incluso antiético, exigir a las Ciencias Sociales fundamentar todos sus resultados de la misma forma en que se puede hacer en las Ciencias Formales y Naturales. En un experimento biológico, sea este con E. Coli, levadura o incluso ratas o cualquier otro animal de laboratorio se puede contar con una cierta cantidad de ellos (en el caso de bacterias, virus y hongos esta cantidad puede ser muy grande) para después hacer someterlos a un proceso en el que no es raro termine con el sacrificio de los sujetos del experimento.

Esto es impensable con un grupo de seres humanos, ya sea por razones prácticas o éticas. Otro impedimento para la experimentación dentro de las Ciencias Sociales es que le conferimos al ser humano una sofisticación y complejidad muy grande, la más grande dentro del mundo natural de hecho. Tal vez esta sea la razón por la que las Ciencias Sociales hayan optado modelar toda la realidad en su conjunto.

Sin embargo, los modelos científicos se caracterizan por presentar una versión simplificada de la realidad. Por modelo científico vamos a entender una representación incompleta de la realidad que percibimos, y es justamente esta cantidad finita de características lo que va a delimitar las condiciones dentro de las cuales el modelo es válido. Para que este sea aceptado debe evolucionar de una forma suficientemente cualitativamente próxima al fenómeno observado.

Otra característica que hay que mencionar es el hecho de que estos requieren una cantidad menor de recursos de los que usa el fenómeno para transitar entre su estado inicial y final, siendo uno de estos el tiempo que le toma evolucionar. Una aspecto que se deriva de lo anterior es que los modelos científicos evolucio-

nan más rápido que el fenómeno observado.

Es precisamente está característica de la que pueden echar mano los sociólogos al reproducir o querer explicar la fenomenología relacionada con los sistemas sociales mediante otro tipo de uso de la computadora, claro que esto implica inevitablemente el acotamiento de la realidad a reproducir.

En esta sección se realizó en un modelo de votaciones usando un mapeo auto-organizado modificado para simular la influencia que cada votante podría tener sobre sus vecinos.

1. Modelo de Preferencias electorales usando SOM

En el modelo desarrollado se considero el siguiente caso. El resultado de una elección es comúnmente atribuido a un cuidadoso proceso de razonamiento en los votantes quienes evalúan las opciones presentadas por los candidatos para la mejor. Es común considerar a los votantes como agentes racionales que toman sus decisiones a través de un proceso interno y casi nunca se considera la influencia de las interacciones con otros agentes.

Con este enfoque, en el que las interacciones con otros votantes no es despreciable, el fenómeno se convierte en un proceso dinámico que varía conforme se acerca el día de la elección. Esta evolución está sujeta a varios factores, como pueden ser la permeabilidad de opinión en cada cada vecino, el rango de interacción de cada uno de ellos, la intensidad y duración de la campaña política, etcétera.

Considerando que cada votante puede influenciar y ser influenciado por sus vecinos, donde cada uno de estos está descrito por un vector en el que cada entrada define una posición sobre una amplia gama de temas y que esta capacidad de influenciar decrece con el tiempo es un esquema que coincide con el funcionamiento de los mapeos autoorganizados, mismos que fueron explicados en la sección 2, donde cada votante estaría representado cada una de las neuronas en la red, su capacidad de influenciar a sus vecinos esta representado por la etapa de cooperación en el algoritmo del SOM.

El proceso de mapeo se va a llevar a cabo entre un espacio en el que están descritos las diferentes características que se consideraron que mejor describen a un votante a nuestro mapa de dimensión dos con el fin de reproducir fenómenos en los que regiones enteras votan por un sólo partido (ver fig. 5.1. El algoritmo del SOM funcionará como el análogo a la la campaña política permeando entre las neuronas las diferentes posiciones políticas.

Cada simulación se corrió suficientes veces con mallas de tamaños que iban

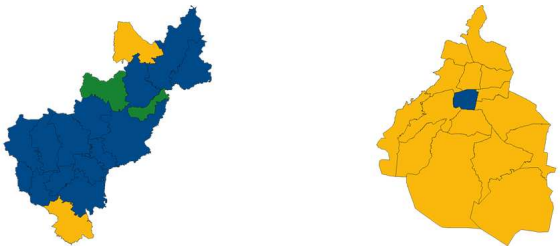


Figura 5.1: Regiones enteras que votan por un sólo partido

entre 5,10,20,30,100. Las variables consideradas están descritas en la Tabla 5.1. Para determinar la calidad de los mapas obtenidos se uso el error topográfico del mapa (TE). Este fue el parámetro de control contra el cual se hizo cada una de las correlaciones.

Mi labor fue analizar los datos al final del proceso de mapeo para calcular la información mutua entre variables consideradas. Los parámetros de control sometidos al análisis de la Función de Información Mutua para determinar correlaciones relevantes están descritos en la Tabla 5.1.

Calculé la información mútua sobre cada una de estas variables y el error topográfico y después por pares de estas variables contra el error topográfico.

Este trabajo fue publicado en [13] y también fue presentado en el 1er Congreso de Ciencias de la Complejidad.

Param. Control	Descripción
H	Vecindad inicial
B	Cantidad de BMU's para cada estímulo
E	Cantidad de épocas
DIM	Dimensión del espacio de estímulos
P	Cantidad de estímulos de entrada
ρ	Diferencia promedio entre los partidos
γ	Diferencia promedio incial entre votantes

Cuadro 5.1: Estímulos que a los que fueron expuestos las neuronas en el modelo de preferencia electoral

Capítulo 6

Resultados

En esta sección se presentan los resultados de las distintas etapas del proyecto en el mismo orden en el que fueron presentadas en el desarrollo del presente reporte.

1. SOM con estrategias no cooperativas

Una vez obtenidas las épocas a de SOM-NC se obtuvieron la secuencia de épocas para varias de ellas.

En la Figura 6.1 se muestra la distribución de las estrategias para una malla de 20x20 de acuerdo con la codificación de colores especificada en la Tabla 6.1 para el entrenamiento del conjunto de prueba anillo para las épocas 1, 2, 3, 49 y 50. Este experimento consistió en comenzar con una todas las neuronas con estrategia cooperativa, y esa es la razón por la que toda la red aparece en azul oscuro, conforme evoluciono el modelo, las estrategias iban mudando conforme un parámetro de mutación a los estados mostrados en las siguientes figuras. Es importante recalcar el hecho de que a partir de la época 30 la distribución de estrategias alcanzó un punto estable y ya no hubo ninguna mutación a partir de esto. Esto fue un fenómeno observado en la mayoría de los experimentos, la evolución del modelo a un estado estable independientemente del mapeo formado. Lo que se puede concluir es que la cooperación irrestricta no es necesaria en la formación de mapeos auto-organizados.

2. Algoritmos Inspirados en Fenómenos Sociales

De los diferentes algoritmos analizados podemos decir lo siguiente:

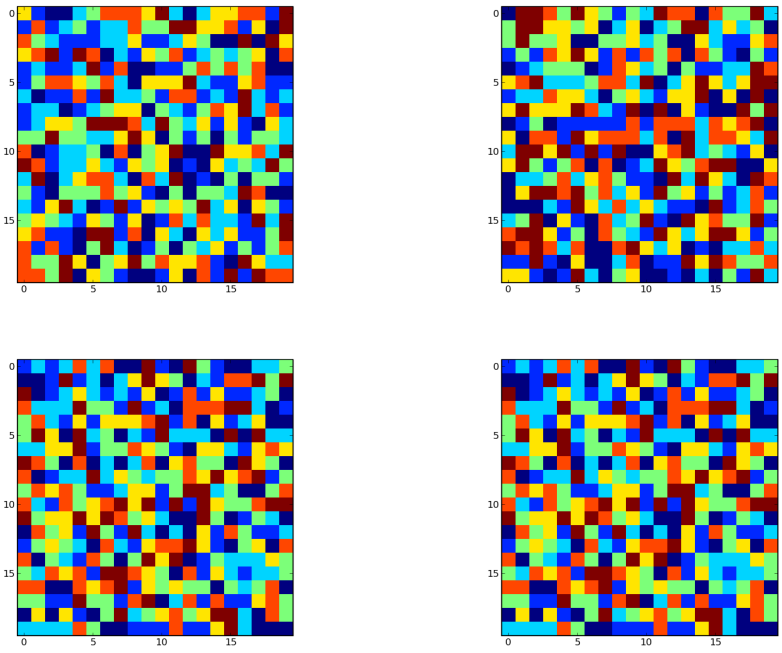


Figura 6.1: Evolución de épocas para el SOM No Cooperativo

2.1. Liderazgo

Sean T la cantidad de iteraciones que se ejecuta el algoritmo, $S(t)$ la cantidad de civilizaciones creadas y $f(S(t))$ la función que identifica a los líderes en cada una de ellas. La complejidad asociada a este algoritmo es:

$$T * O(f(S(t)) * log(S(t)))$$

2.2. Algoritmo Cultural

Dado que se generan k individuos originalmente. Sea $f(i)$ la función de evaluación del individuo i y M la cantidad de generaciones creadas. Los procesos más costosos dentro de este algoritmo son la evaluación de la población inicial que es $kO(f(i))$ mas el costo por la evaluación de cada individuo en el algoritmo principal que es $M * O(f(i))$, en total el costo del algoritmo es de:

$$k * O(f(i)) + Mk * O(f(i))$$

que equivale a

$$kO(f(i))(1 + M)$$

Estrategia	Color
Cooperativa	Azul oscuro
Desertora	Azul claro
Aleatoria	Cían
Alternadora	Verde
Mas probable	Naranja
Menos probable	Rojo
Tit-for-tat	Amarillo

Cuadro 6.1: Codificación de colores para las estrategias

2.3. Formación de Alianzas

Probablemente el algoritmo más costoso dentro de los presentados ya que no existe una solución polinomial para la construcción del conjunto potencia y la complejidad algorítmica de esta construcción es

$$O(2^N)$$

Considerando que el resto del algoritmo no tiene en total un costo tan elevado como para ser comparativamente significativo hace que la anterior cota sea el costo del algoritmo presentado. La implementación de este algoritmo requiere una heurística que muy probablemente esté determinada por el problema particular con el que se esté tratando.

2.4. Optimización por etiquetado social

Sea M el número de generaciones que se ejecuta el algoritmo, $f(p)$ la función de selección de par de interacción y N el número total de individuos en la población al final de la ejecución del algoritmo. El proceso de selección de pares para cada uno de los agentes es la parte del algoritmo más costosa y su cota es:

$$MN * O(f(p))$$

2.5. Segregación

Sea M la cantidad de individuos en la población. Las implementación misma del algoritmo involucra un umbral de satisfacción, mismo que si fuese muy elevado haría que todo el sistema nunca llegue a una configuración estable y por lo tanto el algoritmo nunca terminaría. Si este no fuera el caso la cantidad de reubicaciones de los agentes sería cada vez menor de forma logarítmica.

$O(logN)$

La complejidad computacional de los algoritmos derivados se encuentra resumida en la Tabla 6.2.

Algoritmo	Complejidad
Liderazgo	$T * O(f(S(t)) * log(S(t)))$
Algoritmo cultural	$kO(f(i))(1 + M)$
Formación de alianzas	$O(2^N)$
Optimización por etiquetado social	$MN * O(f(p))$
Segregación	$O(logN)$

Cuadro 6.2: Complejidad computacional de algoritmo inspirados en fenómenos sociales

Se puede observar que en general la complejidad computacional de los algoritmos estudiados presenta un cota bastante aceptable considerando que los problemas para los que están considerados son del tipo NP o NP-completos.

3. Modelo de Preferencias electorales usando SOM

Utilizando la función de información mutua se correlacionaron los datos obtenidos por el algoritmo SOM-VOT para los varios experimentos (mallas de 5,10,20,30,100) sobre las variables especificadas en la Tabla 5.1 contra el error topográfico en todos los casos. La cantidad de estados de este último era de 5. Los parámetros de control que mostraron una correlación interesante fueron B (cantidad de BMUs) discretizado en 4 estados, E (cantidad de épocas) con 7 estados, H (tamaño de la vecindad inicial) con 4 estados, P (cantidad de estímulos de entrada) con 4 estados, DIM (dimensión del espacio) con 5 estados, así como la correlación de información mutua TE con *Ro* y *Gamma*.

La información mutua que cada uno de estas variables presentaba contra contra el error topográfico se muestra en la Figura 6.2. Un resultado particularmente de este trabajo se puede ver en 6.2f en la que resalta una realación entre ρ (distancia promedio en la posición política entre partidos) y γ (distancia inicial promedio entre vecinos).

Los otros resultados que mostraron una clara correlacion con el error topográfico fueron la cantidad de épcoas (figura 6.2a) lo que computacionalmente significa que se llegó a un buen mapa topográfico después de una gran cantidad

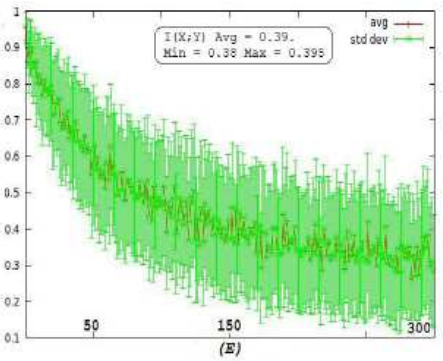
de épocas y respecto del modelo puede ser interpreteado como que si una campaña es lo suficientemente larga entonces toda una zona votará por un candidato.

Otra variable que presentó mucha correlación a lo largo de los distintos experimentos fue el tamaño de la vecindad inicial (figura 6.2c) lo que puede ser interpretado como la extensión que cada votante tiene sobre sus vecinos.

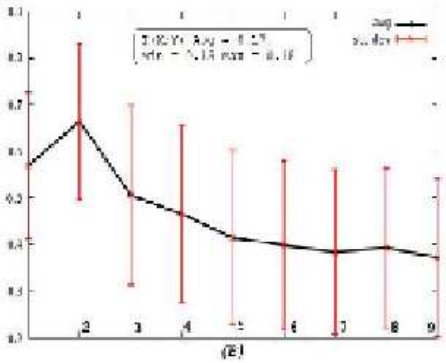
La otra variable a destacar es la cantidad de estímulos de entrada, P (figura 6.2e) lo que puede ser interpretado como la cantidad de partidos a los que están expuestos los ciudadanos.

En un modelo de reciente publicación Nathan Collins [4] hace un aporte en el que incorpora la capacidad de los votantes para no votar por ningún partido en un sistema bipartidista, a diferncia de la perspectiva usual en la que los votantes eligen uno de dos partidos únicamente, cómo el mismo Collins apunta. El objetivo principal del trabajo de Collins es mostrar los cambios de partido en escalas grandes de tiempo.

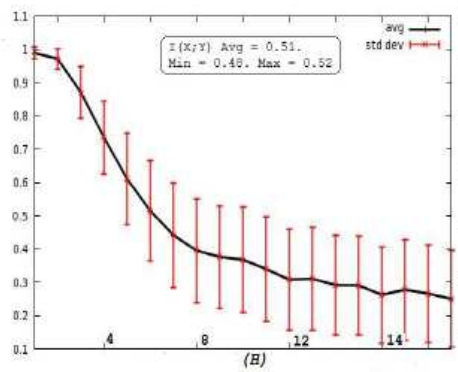
Aunque en este modelo los *apáticos* no fueron considerados como un grupo respresentativo, incluirlos no supone ningún problema, ya que a diferencia del trabajo de Collins el cuál está basado en un sistema de flujos de votantes representados en un sistema de ecuaciones diferenciales donde agregar grupos y dinámicas puede complicar encontrar una solución analítica, la herramienta usada para este modelo se basa en un principio de autoorganización donde la incorporación de grupos nuevos resulta muy sencilla.



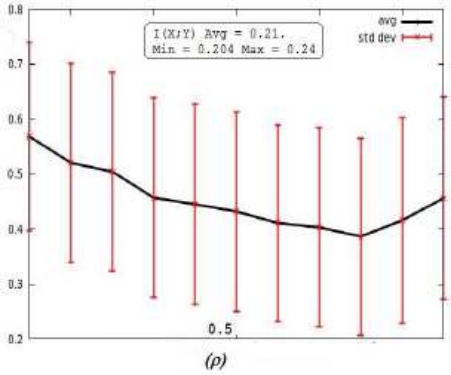
(a) Información mutua para E



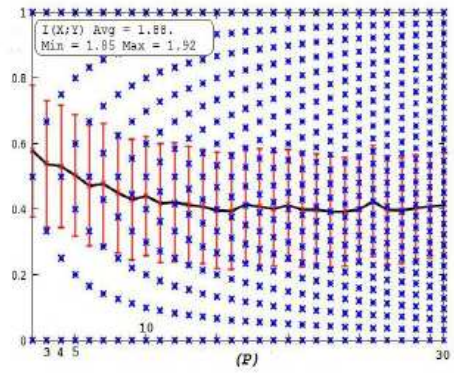
(b) Información mutua para B



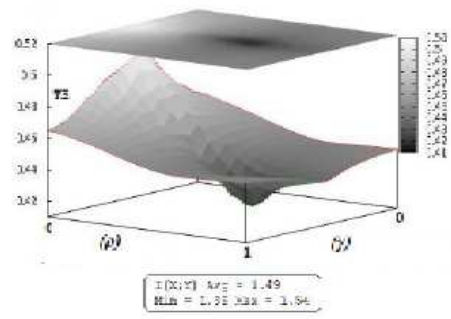
(c) Información mutua para H



(d) Información mutua para DIM



(e) Información mutua para P



(f) Información mutua para todas las variables

Figura 6.2: Correlación en el algoritmo de SOM-VOT para una sola variable

Capítulo 7

Conclusiones

En el presente proyecto se exploró una forma innovadora para hacer investigación en el campo sociológico usando la computadora más allá del trivial uso que ahora se le da. Al utilizar la computadora como un laboratorio donde se pueden implementar modelos de procesos sociales y así explorar una amplia gama de posibilidades en las que se puedan ver involucrados.

La motivación principal para trabajos como el aquí presentado es el de contribuir a la investgiación en Sociología (aunque no tiene que quedarse restringido en este campo) ya que acutalmente mucha de ella se basa en una gran cantidad de estudios estadísticos seguida de la interpretación del investigador a cargo. Pero bien sabido es que correlación no implica causalidad.

Las herramientas con las que contamos hoy para hacer investigación científica, particularmente las que se presentaron a lo largo de este proyecto, nos permite incorporar estos datos y poner a prueba hipótesis y así proveer de un sustento incontrovertible sustentado en un modelo repetible la interpretaciones y conclusiones a las que se lleguen dejando de lado cualquier carga especulativa, o por lo menos la mayor posible.

Otra consecuencia que se deriva de este trabajo es que los campos de conocimiento que se interrelacionan para resolver problemas de la vida cotidiana son muchos, es decir, para resolverlos deben ser abordados de formas muy diversas y que, por lo tanto, intentar atarclos con la lente de una sola disciplina es a todas luces insuficiente.

Los diferentes campos de conocimiento en los que ahora podemos y debemos incursionar implican dos cosas, conocer desde niveles muy tempranos (preparación universitaria) muchas otras ramas de conocimiento y una forma de establecer un lenguaje común. La otra es que inevitablemente tenemos que aprender a realizar trabajo interdisciplinario de manera óptima. En este par de aspectos el papel de las matemáticas y la computación es crucial.

En nuestro caso particular, es cierto que para que exista una comunicación más fluida entre sociólogos y computólogos es necesario que los primeros puedan expresar sus ideas en un lenguaje que los segundos puedan entender, así como los segundos conozcan la problemática de los primeros. Pero este último problema es mucho más sencillo de resolver que el primero, ya que se ya que el plasmar las ideas en un programa requiere, por lo menos, de cierto grado de experiencia.

La herramienta más cercana a los sociólogos, quienes no necesariamente poseen un gran entrenamiento técnico en el uso de la computadora, es la Modelación Basada en Agentes misma que aunque todavía está en desarrollo, o que tiene algunas limitaciones técnicas se encuentra en un estado en el que facilitaría el poder plasmar modelos que fueran generadores de hipótesis y así guiar la subsecuente investigación.

El caso en el que las Ciencias Sociales sirven de inspiración para extraer algoritmos computacionales. La hipótesis principal que subyace a este razonamiento es el hecho de que tomando en cuenta el alto grado de sofisticación y complejidad de abstracción del pensamiento humano pueda implicar una capacidad de cómputo mayor que la que se tiene en los otros algoritmos inspirados en estructuras sociales.

Otra razón que está implícita en los fenómenos analizados es la robustez que pueden presentar. Esto se debe en buena medida a la adaptabilidad de los seres humanos que estaban involucrados en el fenómeno abstraído. Aunque esto es más difícil de plasmar en un algoritmo.

En general este es el punto que se puede concluir de este trabajo, el que exista un punto común entre sociólogos y computólogos en el que ambas disciplinas se vean mutuamente beneficiadas de sus áreas de estudio.

Apéndice A

Implementación del Modelo de Schelling

La implementación del algoritmo de Schelling se hizó en Python ver 2.5. El programa está dividido en dos partes principales, la primera la clase malla que implementa las funciones del entorno donde los agentes van a interactuar, le da la estructura al modelo y contiene métodos para determinar una vecindad dado un punto, así como los vecinos que esten establecidos en esa vecindad.

La segunda parte es la clase vecino que implementa las funciones propias de los habitantes de la malla, como pertener a un grupo, determinar si están o no satisfechos dependiendo de los valores definidos en algún momento de su uso, etcétera.

```
#!/usr/bin/python
#-*- coding: utf-8 -*-

import sys
from random import randint
from math import floor

def hazmalla(I,J,ocup=0):
    """malla de IxJ donde todas estan vacias"""
    m = []
    for i in xrange(int(I)):
        m.append([ocup]*int(J))
    return m

class malla:
    def __init__(self, I,J,cerrado=True):
        """constructor, crea un mundo de IxJ
        si cerrado = False ent el mundo es un toroide"""
```

```
    self.Xmax = I-1 #maxima coordenada existente
    self.Ymax = J-1 #maxima absisa existente
    #print "malla: "+ str((len(self.G),len(self.G[0])))
    self.cerrado = cerrado
    self.W = dict()
    self.libres = set()
    for i in range(I): #vamos a mantener control con este diccionario
        for j in range(J):
            self.libres.add( (i,j) )
def __str__(self):
    r = ""
    for i in range(self.Xmax+1):
        for j in range(self.Ymax+1):
            if self.ocupada(i,j):
                r += str(self.W[i,j].grupo)+" "
            else:
                r += "0 "
        r += "\n"
    return r
def impids(self):
    r = ""
    for i in range(self.Xmax+1):
        for j in range(self.Ymax+1):
            r += str(self.coord(i,j))+"\t"
        r += "\n"
    return r
def coord(self,i,j):
    """regresa el contenido de la coordenada (i,j)"""
    try: self.W[i,j]
    except: 0
def libre(self,i,j):
    """dice si (i,j) esta o no libre"""
    if self.libres.__contains__((i,j)): return True
    else: return False
def c_libre(self):
    """regresa alguna coordenada libre """
    return self.libres.pop()
def ocupa(self, c, v):
    self.W[c] = v
def ocupada(self,i,j):
    """determina si la celda i,j esta ocupada"""
    #print "M.ocupada: ",str(i)," ",str(j),
    #ToDo: checar que i,j no pasen los limites
    #if self.coord(i,j) == 0: return False
    #else: return True
    return not self.libre(i,j)
```

```
def muda(self, v, xf, yf):
    """aloja un vecino en la coordenada i,j"""
    #ToDo: checar que i,j no pasen los limites
    del( self.W[(v.x,v.y)] ) #residencia anterior ahora es tuple
    self.libres.add( (v.x,v.y) )
    self.ocupa((xf,yf),v) #residencia nueva tiene al vecino
    v.defdir(xf,yf)
def vecindad(self, i,j):
    """devuelve la vecindad alrededor de i,j"""
    D = set()
    if self.cerrado and ((i==0) or (j==0) or (i==self.Xmax) or
                         (j==self.Ymax)):
        P = set()
        esq1 = set([(i+1,j),(i+1,j+1),(i,j+1)]) #(0   ,   0)
        esq2 = set([(i-1,j),(i-1,j+1),(i,j+1)]) #(Xmax,   0)
        esq3 = set([(i,j-1),(i+1,j-1),(i+1,j)]) #(0   ,Ymax)
        esq4 = set([(i,j-1),(i-1,j-1),(i-1,j)]) #(Xmax,Ymax)
        if i==0 and j==0:
            P = esq1
        elif i==0 and j>0:
            if j==self.Ymax:
                P = esq3
            else:
                P = esq1.union(esq3)
        elif i==self.Xmax and j==0:
            P = esq2
        elif i==self.Xmax and j>0:
            if j==self.Ymax:
                P = esq4
            else:
                P = esq2.union(esq4)
        elif i>0 and i<self.Xmax:
            if j==0:
                P = esq1.union(esq2)
            elif j==self.Ymax:
                P = esq3.union(esq4)
        for p in P:
            D.add( (p[0],p[1]) )
    else: #dentro de la malla o esquinas toroidales
        for m in [-1,0,1]:
            for n in [-1,0,1]:
              D.add((((i+m)%(self.Xmax+1)),((j+n)%(self.Ymax+1))))
        D.remove( (i,j) )
    return D
def vecinos(self, i,j):
    """regresa los vecinos y coordenadas vacias alrededor de
```

```
        la vecindad i,j """
        V = self.vecindad(i,j)
        D = set()
        for v in V:
            if self.ocupada(v[0],v[1]):
                D.add( self.W[ (v[0],v[1]) ] )
        return D
    def c_coord( self, i, j, dx, dy ):
       """regresa la coordenada (i+dx,j+dy)"""
       nx = ny = 0
       if self.cerrado and ( ((i+dx) > self.Xmax) or
                             ((j+dy) > self.Ymax) or
                             (i+dx < 0) or (j+dy<0) ):
           if ((i+dx) > self.Xmax):
               dx = self.Xmax - i
           if ((j+dy) > self.Ymax):
               dy = self.Ymax - j
           if ((i+dx) < 0):
               dx = -i
           if ((j+dy) < 0):
               dy = -j
           nx = i+dx
           ny = j+dy
       else:
           nx = (i+dx)%(self.Xmax+1)
           ny = (j+dy)%(self.Ymax+1)
       return (nx,ny)

class vecino:
    def __init__(self,i,j,col,afinidad):
        """constructor  que recibe las coordenadas
        el color (grupo) en forma de un k >= 1
        y el umbral de homofilia, tal que 0 < afinidad < 1
        bajo el entendido de que si vecinos > th entonces el vecino
        esta satisfecho con su ubicacion"""
        self.umbral = afinidad
        self.x = i
        self.y = j
        self.grupo = col
        self.id = str(col)+":"+str(i)+":"+str(j)
        self.mudanzas = 0
    def __str__(self):
        return self.id
    def cuenta_vec(self,M):
        """cuenta vecinos y regresa el porcentaje
        de cada uno dependiendo del grupo del actual"""
```

```
        vec = M.vecinos(self.x, self.y)
        d = dict()
        for v in vec:
            if v.__class__.__name__ == 'vecino':
                d[0] = d.get(0,0) + 1 #total de vecinos existentes
                d[v.grupo] = d.get(v.grupo,0) + 1
        for i in range(1,len(d)):
            d[i] = d.get(i,0) / float(d[0])
        return d
    def contento(self, M):
        """determina si este habitante esta contendo
        en la celda en que se encuentra dado el umbral
        definido en el constructor, recibe la malla en la
        que se este trabajando"""
        d = self.cuenta_vec(M) #obtenemos los porcentajes de los grupos
        prc = d.get(self.grupo,0) #recuperamos el porcentaje de iguales
        if prc >= self.umbral: return True
        else: return False
    def mueve(self,M):
        """caminante aleatorio entre las celdas vecinas con
        probabilidad 1/cardinalidad_vecindad"""
        mc = M.c_libre()
        M.muda(self, mc[0], mc[1])
    def defdir(self,i,j):
        """reubica el vecino actual a la coordenada
        i,j especificada"""
        self.x = i
        self.y = j
        self.mudanzas += 1


def establece_vecinos(I,J,K,ro,um):
    """proc. inicial para establecer a los
    vecinos en la malla con un umbral um
    se establecen 50% de un tipo (1) y 50% del otro (2)
    y no deben sobrepasar la densidad establecida con ro"""
    if ro >= 1 or ro < 0:
        raise Exception("ro debe ser entre 0 y 1: "+str(ro))
    elif I*J*ro < K:
        raise Exception("demasiados vecinos: "+str(ro)+", "+str(um))
    else:
        todos = list()
        M = malla(I,J)
        for k in xrange(K):
            c = M.c_libre()
            v = vecino(c[0],c[1],randint(1,2),um)
```

```
            #print k, c," ",
            todos.append(v)
            M.ocupa(c,v)
        return [M,todos]

def analiza(M,V):
    """recorre el mundo para determinar quienes estan contentos o no"""
    U = set()
    for v in V:
        if not v.contento(M):
            U.add(v)
    return U

def iter_segregacion(M,V,infelices,vecindarios):
    k = 0
    while len(infelices)>0:
        vecindarios.append(str(M))
        #print "k: ",k, " #infelices: ", len(infelices)
        inf = infelices.pop()
        inf.mueve(M)
        k += 1

def Main(I,J,K,ro,th,sgen,sevol):
    evol_inf = list()
    vecindarios = list()
    fgen  = open(sgen,'w')
    fevol = open(sevol,'w')
    print "Estableciendo vecinos"
    [M, V] = establece_vecinos(I,J,K,ro,th)
    vecindarios.append(str(M))
    print "Algoritmo"
    infelices = analiza(M,V)
    while ( len(infelices) > 0 ):
        if len(vecindarios) >= 20:
            print "Imprimiendo vecindarios en " + sgen
            for i in range(len(vecindarios)):
                fgen.write(vecindarios[i]+"\n")
            fgen.flush()
            vecindarios = list()
        evol_inf.append(len(infelices))
        iter_segregacion(M,V,infelices,vecindarios)
        infelices = analiza(M,V)
    print "Imprimiendo historia de descontentos en " + sevol
    for i in range(len(evol_inf)):
        fevol.write(str(i+1)+"\t"+str(evol_inf[i])+"\n")
    print "Cerrando archivos"
```

```
    fgen.close()
    fevol.close()
    print "Terminado"

if __name__ == "__main__":
    I = sys.argv[1]       #long malla X
    J = sys.argv[2]       #long malla Y
    K = sys.argv[3]       #cant vecinos
    ro = sys.argv[4]      #densidad de vecinos
    th = sys.argv[5]      #umbral de afinidad
    sgen = sys.argv[6]
    sevol = sys.argv[7]

    Main(int(I),int(J),int(K),float(ro),float(th),sgen,sevol)
```

Apéndice B

Animación de Matrices

Este programa toma como entrada una serie de matrices en forma de enteros para mostrarlas secuencialmente en la pantalla. Fué programado con Python 2.5 y Matplotlib.

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

##imports para pintar vecindades
import sys, getopt
import time
import gobject
import gtk
import matplotlib
matplotlib.use('GTKAgg')
from matplotlib import rcParams
from pylab import *

##funciones
def abre(F,modo='r'):
    try:
        f = open(F, modo)
        print f
        return f
    except IOError:
        print "No se pudo abrir el archivo ",F
        exit(1)

def cierra(f):
    if not f.closed:
        f.close()
    print "Flujo cerrado"
```

```
def carga_matrices(f):
    M = []
    linea = f.readline()
    while linea != "":
        if linea != NVALINEA:
            m = []
            while linea != NVALINEA:
                ##print linea
                ##print "Tabajando matriz numero ", i
                m.append(map(int, linea.split()))
                linea = f.readline()
            ##print m
            M.append(m)
        else:
            linea = f.readline()
            ##print "linea nueva", linea
    return M

def pintavec(*args):
     global idx
     im.set_array(M[idx])
     manager.canvas.draw()
     idx += 1
     if idx == len(M):
          print 'FPS', idx/(time.time() - t0)
          return False
     return True



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


ARCHIVO = "ejpel"
NVALINEA = "\n"
INTERP = "nearest"

optlist, args = getopt.getopt(sys.argv[1:], "f:i:")

for arg in optlist:
    opcion = arg[0]
    valor = arg[1]

    if opcion == "-f":
        ARCHIVO = valor
```

```
    elif opcion == "-i":
        INTERP = valor

    else:
        print "Opcion no reconocida"
        exit(1)


M = [] ##matriz me matrices
print "Abriendo archivo ", ARCHIVO
f = abre(ARCHIVO)
print "Cargando matrices"
M = carga_matrices(f)
print "Archivo: ", ARCHIVO
print "Interpolacion:", INTERP

fig = figure(1)
a = subplot(111)

im = imshow(M[0], cmap=cm.jet, interpolation=INTERP)
manager = get_current_fig_manager()
idx = 0
t0 = time.time()

gobject.idle_add(pintavec)
show()

cierra(f)
```

Apéndice C

Función de Información Mutua

La función de información mutua fue implementada en Python 2.5 y recibe dos listas de datos y regresa el valor de la información mútua entre ambas.

```
#!/usr/bin/python2.5
#-*- coding:utf-8 -*-

from math import log

def maxl(L):
"""devuelve el máximo de una lista"""
m=-1e10000
for i in range(L):
if L[i] > m: m=L[i]
return m

def minl(L):
"""devuelve el mínimo de una lista"""
m=1e10000
for i in range(L):
if L[i] < m: m=L[i]
return m

def lee(f):
"""
lee el archivo con un par de columnas que representan
los dos sistemas a analizar

regresa [X, xmax, xmin, Y, ymax, ymin]: serie X max y
```

```
        min en X e Y max y min en Y
"""
X = []
Y = []
xmin = ymin = 1e10000
xmax = ymax = -1e10000
try:
r = open(f,"r")
except:
print "No se pudo abrir ", f

mas = True
l = r.readline()
while mas:
if l == '':
mas = False
break
else:
xi = float(l.split()[0])
  if xi > xmax:
xmax = xi
if xi < xmin:
xmin = xi
X.append(xi)
yi = float(l.split()[1])
if yi > ymax:
ymax = yi
if yi < ymin:
ymin = yi
Y.append(yi)
l = r.readline()

r.close()
return [X, xmax, xmin, Y, ymax, ymin]

def lee_ventana(x, tvent=0):
"""
se lee el mismo archivo que contiene la serie de tiempo
de valores enteros a analizar y se va a analizar dependiendo
del el tamano de la ventana
input: manejador x, manejador y, tamaño ventna tvent

regresa: la ventana de la serie X
"""
X = []
```

```
lx = x.read(tvent)

for i in xrange(len(lx)):
X.append(int(lx[1]))

return X

def discreto(var, max, min, n_edos):
"""
discretiza la variable v dentro
del rango establecido por max y  min
con la cantidad de estados definidos en n_edos

regresa dig: un entero que var discretizada
"""
intervalo = max - min
R = float(intervalo) / float(n_edos)

dig = int((var-min) / R) - 1
if dig < 0:
return 0

return dig

def discretiza(V, vmax, vmin, n):
"""
discretiza el conjunto de datos que se le paso en el
parametro, tambien se tiene que especificar el valor
minimo, maximo y el numero de estados

regresa D: lista con los valores discretos de P
"""
D = []
for i in xrange(len(V)):
d = discreto(V[i], vmax, vmin, n)
D.append(d)
return D

def prob(X):
"""
determina la probabiliad de los elementos en
el conjunto X

regresa [P, N]: un vector con 3 objetos:
P. probabilidad de ocurrencia de cada simbolo en X
N. la cantidad original de elementos en X
```

```
"""
D = {} #diccionario donde guardamos la ocurrencia de cada
               #simbolo
N = len(X)

        #contamos las ocurrencias
for i in xrange(N):
D[X[i]] = D.get(X[i],0) + 1

#determinamos probabilidades
for i in range(len(D)):
D[D.keys()[i]] = float(D.values()[i]) / float(N)

return [D, N]



def prconj(Yd, Xd):
"""
determina la probabilidad conjunta P(Y=yi|X=xi)
entre los conjuntos X e Y que deben estar
discretizados

regresa [PX, PY, D, PXY]:
PX: probabilidad de aparicion de X
PY: probabilidad de aparicion de Y
D: lista de estados
PXY: probabilidad conjunta de los sistemas X e Y
"""
if len(Yd) != len(Xd):
raise Exception, 'Datos de diferente tamaño'

[PX,NX] = prob(Xd)
[PY,NY] = prob(Yd)

D = dict()
N = len(Xd)

for i in range(N):
D[Xd[i],Yd[i]] = D.get((Xd[i],Yd[i]),0) + 1

for i in range(len(D)):
D[D.keys()[i]] = float(D.values()[i])/N

return [PX, PY, D.keys(), D]
```

```
def H(X):
"""
regresa la entropia de p en los estados que deben venir
en un diccionario como esta definido en prob, las unidades de
H esta en bits

regresa H: float con la entropia de X
"""
H = 0.0

for j in range(len(X)):
H += X[j] * log(X[j])

H /= -log(2)

return H


def Hxy(P):
"""
calcula la entropia conjunta
recibe  P: un diccionario con la probabilidad conjunta

regresa la entropía conjunta entre dos sistemas
"""
S = 0.0

for i in range(len(P)):
S += P.values()[i] * log(P.values()[i])

S /= -log(2)
return S

def inf_mutua(H1, H2, HXY):
"""
calcula la entropia conjunta de H1 y H2
recibe tambien HXY que es la entropia conjunta de H1 y H2

regresa la informacion mutua dependiendo de la entropia
del primer sistema, la del segundo y la conjunta
"""
I = H1 + H2 - HXY
return I

def IM(X,Y,nx,ny):
"""obtiene la inf mutua de las series X e Y, hay que pasarle
```

```
los parametros de discretizacion para cada serie"""
Xd = discretiza(X,maxl(X),minl(X),nx)
Yd = discretiza(Y,maxl(Y),minl(Y),ny)
[Px,Py,d,Pxy] = prconj(Yd,Xd)
Hx = im.H(Px)
Hy = im.H(Py)
Hxy = im.Hxy(Pxy)
return inf_mutua(Hx,Hy,Hxy)
```

Bibliografía

[1] Boccara, N. *Modeling Complex Systems*. Springer, 2004.

[2] Burgin, M., and Dodig-Crnkovi, G. A systematic approach to artificial agents. *CoRR abs/0902.3513* (2009).

[3] Coello, C. e. a. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2007, ch. MOEA Theory and Issues.

[4] Collins, N. A. The dynamics of voter turnout and party choice. *JOP* (2010).

[5] D., W., and Macready, W. No free lunch theorems for search. Working Papers 95-02-010, Santa Fe Institute, Feb 1995.

[6] D., W., and Macready, W. What makes an optimization problem hard? *Complex. 1*, 5 (1996), 40–46.

[7] Heath, B., Hill, R., and Ciarallo, F. A survey of agent-based modeling practices (january 1998 to july 2008). *Journal of Artificial Societies and Social Simulation 12*, 4 (2009), 9.

[8] Holland, J. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.

[9] Jennings, N. R., and M., W. Applying agent technology. *Journal of Applied Artificial Intelligence* (1995). special issue on Intelligent Agents and Multi-Agent Systems.

[10] Mitchell, M. *Complexity. A Guided Tour*. Oxford University Press, 2009.

[11] "Neme, A., Hernández, S., Neme, O., and Hernández, L. Self-organizing maps with non-cooperative strategies (som-nc). In *WSOM '09: Proceedings of the 7th International Workshop on Advances in Self-Organizing Maps* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 200–208.

[12] Neme, A., and Hernánez, S. *Nature-Inspired Algorithms for Optimisation*. Studies in Computational Intelligence. Springer, 2009, ch. Algorithms Inspired in Social Phenomena.

[13] Neme, A., Neme, O., and Hernández, S. An electoral prefernces model based on self-organizing maps. *Journal of Computational Science* (2011).

[14] Nicolis, G., and Rouvas-Nicolis, C. Complex systems. *Scholarpedia 2*, 11 (2007), 1473.

[15] Nigel, G. *Agent-Based Models*. Quantitative Applications in the Social Sciences. SAGE Publications, 2002.

[16] Resnick, M. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Complex Adaptive Systemas. MIT Press, 1997.

[17] Reynolds, R. G. An introduction to cultural algorithms. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming* (1994), World Scienfific Publishing, p. 131–139.

[18] Schelling, T. C. Dynamic models of segregation. *Journal of Mathematical Sociology* (1971).

[19] Weise T., Chiong R., e. a. *Nature-Inspired Algorithms for Optimisation*. Studies in Computational Intelligence. Springer, 2009, ch. Why is Optimization Difficult?

[20] Yin, H. *Computational Intelligence: A Compendium*. Studies in Computational Intelligence. Springer Berlin / Heidelberg, 2008, ch. The Self-Organizing Maps: Background, Theories, Extensions and Applications.