UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Simulación de un Sistema Urbano usando Autómatas Celulares

Fidel Serrano Candela

Abril 2006

QUE PARA OBTENER EL TÍTULO DE:

FÍSICO

"... modelar no con el fin de hacer predicciones precisas sino como una manera sistemática de aprender de la realidad, desarrollando al menos un modelo que pueda generar por si mismo la trayectoria del sistema en el pasado" Peter Allen [1]

Agradecimientos

La doctora Carmen Reyes, Yosu Rodríguez, y Alejandro Mohar fueron esenciales impulsores de este proyecto, Mauricio Santillana quien fungió de asesor en este trabajo, Jesús Trujillo que fue de gran ayuda para aprender ARC info. Y a mi mujer por supuesto, que me acompaño estoicamente en el proceso.

Contenido

Resumen

En este trabajo se describe una metodología computacional que permite estudiar el crecimiento morfológico de los asentamientos humanos, ademas de la creación de escenarios hipotéticos tomando en cuenta variaciones en el entorno citadino, como la construcción de vialidades, o cambios en las masas boscosas cercanas.

La base de esta metodología es una herramienta matemática, relativamente simple pero bastante poderosa, los autómatas celulares. Estos se han utilizado con éxito en la simulación de sistemas biológicos y urbanos. Los autómatas celulares son sistemas dinámicos discretos en los que un comportamiento local simple, puede generar una dinámica global muy compleja.

La calibración de este tipo de modelos es un problema esencial para su aplicación, por lo cual probaremos un algoritmo evolutivo como método de calibración.

En el capítulo 1 se establecen ciertas ideas preliminares, en el capítulo 2 se describen las herramientas de modelación que fueron utilizadas, mientras que en el capítulo 3 se reporta la implementación de un modelo para simular el crecimiento de un pueblo cercano a la ciudad de México, en el capítulo 4 se reportan los resultados para este caso de estudio y finalmente en el capítulo 5 se delinean algunas conclusiones.

Capítulo 1

Introducción

El estudio de la dinámica de crecimiento de las ciudades es cada día más necesario. Pues la población del planeta ha tendido ha concentrarse en núcleos urbanos que crecen cada vez más en tamaño y extensión. En México la mayoría de las ciudades actuales han crecido desordenadamente, sin regulaciones efectivas, que anticipen y encausen su crecimiento.

Las autoridades no han podido prevenir las formas inconvenientes de crecimiento urbano, en parte por que no tienen forma de preverlas, nosotros creemos que un modelo computacional que simule el crecimiento de la mancha urbana, sería de utilidad.

Algunos estudiosos afirman que "el estado mexicano ha claudicado en la misión de dirigir o encausar el desarrollo urbano del país, pensando que el mercado puede cumplir la función de asignación del suelo y construcción de la infraestructura necesaria" [2]. Esta situación preocupa, pues el crecimiento desmedido, y desorganizado de las ciudades, y en particular de la ciudad de México, pone en riesgo el porvenir de todo el entorno natural y la viabilidad de un futuro decoroso.

Predecir el comportamiento humano esta fuera del alcance de este trabajo, sin embargo dentro de la complejidad del sistema urbano hay tendencias, dinámicas locales, y estructuras globales que podemos estudiar y entender hasta cierto punto. Esto nos permite por una parte aprender del pasado y por otra "proyectar" estas experiencias al futuro.

La motivación para modelar y simular el crecimiento urbano es proveer una herramienta que permita conocer como crecería la ciudad bajo determinadas hipótesis como, la construcción de infraestructura, o el establecimiento de políticas de planeación.

El surgimiento de la nueva generación de modelos de simulación urbana esta ligado a la noción de que los sistemas urbanos son sistemas complejos [3], pues constan de un gran número de actores y variables del entorno que interactúan entre si a nivel local. Langton, por ejemplo, caracteriza la

transición de fase del sistema que consta de gran cantidad de componentes microscópicos como dos procesos por separado, el periódico y el caótico. Entre los dos hay un régimen de transición que muestra un comportamiento complejo, poniendo a estos sistemas complejos en "la orilla del caos" [4].

Si bien es claro que una predicción precisa del futuro no es posible, creemos que mediante la modelación y simulación del sistema urbano es posible mejorar el entendimiento de la dinámica que gobierna el crecimiento de la mancha urbana.

Los investigadores en este campo han tenido que simplificar los sistemas urbanos para poder usar la estadística o la deducción. De hecho por la dificultad de aplicar la deducción a estos sistemas, "la simulación es a menudo la única manera práctica de estudiar la dinámica de estos sistemas complejos." [5]. Tradicionalmente hay dos formas de aproximarse a las propiedades de un sistema complejo. Ambas reducen el sistema de complejidad a un sistema de simplicidad, la primera simplifica de tal manera que el cerebro humano sea capaz de aplicar un pensamiento deductivo, y la segunda simplifica para usar métodos estadísticos y filtrar el ruido con la intención de extraer la parte "regular" del comportamiento, de cualquier modo hay problemas al usar cualquiera de estos acercamientos. Es difícil encapsular un sistema complejo sin perder los detalles cruciales del sistema, mientras que es aceptado comúnmente que un modelo es una simplificación de la realidad, reducir la complejidad puede llevar a un modelo a reproducir comportamientos cualitativamente distantes de la realidad.

Un modelo computacional puede mantener detalles útiles de tal forma que reproduzca el comportamiento fundamental del sistema real. La simulación por lo tanto puede revelar la dinámica oculta que resultaría difícil de notar sin correr un modelo. Conte y sus colegas [6] argumentan que la simulación puede proveer una alternativa para estudiar los fenómenos sociales que son tan complicados que la observación no puede discernir claramente la interrelación.

Un modelo de simulación es en esencia una analogía al mundo real. Que tanto simplifica el comportamiento del sistema real es un asunto crucial, aquí ciertamente existe el dilema entre reconocer la complejidad de las reglas y entender el comportamiento del sistema. "Si la complejidad de las reglas es reducida tanto que no pueda captar las propiedades del sistema real, la simulación produce las propiedades de reglas simplificadas, en la mayoría de los casos manifestadas en un conjunto de ecuaciones matemáticas, en vez de las propiedades de la realidad y si por el contrario la complejidad de las reglas se mantiene, el espacio de reglas es tan complicado que es imposible entender las propiedades del modelo, pues, los valores de los parámetros pueden ser combinados en un número astronómico de diferentes formas."[7]Por lo que un método de calibración que permita lidiar con este gran número de com-

binaciones de parámetros, seria un gran avance pues permitiría incluir más factores en el modelo.

"La segunda ley de la termodinámica dice que sistemas físicos reversibles tienden con el tiempo a estados de mayor entropía y mayor *desorden*, pero sistemas *disipativos* que involucran irreversibilidad microscópica, o sistemas abiertos a interacción con su entorno, pueden evolucionar de *desordenados* a un mayor *orden*. Ejemplos de esto son los copos de nieve, patrones de flujo en fluidos turbulentos, y sistemas biológicos." [8]. Las ciudades también generan patrones regulares de crecimiento, incrustados en una dinámica caótica, esto es aparentemente producto de la interrelación de los actores locales y sus actividades y preferencias, en el siguiente capitulo se describe un conjunto de herramientas que permiten un acercamiento a este tipo de sistemas.

Capítulo 2

Herramientas de Modelación

Un modelo es en esencia una abstracción del mundo real, casi necesariamente una simplificación, una cuestión por resolver es ¿cuales simplificaciones del mundo nos ayudan a entenderlo?. En este capitulo describiremos el conjunto de herramientas matemáticas que se utilizamos para proponer un modelo de expansión urbana, en las inmediaciones de la ciudad de México.

2.1 Complejidad

"Desde la antigüedad la ciencia se ha enfocado en el estudio de las regularidades y esto ha significado que la complejidad, vista como la ausencia de regularidades, se haya tendido a evitar o ignorar." [9]. Sin embargo ha habido ocasionalmente discusiones sobre la complejidad, por ejemplo cerca del 200 AC los Epicúreos discutían la idea de que complejas y variadas formas de la naturaleza podrían estar constituidas por arreglos de un pequeño numero de tipos de átomos elementales, de la misma forma que complejos y variados textos que se han escrito están hechos con un numero pequeño de letras. Con su fuerte énfasis en leyes simples y medidas numéricas, la física a tendido normalmente a definirse a si misma evitando la complejidad. Pero al menos desde los 1940s asuntos de complejidad han sido ocasionalmente mencionados por físicos como de importancia, la mayoría de ellos en conexión con turbulencia de fluidos o detalles de ecuaciones diferenciales no lineales. Preguntas sobre la formación de patrones, particularmente en biología y en relación con la termodinámica, llevaron a una secuencia de estudios de ecuaciones de reacción-difusión, que ya para los 1970s se presentaban como relevantes temas de complejidad, bajo nombres como auto-organización, sinergetica y estructuras disipativas. Para finales de 1970s el trabajo de Benoit Mandelbrot en fractales, fue un ejemplo importante de enfoque general para tratar un tipo de complejidad. Y la teoría del caos con base en la teoría de sistemas

dinámicos también empezó a ser popular a fines de los 1970s, particularmente en conexión con la turbulencia de fluidos, sin embargo, en prácticamente todos estos casos, el énfasis siguió siendo tratar de encontrar algún aspecto de los comportamientos de complejidad que pudiera ser resumido en un numero o en una ecuación matemática. A principios de los 1980s había ya varios tipos de sistemas abstractos cuyas reglas eran simples, y sin embargo mostraban un comportamiento complejo, particularmente en simulaciones por computadora. Pero usualmente eran vistas como una curiosidad, y no había la creencia de que hubiera fenómenos generales que fueran de interés central, en la ciencia. Y de hecho permaneció como una creencia casi universal que para capturar cualquier complejidad de verdadero valor científico, era necesario un modelo considerablemente complejo.

En la siguientes secciónes delinearemos herramientas que permiten estudiar la complejidad con un enfoque diferente y que ha sido aplicada en las últimas décadas, en diversos campos de la ciencia.

2.2 Sistemas Dinámicos Discretos

La teoría de los sistemas dinámicos es una rama clásica de las matemáticas, que vio sus inicios con Newton alrededor de 1665. Y provee modelos matemáticos para sistemas que evolucionan en el tiempo.

En 1880, Poincare uso sistemas dinámicos continuos para estudiar la estabilidad del sistema solar, y encontró conveniente remplazar el fluir continuo del tiempo por un análogo discreto en el cual el tiempo se incrementa en saltos regulares. Estos sistemas son ahora llamados sistemas dinámicos discretos. Así que, por más de un siglo, se han estudiado paralelamente los sistemas dinámicos continuos y discretos. Los sistemas dinámicos discretos usualmente se representan como la iteración de un mapa (también llamado endomorfismo) de un dominio en si mismo. Una forma de escribir esto es

$$x_{n+1} = f(x_n), \tag{2.1}$$

donde x representa el estado del sistema y f una función que determina el siguiente estado del sistema.

Puede ser invertible (siendo uno a uno y sobre), o no invertible (si falla alguna de éstas o las dos). Poincare introdujo los mapeos invertibles que han sido estudiados desde entonces. Los estudios sobre los no invertibles habían sido dispersos hasta hace relativamente pocos años, y se volvieron una de las ramas mas activas en la investigación de frontera, debido a su extraordinaria utilidad en diversas aplicaciones.

La "teoría del caos" es un pseudónimo muy popular para referirse al estudio de los sistemas dinámicos. Esta manera de llamarlos se hizo popular hace

un par de décadas, cuando su aplicación a sistemas naturales aparentemente caóticos se dio a conocer. El estudio de los sistemas dinámicos discretos para mapas no invertibles, comenzó algunos años después de Poincare, y su desarrollo se ha acelerado particularmente por la revolución computacional de las últimas décadas, siendo hoy día un joven campo de estudio.

Los sistemas dinámicos discretos, como su nombre lo indica son sistemas que evolucionan en el tiempo tomando valores discretos

2.3 Autómatas Celulares

"Los autómatas celulares son idealizaciones matemáticas de sistemas físicos en las cuales el espacio y el tiempo son discretos, y las cantidades físicas toman un numero finito de valores discretos." [8], fueron inventados por los matemáticos von Neumman y Ulam en los 1940s mientras trabajaban en Los álamos Nacional Laboratory y después los vieron como una posible idealización de sistemas biológicos [10], el autómata celular mas famoso es tal vez el juego de la vida, inventado en los 1960s por el matemático John Conway, y es un autómata celular (AC) bidimensional donde el equilibrio global de la población depende de la dinámica local. Esta clase de artefacto se vio por mucho tiempo como una curiosidad pero poco a poco se ha tomado con más seriedad en el ámbito científico.

Los autómatas celulares son sistemas dinámicos discretos simples de construcción pero que pueden desarrollar un comportamiento muy complejo [11]. Están constituidos por una retícula regular de celdas de una o mas dimensiones, donde cada celda esta en un estado que depende directamente del estado previo de las celdas cercanas. Y en este sentido tienen su parecido con las ecuaciones diferenciales parciales discretizadas, por ejemplo la ecuación de calor

$$\frac{\partial u}{\partial t} = c\frac{\partial^2 u}{\partial x^2}, \tag{2.2}$$

definida en el intervalo $0 \leq x \leq L$, para $t \geq 0$.

Si discretizamos esta ecuación, para una dimensión obtendremos.

$$u_i^{n+1} = u_i^n + c\frac{\Delta t}{(\Delta x)^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n), \tag{2.3}$$

Donde u_i^n representa a u evaluada en la posición i-esima para el tiempo n. Como se observa en la ecuación el valor de u_i^{n+1} es función de vecindad espacial, o sea $u_i^{n+1} = F(u_{i+1}^n, u_i^n, u_{i-1}^n)$. En el contexto de los autómatas celulares, el paralelo de esta F son las reglas de transición. Que definen el estado de cada celda para el tiempo n+1 en función de del estado de las celdas vecinas en el tiempo n.

Stephen Wolfram [12] hizo un estudio exhaustivo de las reglas de transición para autómatas celulares en una dimensión y clasifico estas reglas en cuatro clases.

I) El AC acaba en un estado global homogéneo.

II) El AC oscila en un ciclo periódico.

III) El AC muestra desorden.

IV) El AC muestra un comportamiento complejo.

Este comportamiento complejo es el que nos interesa particularmente pues una gran cantidad de sistemas abiertos muestran un comportamiento similar. En la figura 2.2 se muestra la evolución de un autómata celular de una dimensión que tiene raglas de transición de la clase IV, el punto de hasta arriba representa la condición inicial y el tiempo corre hacia abajo. Para modelar fenómenos espaciales se utiliza normalmente una retícula de dos dimensiones, pero para visualizar la evolución de un autómata de dos dimensiones, hacen falta tres dimensiones esto se puede conseguir con una serie de mapas, o bien con una animación, es por eso que como ejemplo mostramos un autómata unidimensional.

Sin embargo los autómatas que son de interés en esta investigación son de al menos dos dimensiones, esto hace que la vecindad de una celda varíe en tamaño y forma, en la figura 2.1 se muestran algunos ejemplos de vecindad. El tipo de vecindad que se utiliza tiene directamente que ver con la abstracción del sistema en un modelo, y varía según la necesidad.

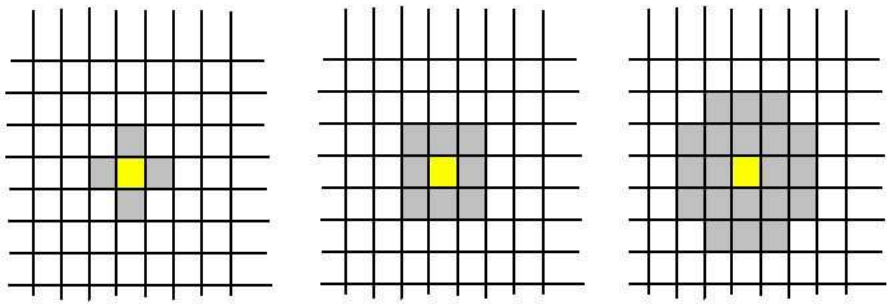

Figura 2.1: Tres diferentes vecindades para una retícula cuadrada.

En el formalismo propuesto por Wolfram en 1984 las reglas de transición son universales, es decir, se aplican uniformemente en todo el dominio. Visto de otra forma, todas las celdas siguen exactamente la mismas reglas de transición de estado, sin importar su ubicación en el espacio.

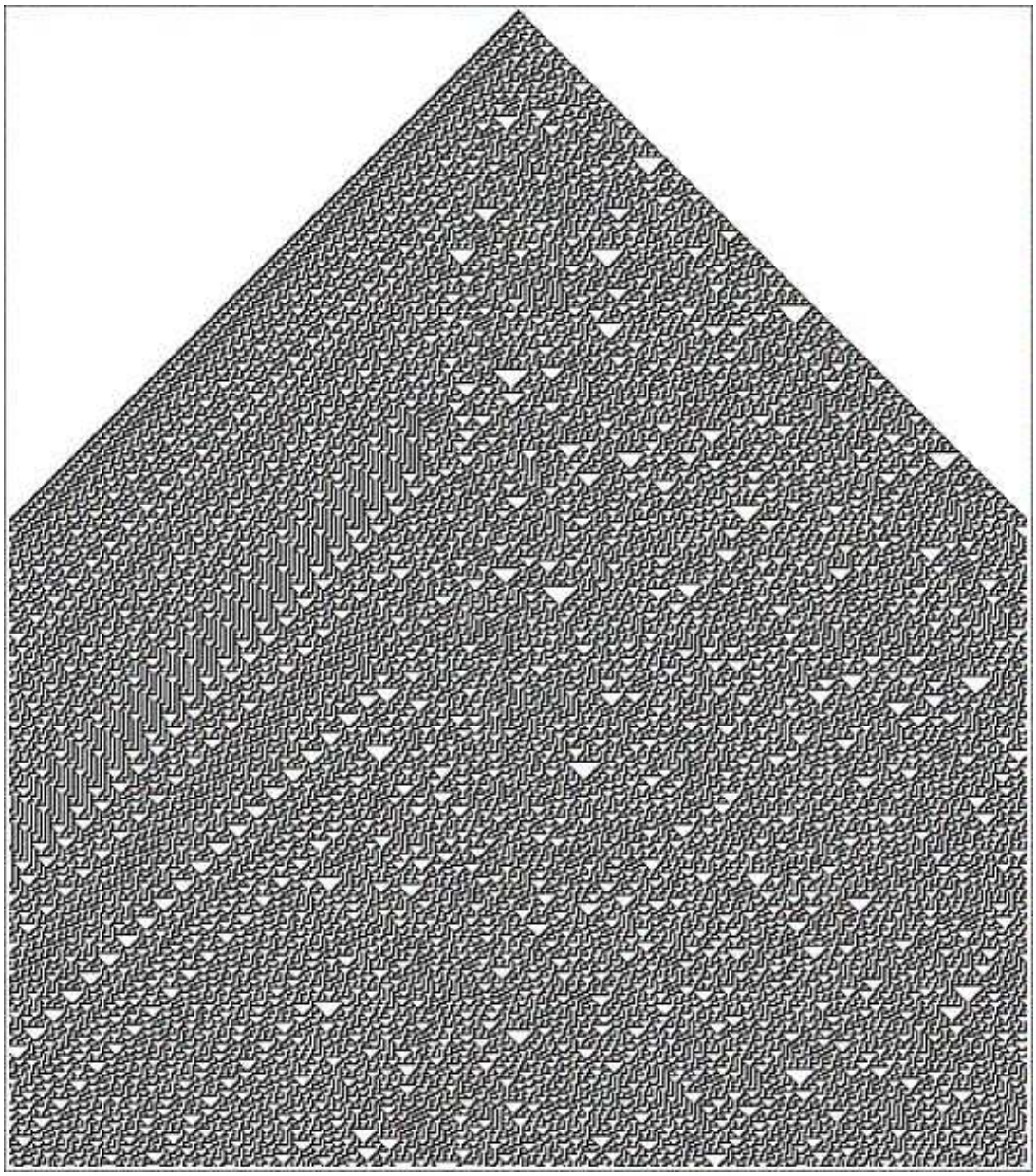

Figura 2.2: Visualización de los primeros 500 pasos en la evolución del autómata celular unidimensional con reglas de transición de la clase IV

Variaciones a este formalismo se han implementado para modelar una gran variedad de sistemas físicos y biológicos, como por ejemplo, la dinámica poblacional de un ecosistema, los solitones, la turbulencia, y varios más.

2.4 Modelacion de Sistemas Urbanos con Autómatas Celulares

Los autómatas celulares se han utilizado para modelar una variedad de sistemas urbanos, ejemplos notables de esto son: la dinamica de usos de suelo por los grupos de trabajo de Fulong Wu [13] y Roger White [14], la migración interna en las ciudades por Diane Vanbergue [15] y la difusion de enfermedades contagiosas por Mansilla [16].

En este trabajo nos dimos a la tarea de modelar el cambio de usos de suelo en una zona dada, esto significa que el estado de cada celda esta definido por el uso que los humanos le den al territorio que representa. En la figura 2.3 se muestra la evolución de un autómata celular de dos dimensiones con tres posibles valores, a partir de una única semilla con reglas de transición que toman en cuenta una vecindad de cruz. La idea es que el estado de una celda $C_{i,j}$ en la n-esima iteracion, depende del estado de las celdas que pertenecen a la vecindad de $C_{i,j}$ en la iteracion anterior.

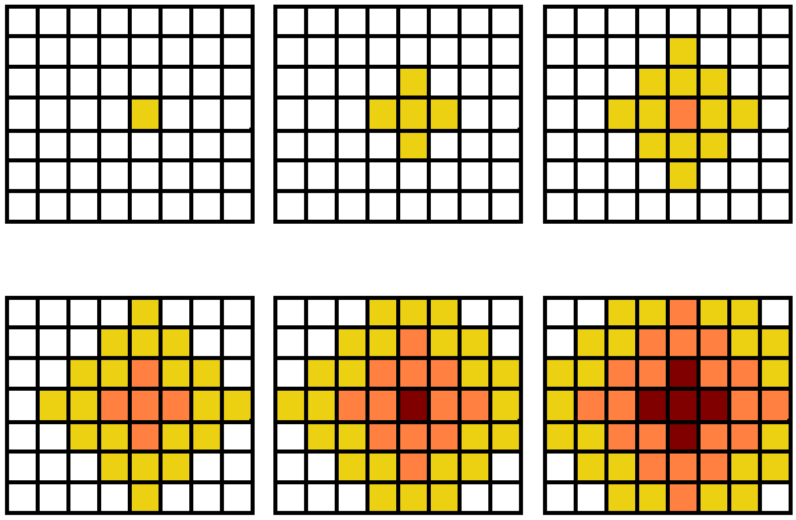

Figura 2.3: Evolucion de un autómata celular bidimensional a partir de una única semilla

El crecimiento y densificación de la mancha urbana, es lo que nos interesa modelar, por lo que es necesario definir el significado de las celdas y de las reglas de transición de estado.

En modelos urbanos una celda puede representar una o varias cuadras, una hectárea, o hasta un Kilómetro, dependiendo de las dimensiones de la ciudad y el propósito del modelo. En aplicaciones geográficas las celdas normalmente se construyen a partir de una teselacion regular cuadrada, aunque

se han intentado teselaciones hexagonales también [17]. “El estado de la celda podría verse como una clasificación mas o menos compleja de usos de suelo, valor de la tierra, cobertura del terreno, densidad poblacional y otras” [18], es decir que los estados posibles de cada celda son una especie de simplificación de lo que se observa en el mundo real.

El significado de las reglas de transición es un poco mas intrincado, ya que en la realidad el territorio que representa una celda no muta, así como así, mas bien “actores humanos constructores, empresas, instituciones financieras, autoridades, terratenientes, propietarios, compradores de casas maniobran, colaboran, y compiten para cambiar la ciudad dados sus diferentes propósitos... Las reglas de trasición deben ser entendidas como algo que engloba todas estas actividades”[18].

También es posible incluir restricciones geográficas, a costa de relajar el carácter universal de las reglas de transición, es decir que las reglas de transición de cada celda dependan de su posición. Esto puede representar la heterogeneidad del terreno o algún otro atributo no homogéneo en dominio espacial. Los investigadores en este campo han probado diferentes variaciones al formalismo original de los autómatas celulares, para incluir factores como la pendiente del terreno, la red de calles y carreteras, el tiempo de traslado al trabajo, el precio del terreno y otros. La inclusión de estos factores es lo que lleva a la definición de los parámetros que permitirán el ajuste del modelo al sistema real.

La decisión de cuáles y cuántos factores geográficos son tomados en cuenta, es de trascendencia pues demasiados factores llevarían al modelo a ser inoperante por el tiempo de cómputo requerido para su calibración mientras que pocos, podrían ocasionar que el modelo no captase características fundamentales del sistema. Una vez definidos los factores geográficos no homogéneos en el dominio espacial, el conjunto óptimo de parámetros será el que genere el comportamiento mas aproximado al observado, al proceso que lleva a encontrar este conjunto de parámetros se le denomina calibración.

Hasta el momento no contamos con una metodología estandarizada para realizar la calibración. Sin embargo existen ejemplos relativamente exitosos de modelos basados en autómatas celulares con calibraciones locales para algunas ciudades europeas y norteamericanas.

2.5 Índice de Similitud Morfológica

Ya que tenemos un modelo es necesario ajustarlo lo mas posible al sistema real que estamos modelando, eso significa poder medir la “cercania” entre un resultado dado y el resultado esperado, esta es la razon por la que desarrollamos un índice de similitud morfológica.

Equipos de investigadores en procesamiento de imágenes han desarrollado, diferentes maneras de cuantificar la similitud entre dos imágenes, por lo que se utilizan distintas medidas, dependiendo de la aplicación en cuestión. En esta sección describiremos el índice de similitud morfológica que utilizaremos para la calibración del modelo.

Consideramos apropiado para los propósitos de esta investigación un índice de similitud i entre dos imagines binarias, (urbano, no urbano). Para encontrar i entre el escenario "real" R, y la salida del modelo O, dividimos el dominio espacial Ω del en sub-dominios Ω_j de tal forma que

$$\bigcup_j \Omega_j = \Omega \tag{2.4}$$

y calculamos el índice i de similitud de la siguiente forma

$$i = i(R, O) = \frac{1}{N(\Omega_j)} \sum_{celdas \in \Omega_j} (c(R) - c(O))^2 \tag{2.5}$$

Donde $N(\Omega_j)$ es el numero total de subdominios Ω_j, $c(R)$ y $c(O)$ valen 1 si la celda esta urbanizada y 0 si no. Como ejemplo sea $N(\Omega_j) = 1$, entonces $\Omega = \Omega_1$. Y el índice i en este caso, representa la diferencia en área de las dos imágenes, pero para esta subdivisión, i no da ninguna información en cuanto a la forma, pero variando la manera de subdividir podemos obtener una medida que si reconozca forma. En este sentido escogimos $N(\Omega_j)$ para tomar en cuenta lo que llamaremos la "escala relevante" de nuestro modelo.

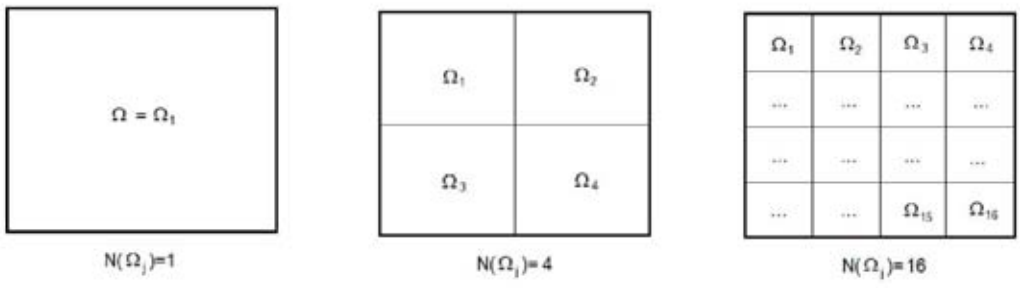

Figura 2.4: Ejemplos de partición del dominio especial.

Un índice E mas general podría ser utilizado para captar las diferencias de forma a diferentes escalas, por ejemplo definiendo E como una combinación lineal pesada de los índices i_α :

$$E = E(R, O) = \sum_\alpha w_\alpha i_\alpha(R, O) \tag{2.6}$$

donde i_α representa el índice i previamente definido, dada una partición α y para los pesos w_α de cada i_α.

Los pesos w_α deben satisfacer $\sum_\alpha w_\alpha = 1$. Y de esta forma E proporciona una manera natural de tomar en cuenta diferentes escalas en el índice de similitud morfológica.

De hecho si consideramos las imágenes i funciones escalares cuyo dominio es un subconjunto de $\Omega \in R^2$, o sea $I : \Omega \to R$ y una partición ? del dominio tal que $N(\Omega_j)$ sea igual al numero de celdas consideradas en el modelo, entonces tendremos que $\alpha = \alpha_{max}$ y i_α es directamente la norma L^2 normalizada por el tamaño del dominio $\mu(\Omega)$, de la diferencia entre R y O.

$$i_{\alpha_{max}} = \frac{1}{\mu(\Omega)} ||R - O||^2_{L^2} \tag{2.7}$$

Desde este punto de vista tiene sentido pensar en E como una función de distancia entre O y R. Para nuestros propósitos particulares consideramos, $E = w_1 i_1 + w_2 i_2 + w_3 i_3$ con $\alpha = 1, 2, 3$ escogidos de tal forma que $N(\Omega_j) = 1, 4, 16$ respectivamente, los valores para w_α fueron determinados empíricamente a partir de cuantiosos de experimentos numéricos para obtener el mejor comportamiento. Aquí cabe mencionar que definimos esta E con el unico fin de calibrar el modelo en términos visuales.

2.6 Calibración del Autómata

Ya que contamos con un indice de similitud que nos permite de alguna forma evaluar que tan bueno es un conjunto de parametros dados, quisieramos encontrar el mejor conjunto de parametros, o almenos un conjunto aceptablemente bueno de ellos. Esto es un tema por el hecho de que probar todas las combinaciones de parámetros llevaría demasiado tiempo de cómputo.

Sean R_{t_1} y R_{t_2} las matrices que representan al sistema real en el tiempo t_1 y t_2 respectivamente, y O_{t_2} una salida del modelo para el tiempo t_2, entonces calibrar el modelo, se refiere a encontrar los valores de los parámetros para los cuales O_{t_2} sea lo mas cercana a R_{t_2}. Podemos ver esto como un proceso de optimización en el que queremos encontrar la imagen $O^k_{t_2}$, generada en el k-esimo paso de la "técnica de optimización", para la cual

$$i(R_{t_2}, O^k_{t_2}) \leq \delta \tag{2.8}$$

Donde δ es un número pequeño que representa el "error" que es "aceptable" como salida del modelo.

Ahora bien, si logramos encontrar un conjunto de parámetros para los cuales se que cumpla la ecuación 2.8 podremos decir que el modelo, calibrado con ese conjunto de parámetros reproduce la dinámica del sistema, con un

error menor que δ, en el intervalo de tiempo (t_1, t_2). Y con ese mismo conjunto de parámetros podemos hacer una proyección a futuro, por ejemplo O_{t_2+7} para obtener un escenario hipotético 7 años después de t_2.

Creer firmemente en este escenario hipotético seria suponer que la manera en la que el sistema evolucionó, será la manera en la que el sistema evolucionará, esto aclara porque no pretendemos hacer una predicción precisa, sino solo una proyección tentativa.

De cualquier modo lo que nos atañe ahora es el metodo para encontrar ese conjunto de parámetros que cumpla con la ecuación 2.8. Existen varios métodos de optimización, entre los más poderosos están los que calculan el gradiente discretizado de una función "desconocida" a optimizar, esto se hace evaluando la función en un punto p_j y su vecindad en el espacio n-dimensional de los parámetros, para después probar la función en un punto $p_{j+1} = p_j + C(\nabla p_j)$, donde C es una constante y ∇p_j es el gradiente discretizado de la función. De esta forma se van evaluando solamente los puntos mas prometedores.

Por otro lado están los algoritmos evolutivos que tampoco prueban todos los puntos en espacio n-dimensional de los parámetros, y no requieren del calculo del gradiente discretizado, pues consideran una población de conjuntos de parámetros, que son sometidos a una especie de selección natural para obtener un conjunto "evolucionado" de parámetros. Este método tiene la ventaja de que funciona para calibrar modelos no deterministas como el que desarrollamos en esta investigación. En la siguiente sección describiremos con mas detalle este tipo de algoritmos.

2.7 Algoritmos Evolutivos

Los algoritmos evolutivos son artilugios computacionales que permiten lidiar con modelos no deterministas, su nombre fue inspirado en la teoría de la evolución de las especies en donde la selección natural y las mutaciones juegan el papel central. Los algoritmos evolutivos han sido ampliamente usados en varios campos de la ciencia, en las ultimas dos décadas, la clase de problemas para los que son útiles fueron caracterizados por Banzhaf[19]:

1 Problemas en los que la interrelación entre las variables relevantes es pobremente entendido.

2 Problemas donde encontrar el tamaño y forma de la solución es la parte más difícil del problema.

3 Problemas donde el análisis matemático convencional no provee, o no puede proveer una solución analítica.

4 Problemas donde una solución aproximada es aceptable (o es el único resultado que puede ser obtenido)

5 Problemas donde pequeños incrementos en el desempeño son rutinariamente medidos y altamente apreciados

6 Problemas donde hay una gran cantidad de datos, en formato legible por computadora, que requieren examinación, clasificación, e integración (como biología molecular para proteínas y secuencias de ADN, datos astronómicos, datos provenientes de imágenes de satélite, etc.)

Nuestro problema cae en varios de los anteriores, razón por la cual decidimos explorar el potencial de los algoritmos evolutivos como herramienta para calibrar el modelo de crecimiento urbano.

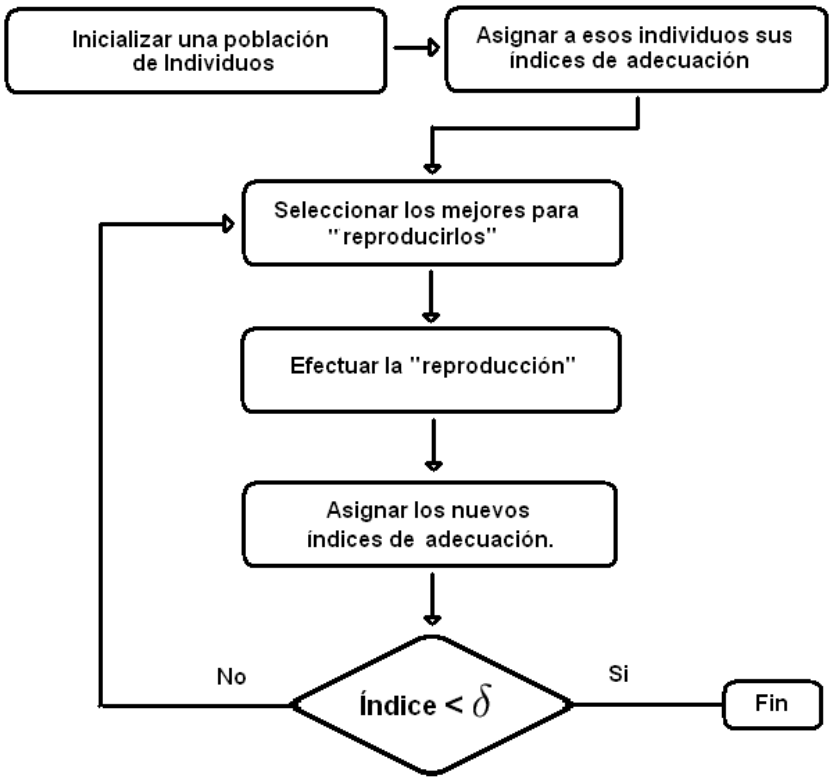

Figura 2.5: Diagrama de flujo de un algoritmo evolutivo.

Pero ¿como funciona un algoritmo evolutivo para "optimizar" el desempeño de un modelo? Un algoritmo evolutivo simple se puede presentar como series de seis pasos [20] que se pueden visualizar en el diagrama de flujo

que se muestra en la figura 2.5, donde δ es el límite aceptable para el índice de adecuación, mientras que "reproducir" se refiere básicamente a tomar dos conjuntos de parámetros y hacer una combinación de ellos para producir un nuevo conjunto de parámetros.

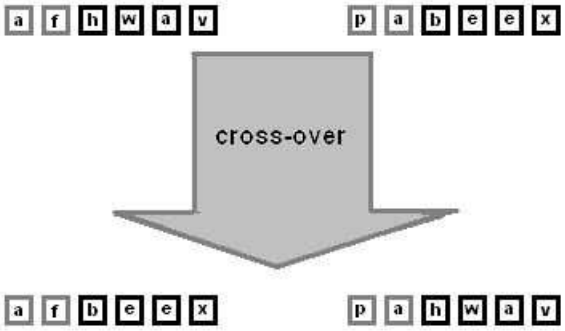

Figura 2.6: Crossover de dos padres con 6 genes

Una forma de aparear dos conjuntos de parámetros es mediante el cross-over, que no es otra cosa que dividir en subsecciones los parámetros y cruzarlos como se muestra en la figura 2.6. De esta manera de dos padres tenemos dos hijos que tienen partes complementarias del papá y de la mamá, dependiendo del tipo de cross-over puede hacerse en dos o más segmentos. Esta forma de "reproducir" tiene la ventaja de que a la larga aísla los segmentos de parámetros más ganadores.

Figura 2.7: Mutación de dos de los 6 genes

Además, para evitar que la población se quede sin el equivalente de la variabilidad genética, se implementa el operador de mutar, que consiste en, modificar 1 o mas genes de un "ADN", o conjunto de parámetros, por ejemplo en la figura 2.7 cambian o "mutan" el primer y el ultimo gen.

Entonces una generación esta constituida por los "hijos" de los mejores individuos de la generación anterior, talvez con algunas mutaciones. De esta forma es más claro cómo este algoritmo esta inspirado en la teoría de la evolución de las especies, debe sin embargo, quedar establecido que no estamos, ni asegurando que esta teoría sea correcta, ni sugiriendo que este algoritmo pueda simularla. Simplemente que nos es de utilidad.

Capítulo 3

Implementación

En este capitulo describiremos la implementación de un modelo basado en autómatas celulares que simula el cambio de usos de suelo, para una zona de estudio especifica.

Esto incluye los estados posibles de cada celda, las reglas de transición, la adquisición de datos y el método de calibración.

3.1 Modelo de Crecimiento Urbano

¿Cuanto tiempo tardara una zona dada en estar totalmente ocupada por humanos?, o ¿cual será la dirección en la que mas crezca un poblado?. Estas son el tipo de preguntas que nos gustaría ayudar a responder, con esta idea en mente se implemento un modelo que simula el cambio de usos de suelo. Esto significa el que estado de cada celda debe reflejar de alguna forma el grado de ocupación humana de esa parte del territorio. Con esto en mente definimos los valores que peden tomar las celdas, y corresponden a los siguientes estados:

Estado de la Celda	Valor Asignado
Despoblado	0
Levemente Construido	1
Medianamente Construido	10
Densamente Construido	100

Estos valores fueron definidos de diferente orden de magnitud para facilitar la caracterización de las reglas de transición. Ademas las reglas de transición deben respetar la restriccion de que las celdas se pueblan de forma paulatina, osea que un 0 no puede bricar a ser 10 o 100 en un solo paso, ni un 100 involucionar en 0, en otras palabras si $S(t)$ es la funcion de estado en el tiempo para una celda dada, $S(t)$ es monotónica no drecreciente.

El autómata fue implementado con librerías de arc-info, en formato Arc Macro Language (AML). No es un autómata puro, en el sentido de que las reglas de transición no sólo dependen del valor de las celdas vecinas, sino tambien de factores "geográficos". Estos factores fueron los siguientes:

- Distancia a las vías de comunicación
- Pendiente del terreno
- Uso de suelo (Agrícola o Bosque)
- Distancia a la mancha urbana

Escogimos estos factores entre otras razones por la disponibilidad de datos. Su inclusión en las reglas de transición determina qué parámetros servirán para calibrar el modelo. Otros factores como servicios de agua, luz y teléfono, no fueron considerados porque no contamos por el momento con esa información, pero el código del automata puede ampliarse para incluir estos y otros factores de relevancia.

Los autómatas celulares están basados en operaciones realizadas con los valores de las celdas aledañas, como por ejemplo la "suma" de todos los vecinos. El tipo de vecindad varía con el caso a tratar, tomando diferentes formas y tamaños, nosotros probamos algunas variantes y finalmente optamos por una vecindad en forma de cruz de dos cuadros de radio.

Las reglas de transición de estado, son una mezcla entre interacción local y modificadores globales, por ejemplo para pasar del estado 0 al 1 hay tres posibles mecanismos, difusión por vecinos, dispersión por las vías de comunicación y expansión por densidad, la figura 3.1 muestra el diagrama de flujo que implementa estos tres mecanismos.

Una vez establecidos los mecanismos de transición le agregamos una función aleatoria, para permitir o no que una celda cambie de valor. Esto representa de alguna forma nuestra imposibilidad para saber los deseos o motivaciones particulares de cada persona, familia, o empresa.

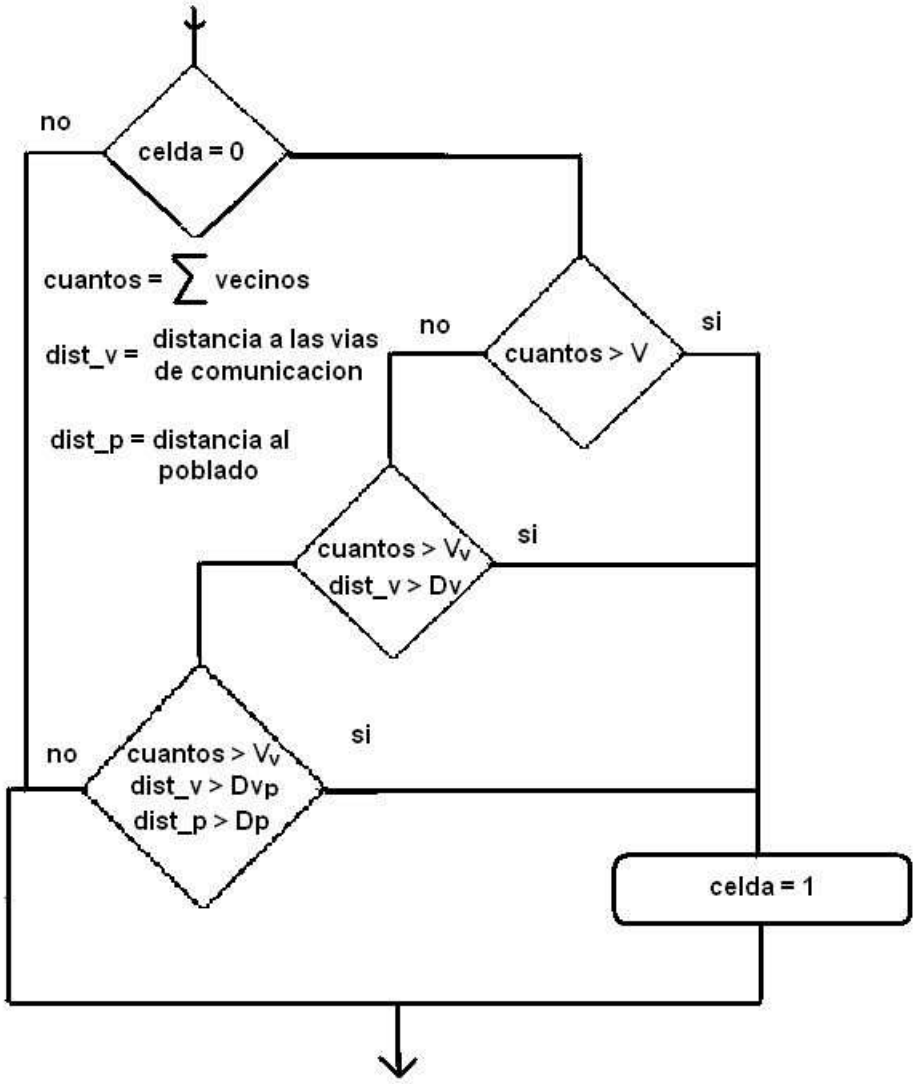

Figura 3.1: Diagrama de flujo para las reglas de transición. Los valores de V, Dv, Vv, Dv_p, y Dp son los parámetros que permiten ajustar el modelo.

3.2 El Calibrador

Para hacer posible la calibración del autómata se desarrolló un algoritmo evolutivo que consta de generaciones de 8 individuos con 7 parámetros cada uno, cada nueva generación esta formada por los dos mejores de la generación anterior, cuatro hijos directos de esos primeros dos y dos mutaciones, para un total de ocho individuos. En la figura 3.2 se muestra un esquema del ciclo del "calibrador".

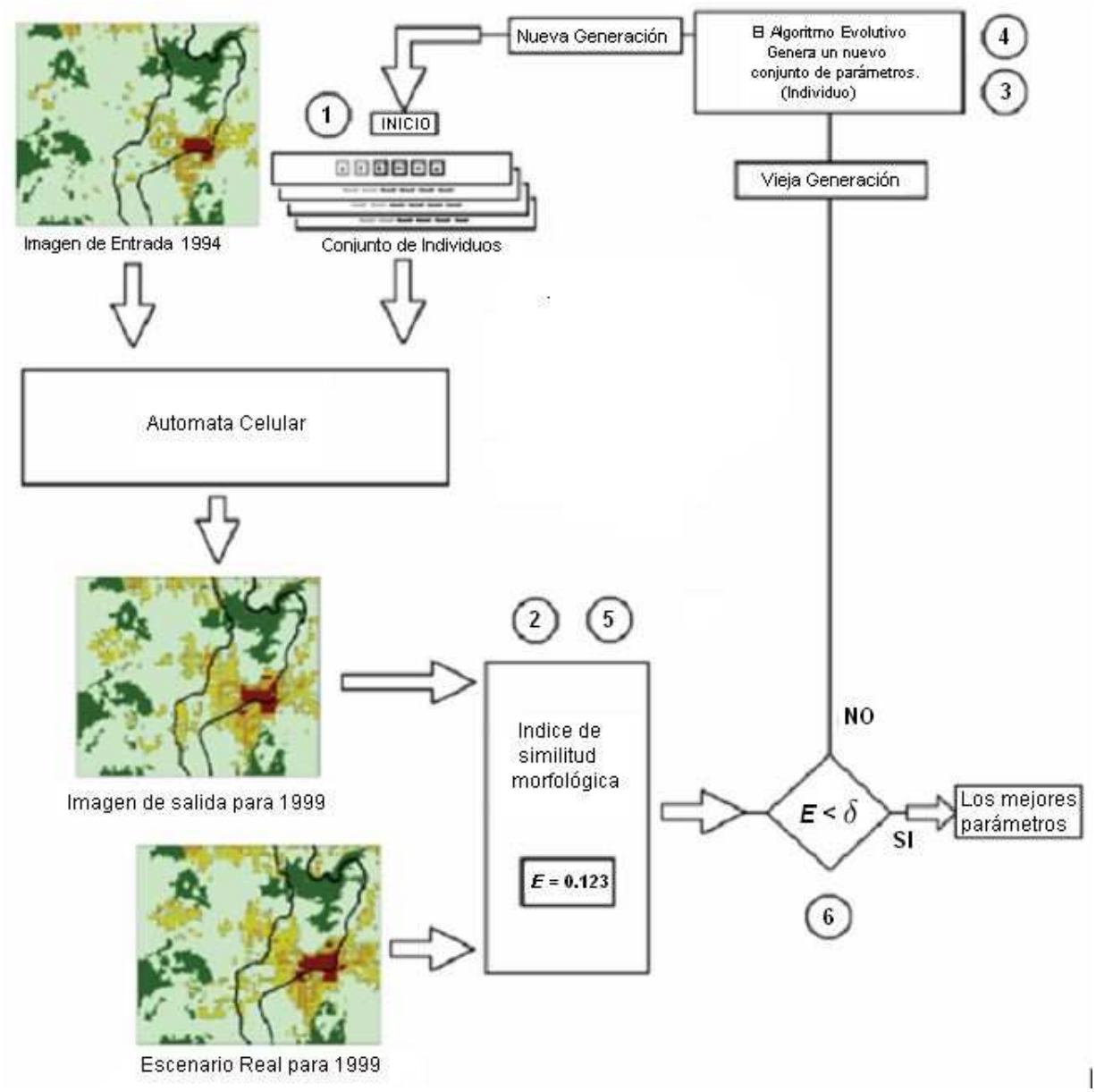

Figura 3.2: Esquema del ciclo calibrador, que evoluciona los parámetros del autómata.

Este algoritmo se programó en Java y funciona de tal manera que activa procesos de arc-info para probar combinaciones de parametros, y mientras tanto genera gráficas para monitorear el índice de adecuación de los parámetros mas adaptados. El índice que utilizamos para calificar el comportamiento de los parámetros fue el descrito en la sección 2.5.

3.3 Caso de Estudio: Topilejo

El pueblo de Topilejo se encuentra en lo que se denomina suelo de conservación, en el sur del distrito federal. En esta zona predominan las tierras de cultivo y los bosques, es una zona estratégica para la recarga del manto acuífero de la ciudad de México. El pueblo esta dividido por la autopista de cuota que va hacia Cuernavaca, y su desarrollo es de particular interés, puesto que el "suelo de conservación" que abarca gran parte del sur del distrito federal, esta en riesgo por el crecimiento de asentamientos como este.

Para inicializar el autómata que modela el crecimiento y densificacion de topilejo fue necesario obtener una serie de datos, los cuales se generaron partir de fotografías aéreas (ortofotos), e imágenes de satélite. En la figura 3.3 se muestra una fotografía aérea tomada en el año de 1994.

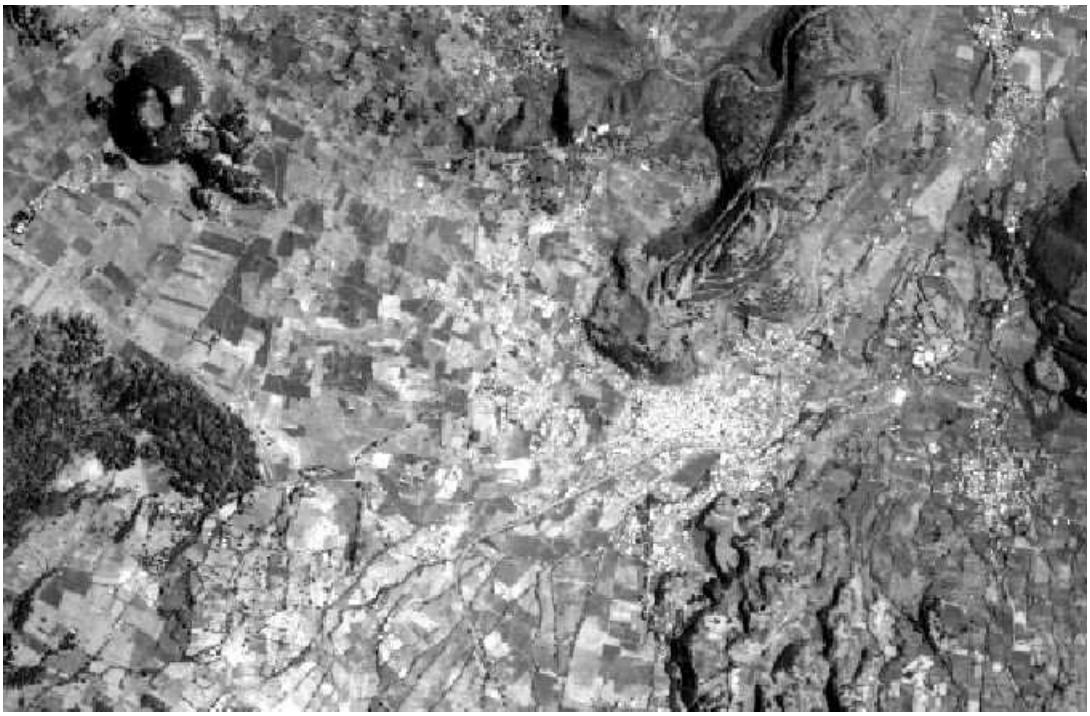

Figura 3.3: Fotografía aérea de Topilejo en 1994

Como se estableció previamente cada celda puede tomar 4 diferentes valores que corresponden a los siguientes estados:

- Despoblado
- Levemente Construido
- Medianamente Construido
- Densamente Construido

El trabajo de clasificación se hizo a mano, a partir de ortofotos de 1994, 1999, y 2002 de Topilejo y sus alrededores. Utilizamos una retícula cuadrada de 100x100mts. Esta manera de clasificar fue rudimentaria y se deben explorar métodos para automatizarla, por ejemplo, mediante redes neuronales o filtros de Markov. Esto con el fin de poder procesar gran cantidad de datos de manera sistemática, reduciendo el tiempo de procesamiento de datos previo a la utilización del autómata. En las figuras 3.4, 3.5 y 3.6 se muestra el resultado de esta clasificacion, para los años 1994, 1999 y 2002 respectivamente.

Las vías de comunicación terrestre las obtuvimos en formato shapefile, y hubo que agregar algunas calles que faltaban, pero con un poco de paciencia se completó el entramado de calles. La figula 3.7 muestra el resultado de esto.

El autómata considera el entramado de vías de comunicación como estático, dado que haremos progresiones en periodos relativamente cortos de tiempo. Aun así hubo zonas donde surgieron calles con mucha velocidad, por lo cual en el futuro debe considerarse como un proceso dinámico, abriendo esto una línea de investigación en la que se puede profundizar.

Las áreas cubiertas de bosque se obtuvieron de manera similar, clasificando a partir de la imagen, la automatización de este proceso también es factible, mediante indices de vegetacion, obtenidos a partir de imagenes multiespectrales. En la figura 3.8 se muestra la digitalización del bosque en el área de estudio.

Y por ultimo la pendiente para cada punto la generamos a partir de las curvas de nivel mediante un algoritmo de interpolación que provee arc-info, el resultado de este puede observarse en la figura 3.9.

Una vez obtenidos los datos se prosiguió a correr el "calibrador", aquí cabe mencionar que la adquisición de los datos en el formato adecuado, fue la parte mas laboriosa de la investigación, ya que la calibración fue automatizada por el algoritmo evolutivo.

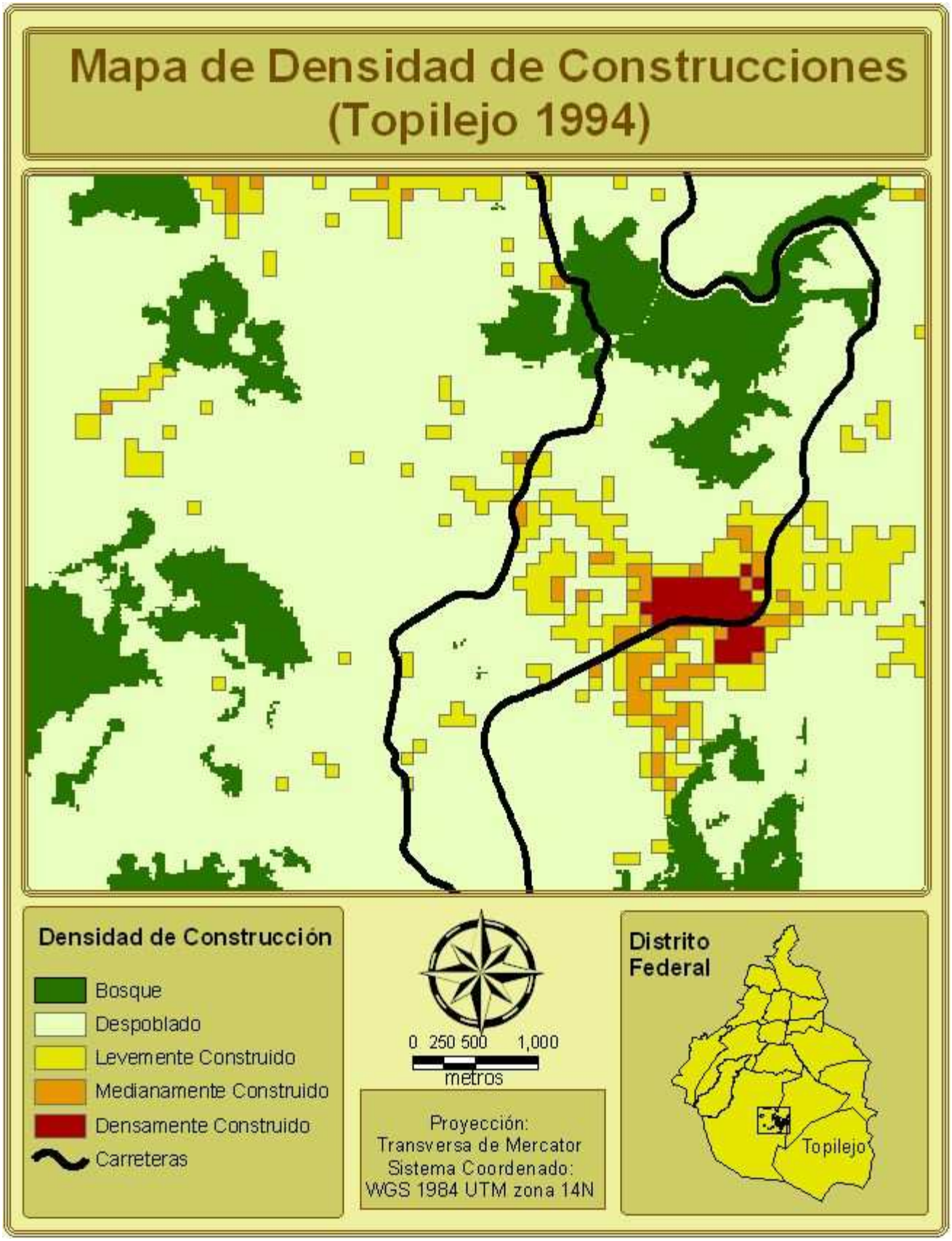

Figura 3.4: Mapa de densidad de construcciones en Topilejo en el año de 1994

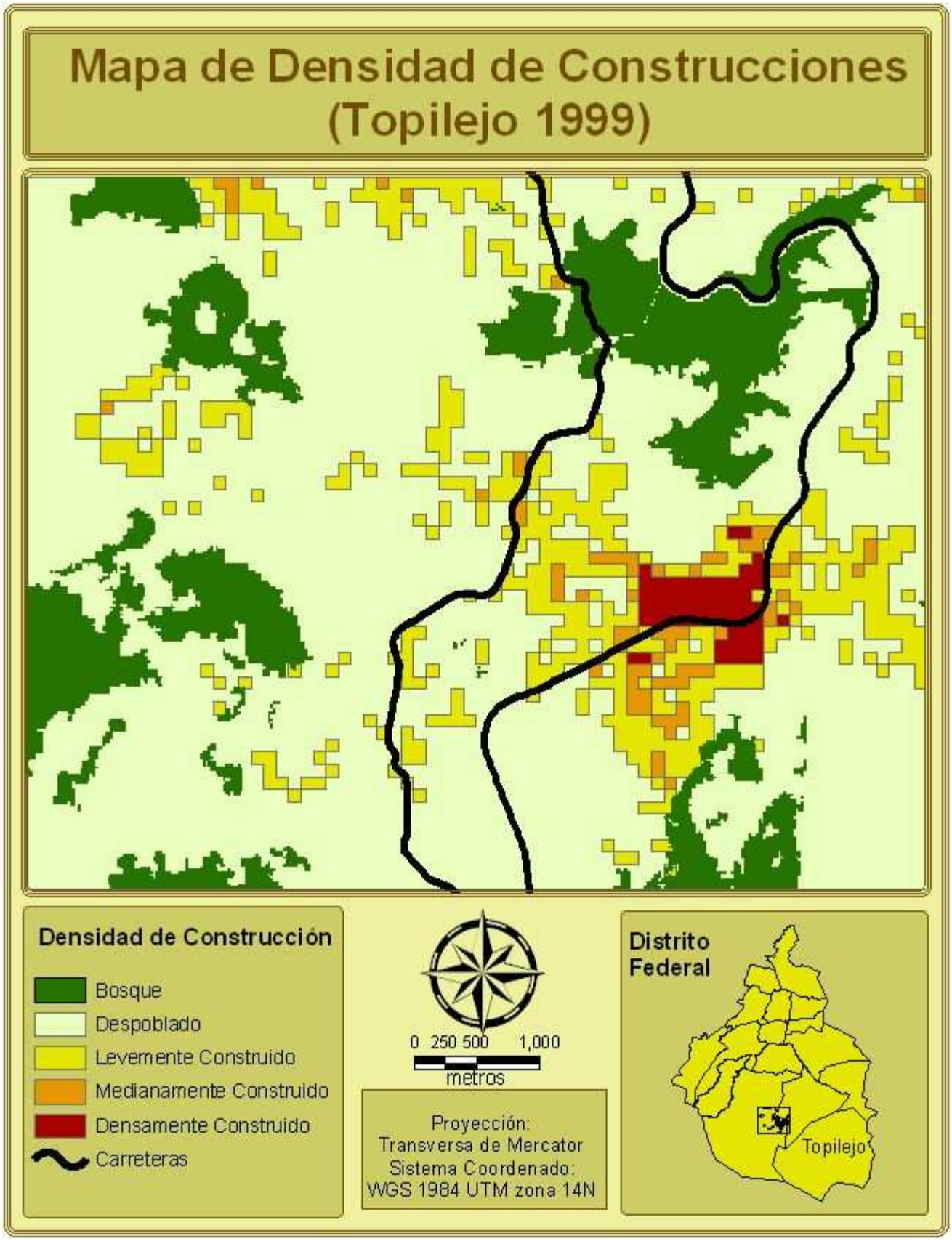

Figura 3.5: Mapa de densidad de construcciones en Topilejo en el año de 1999

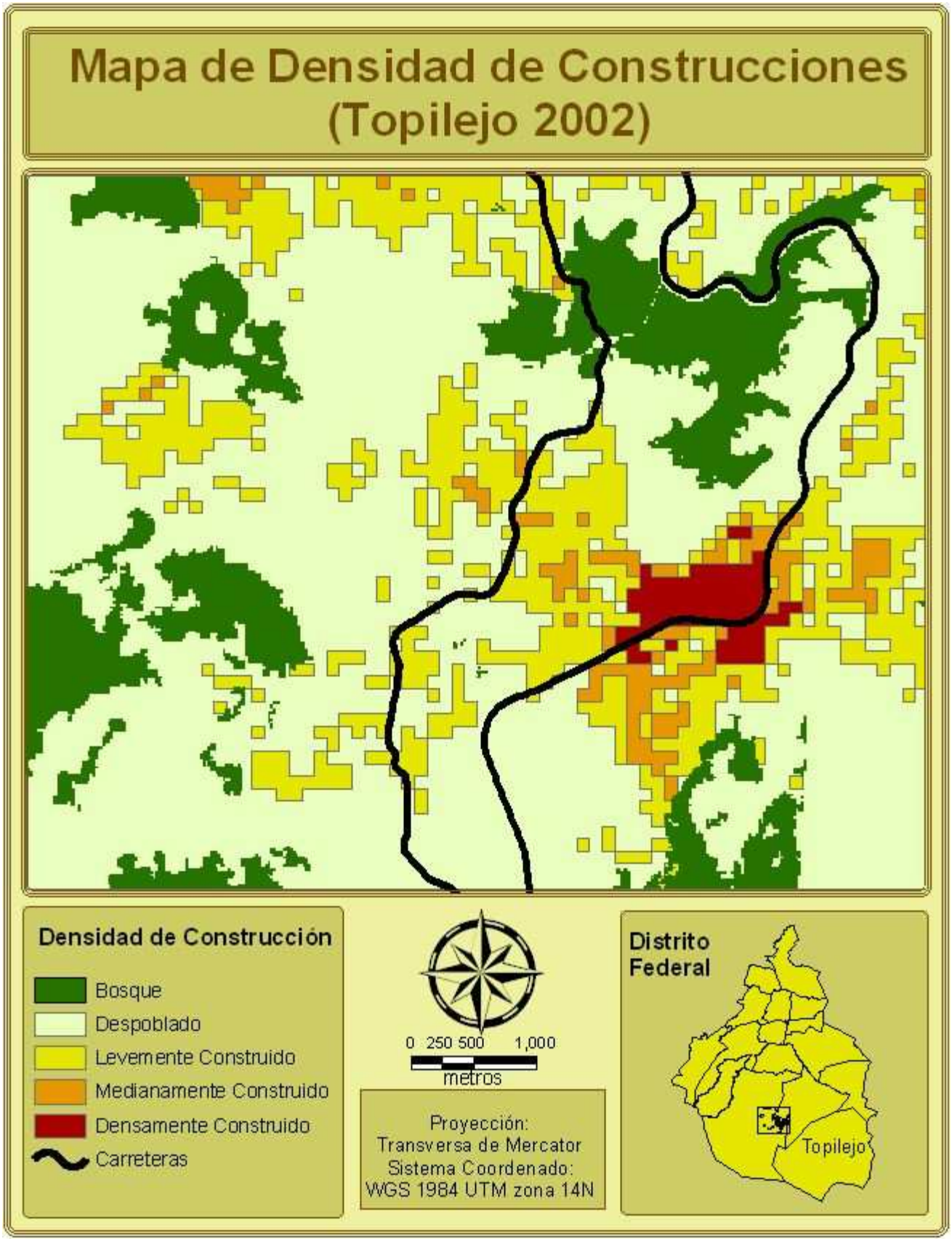

Figura 3.6: Mapa de densidad de construcciones en Topilejo en el año de 2002

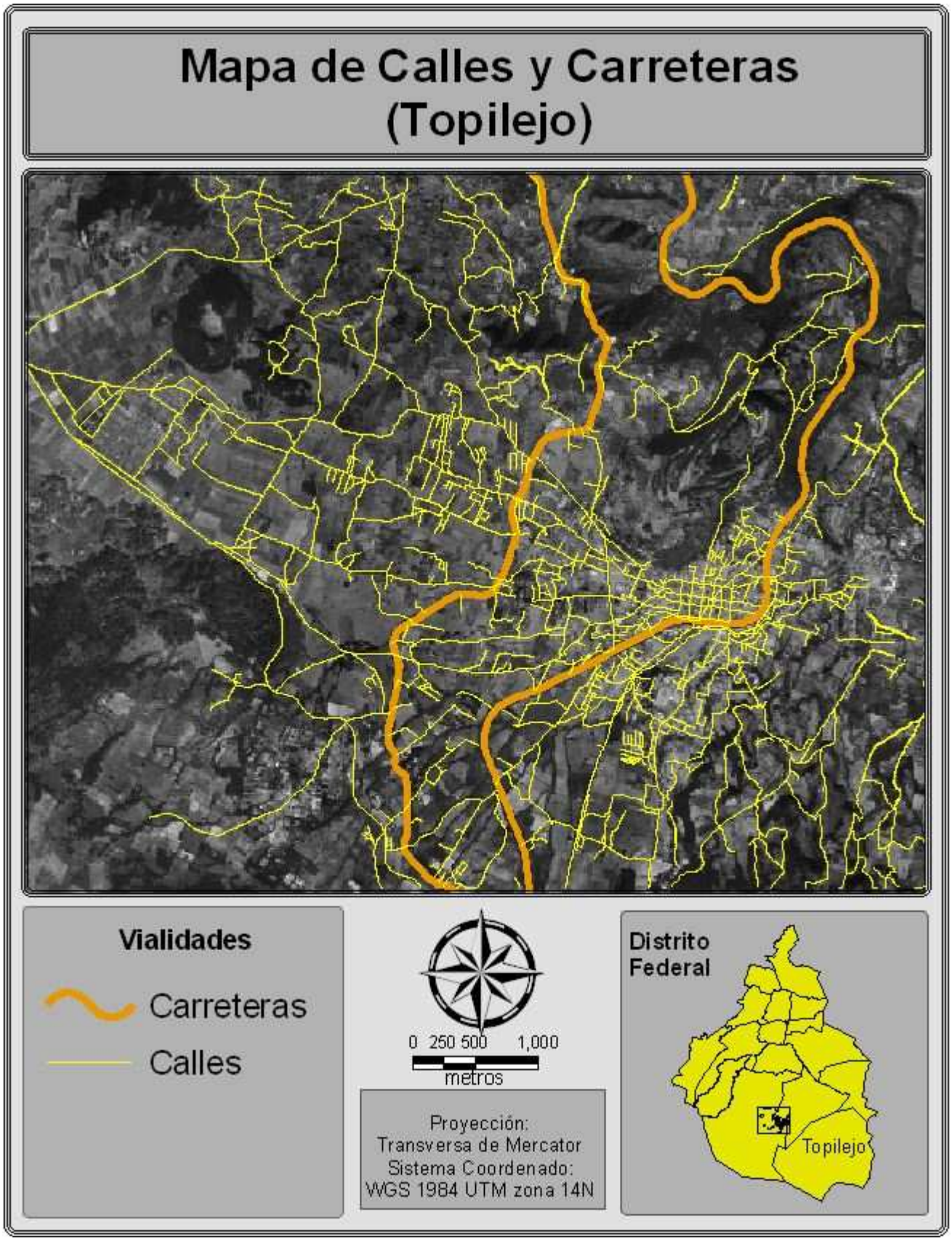

Figura 3.7: Mapa de vias de comunicacion terrestre

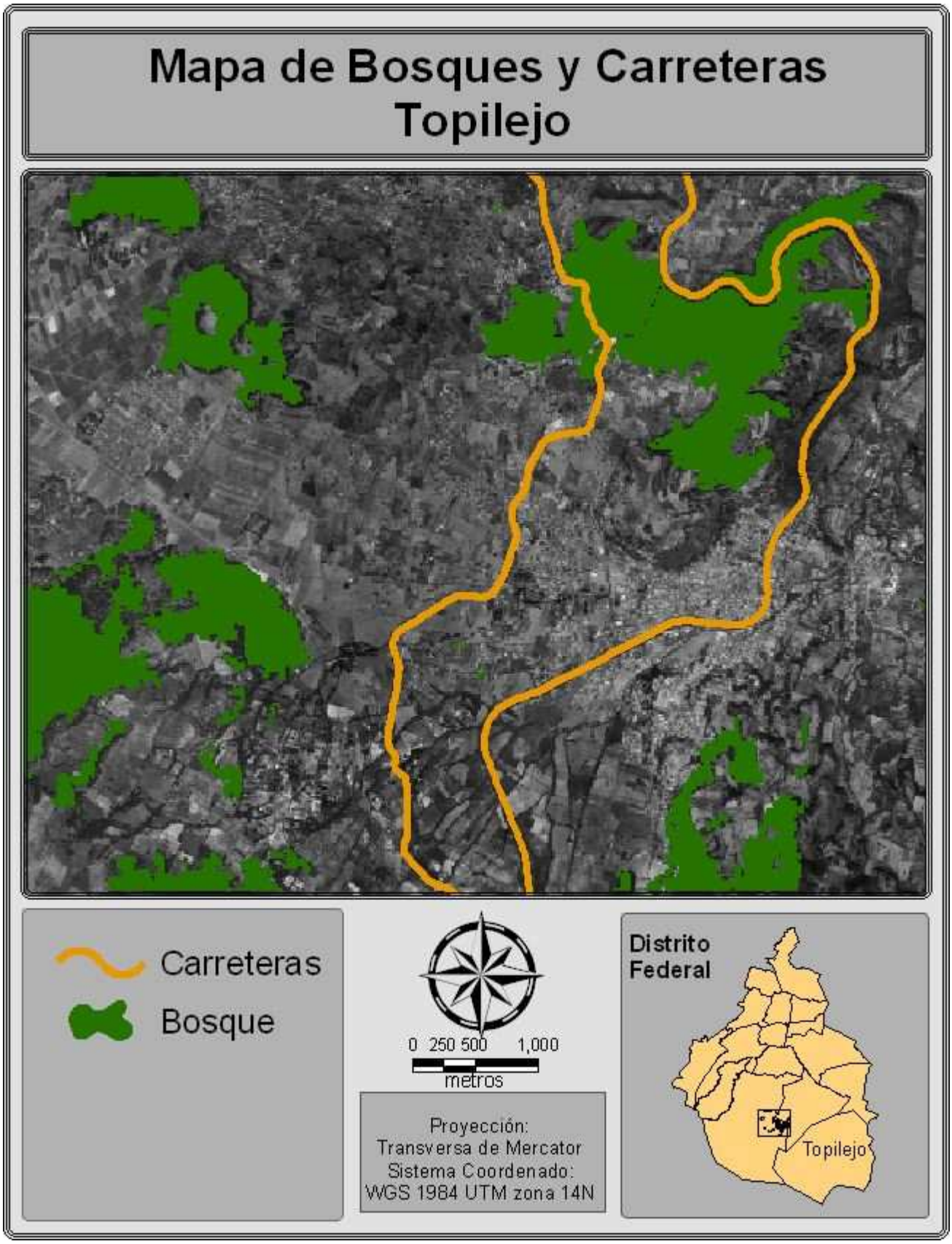

Figura 3.8: Mapa de Bosques de la Zona de Topilejo.

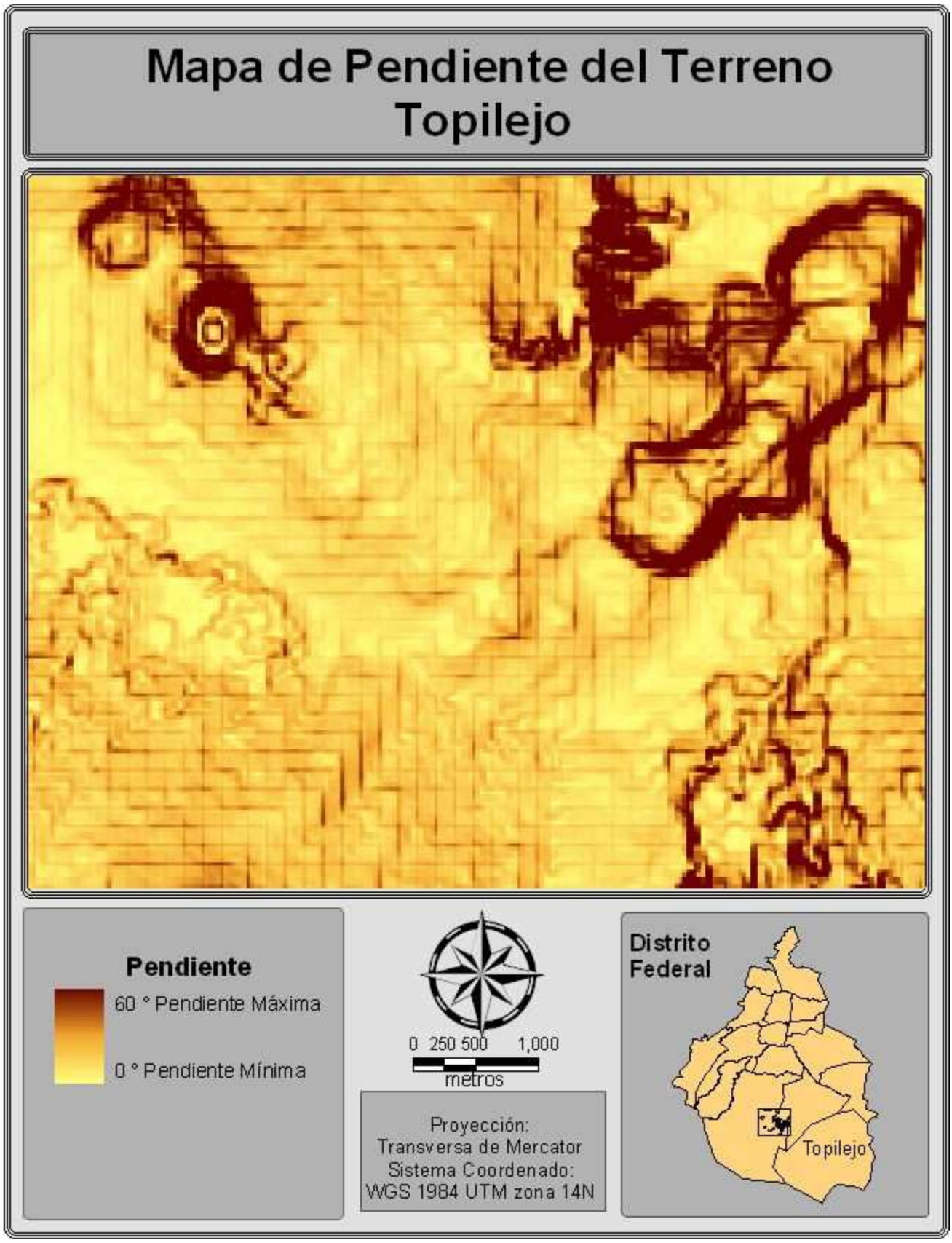

Figura 3.9: Mapa de Pendiente en la Zona de Topilejo.

Capítulo 4

Resultados

Esta investigación pretendía dos fines concretos; uno, crear un modelo base para correr simulaciones de crecimiento urbano, y el otro, probar un algoritmo evolutivo, como método de calibración. En esta sección de resultados, comentaremos lo logrado en estos dos aspectos.

El algoritmo evolutivo que se utilizó consta de generaciones de 8 individuos, de cada generación se toman los dos mejores y se reproducen, para crear la siguiente generación.

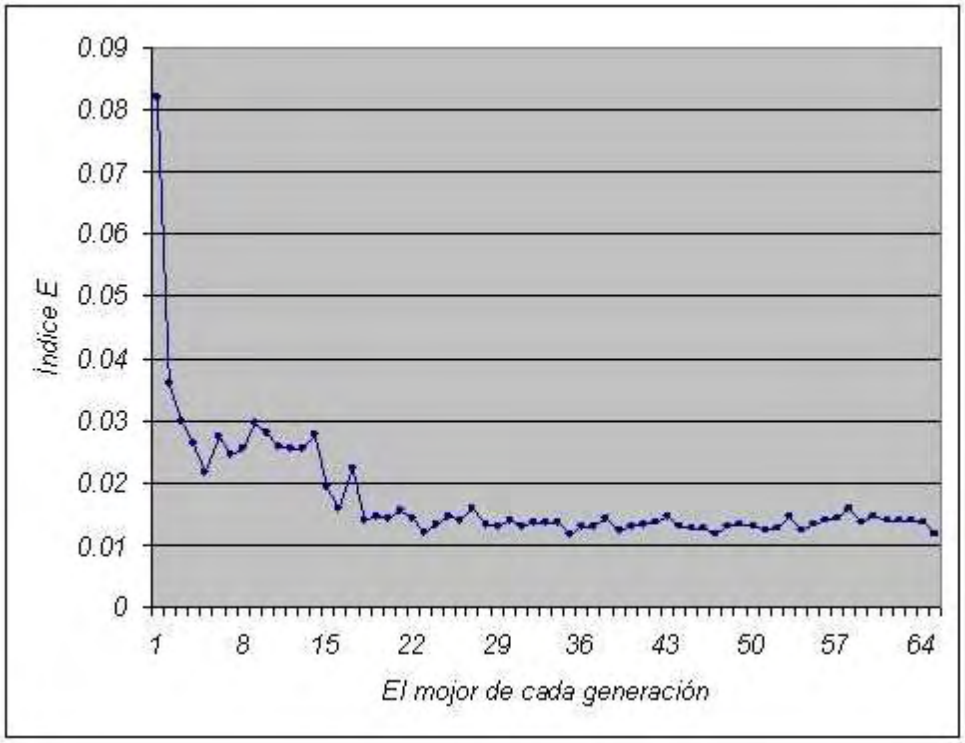

Figura 4.1: Desempeño del algoritmo calibrador

Este método de calibración, probó ser una herramienta de utilidad, pues en aproximadamente dos semanas de "evolución", se obtuvieron buenos resultados. Esto es un gran avance si consideramos que probar todas las combinaciones de parámetros nos hubiera llevado al menos un millón de días, aproximadamente 2700 años.

En la figura 4.1 se muestra una grafica del desempeño del algoritmo evolutivo. Cada punto, en la gráfica representa al "mejor" de cada generación, para las primeras 65 generaciones. La grafica muestra como el índice E baja rápidamente, hasta valores cercanos a 0.01, lo que significa que la salida del autómata es muy cercana a lo esperado. O sea que tenemos combinaciones de parámetros muy adaptados a Topilejo.

Dado que el modelo no es determinista, una vez encontrado un conjunto de parámetros "adaptados a Topilejo" para el período 1995-1999, se requiere llevar a cabo cierto número de realizaciones para observar su comportemiento y evaluar el promedio del índice E. La figura 4.2 muestra el resultado de 50 realizaciones con estos parámetros para dicho periodo. Cada punto representa el índice de similitud entre la "predicción" del modelo y la situación observada para 1999.

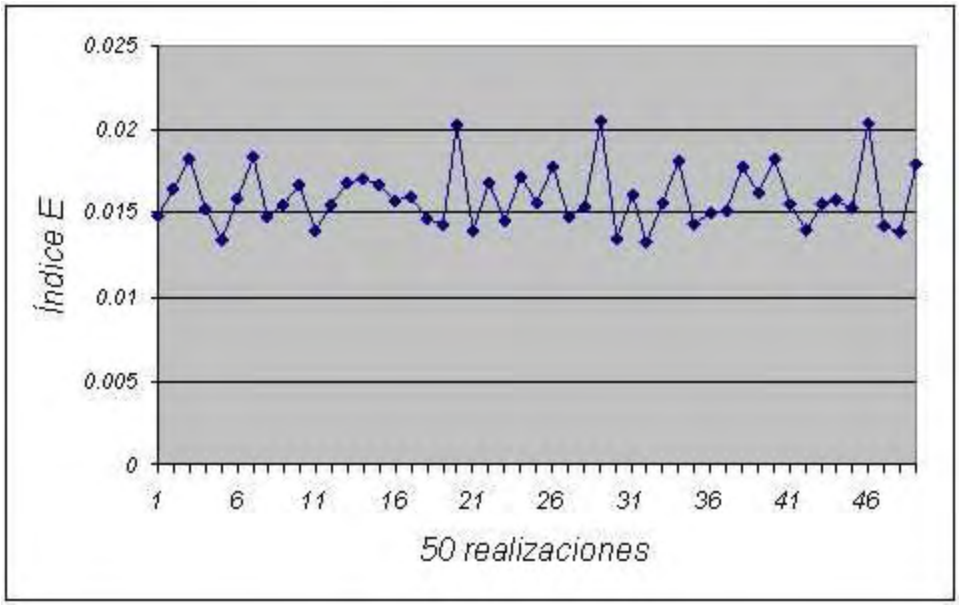

Figura 4.2: El mismo conjunto de parámetros para el periodo 1994-1999

Encontramos que el promedio del índice E para estas 50 realizaciones fue

$$\langle E(O_{1999}, A_{1999})\rangle = 0.016 \tag{4.1}$$

donde O_i y A_i denotan la situación observada y la "predicción" del modelo para el año i, respectivamente. Esto nos da una idea de que tanto logra el modelo ajustarse a lo observado en este periodo y en esta región.

La desviación estándar de estos datos fue $D = 0.0017$ lo cual nos indica la consistencia del modelo. El modelo base para simular la urbanización, debe juzgarse en tanto permita hacer algún tipo de "predicción". Para verificar lo anterior tomamos como condición inicial Topilejo en 1999, corrimos el modelo con los parámetros obtenidos mediante el algoritmo evolutivo, y comparamos el resultado del autómata con la situación observada en Topilejo en el año 2002. En la figura 4.3 se muestra esta comparación en 50 realizaciones del modelo para los mismos parámetros.

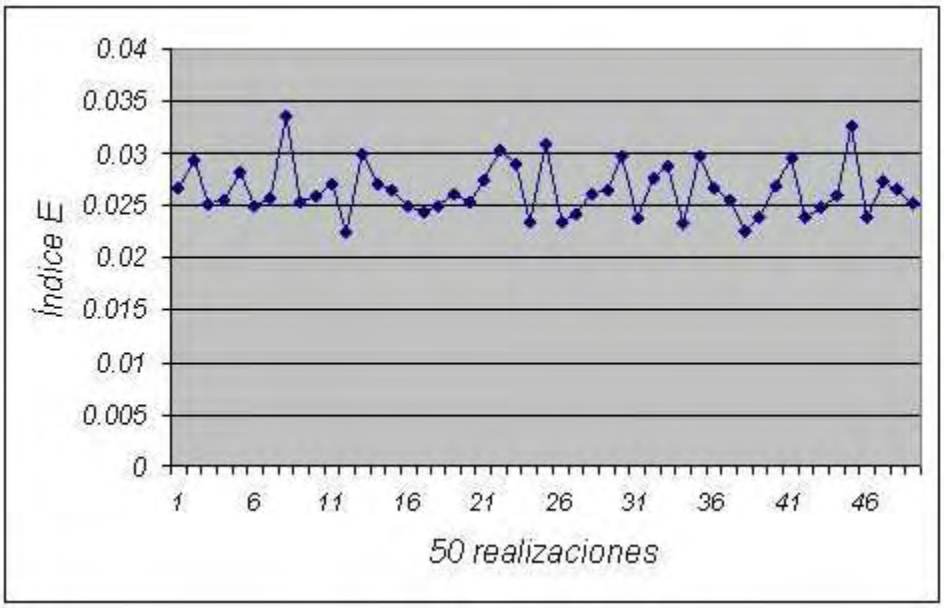

Figura 4.3: El mismo conjunto de parámetros para el periodo 1999-2002

El promedio del índice de similitud, E, entre las 50 realizaciones para el 2002 fue

$$\langle E(O_{2002}, A_{2002})\rangle = 0.026 \tag{4.2}$$

y su desviación estándar $D = 0.0025$.

Para darse una idea de que significan los resultados arriba expuestos, el índice de similitud para los escenarios reales de 1999 y 2002 fue

$$E(O_{1999}, O_{2002}) = 0.079, \tag{4.3}$$

y entre 1995 y 1999,

$$E(O_{1995}, O_{1999}) = 0.052, \tag{4.4}$$

Aquí cabe mencionar que el índice fue construido explícitamente para calibrar el modelo y no como una cantidad física del sistema. Y la calibración se puede juzgar visualmente en la figura 4.4, donde se puede observar la semejanza entre la salida del modelo y la situacion observada en el año 2002.

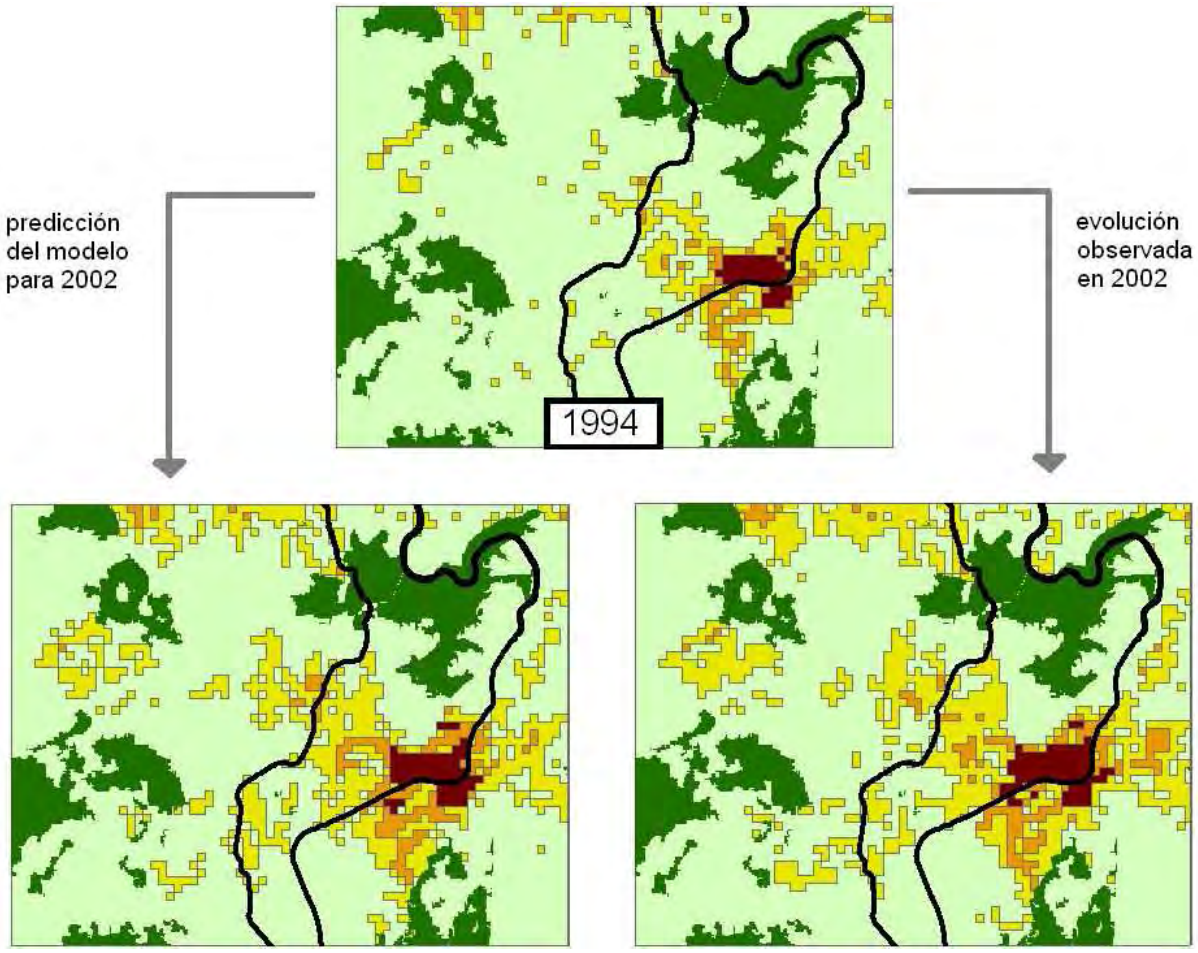

Figura 4.4: Comparación entre la salida del modelo y la situación observada para 2002. El área en color verde oscuro representa zonas de bosque, mientras que las zonas de colores amarillo, naranja y rojo representan la densidad de construcciones.

Capítulo 5

Conclusiones

Siendo la calibración un factor central para el buen desempeño de modelos como este, consideramos que la implementación de algoritmos evolutivos es de utilidad como herramienta calibradora. Sabemos de métodos de optimización que muestran resultados interesantes como los de gradientes, pero estos no son aplicables a este tipo de modelo por su componente estocástica.

Por otra parte la medida que utilizamos fue exitosa en reconocer la forma, a cierta escala, y es importante recalcar que en los estudios futuros, se debe considerar la generalización de esta medida, para que el reconocimiento de la forma se haga en la escala relevante, dependiendo del tamaño del dominio y de las celdas.

Se obtuvieron buenos resultados, en el sentido de que con relativamente pocos factores se logro captar una buena parte de la dinámica observada en esta región. Para incluir en el modelo el conocimiento local de conocedores de la zona es posible implementar una variante de la tecnica strabo [21]. Ademas, aplicar el modelo a una región mas amplia que incluya todo el sur de la ciudad de México puede ayudar a entender y encausar el crecimiento en esta zona.

Para aplicar el modelo a una región mas amplia hay dos posibles caminos; uno, es considerar cada parámetro variable en todo el dominio y otro es dividir en subdominios y ajustar en las fronteras.

El hecho de que el índice de similitud entre el escenario real y el simulado nunca fue inferior a .01, muestra que hay una parte de la dinámica que no captamos en este modelo relativamente simple, por lo que este modelo puede tomarse como la base para seguir incluyendo más factores de los que se tengan datos históricos.

La selección de los factores que se incluyan al modelo debe hacerse con cuidado pues a mas factores incluidos mayor tiempo de computo necesario para calibrar, sin embargo con el avance en el poder de las computadoras será posible en términos de tiempo de cómputo incluir más factores, y en ese sentido seria importante tener acceso o generar datos georeferenciados para

su futura utilización.

La utilización de tecnologías libres como Java, Eclipse, JSci, y otras fueron fundamentales para realizar este trabajo y seria un gran avance montar el autómata celular en tecnología libre, como GRAS o GeoTools.

Para fines de este modelo consideramos la red de calles y carreteras y los bosques, como invariantes durante el tiempo de simulación. Esto puede mejorarse acoplando este modelo con otros que representen la dinámica de creación de calles, o de fragmentación de bosques.

Por otra parte el proceso de clasificación de fotografías aéreas históricas para darle entrada al modelo se hizo a mano y celda por celda, lo cual podría automatizarse mediante la implementación de algoritmos capaces de reconocer donde hay casa(s) y donde no. Dos posibilidades en este sentido son las redes neuronales y los filtros de Markov.

Si bien es claro que hay camino por andar para contar con un modelo confiable, los resultados obtenidos son alentadores. Y esperamos que sus futuras aplicaciones puedan ser de utilidad en el entendimiento de la dinámica de crecimiento de esta muy querida y muy sufrida ciudad.

Apéndice A

Automata Celular

Para correr Autómata se utilizaron tres scripts en AML (Arc Macro Language): Autómata, Mide, y Borra a continuación se muestran sus códigos

A.1 Automata.aml

Este código toma como entradas, varios ARC-grids y en base a las reglas de transición y algunos parámetros calcula el desparramamiento de construcciones para un numero dado de iteraciones o años.

```
&ARGS cuantas, vecino_0, vecino_carretera_0, vecino_pueblo,
buffer_carretera_0, buffer_pueblo, buffer_carr_pueb, grid_salida,
grid_meta, profundidad_del_indice, iteraciones

/* este aml necesita :
/*   manchaur0 (la situacion inicial)
/*   pendiente (lapendiente)
/*   bosque    (un grid que delimita lo que es bosque)
/*   dist_vias (es un eucdistance de las vias de comunicacion)
/*   fact0     (la situacion final)

&sv vecino_1 = 62
&sv vecino_carretera_1 = 24
&sv buffer_carretera_1 = 50

&sv vecino_10 = 666
&sv vecino_carretera_10 = 200
&sv buffer_carretera_10 = 30

&sv wagri =  .8 /* peso agricultura-bosque
&sv wpend =  .99  /* peso pendiente
```

```
&ty %cuantas%
/*&sv iteraciones = 3
grid
&r borra manchaur0
&do m = 1 &to %iteraciones%
&r borra manchaur%m%
&sv l = %m% - 1
&r borra vecino%l%
&end
manchaur0 = %grid_salida%
&do I = 1 &to %iteraciones% /* Actualizaciones del automata
&sv j = %I% - 1

setcell 100
vecino%j% = focalsum(manchaur%j%, circle, 3)
dist_pueb = eucdistance(manchaur%j%)
setwindow manchaur0
setcell 100
randomgrid = rand()
docell
  if ( (que_tanto >= .7) && (randomgrid <= %cuantas%) )
  begin
    if (manchaur%j% == 0)
      if (vecino%j%  >= %vecino_0%)
            manchaur%I% = 1 /* difusion por vecino
      else if (vecino%j% >= %vecino_carretera_0%
            && dist_vias <= %buffer_carretera_0%)
            manchaur%I% = 1 /* dispersion cercana a la carretera
            else if (vecino%j% >= %vecino_pueblo%
                     && dist_pueb <= %buffer_pueblo%
                     && dist_vias <= %buffer_carr_pueb%)
                     manchaur%I% = 1       /* dispersion cercana al pueblo
else manchaur%I% = 0
    if (manchaur%j% == 1)
      if (vecino%j%  >= %vecino_1%)
           manchaur%I% = 10 /* difusion (engorda los 10's)
      else if (vecino%j%  >= %vecino_carretera_1%
                  && dist_vias <= %buffer_carretera_1%)
                  manchaur%I% = 10 /* dispersion cercana a la carretera
      else manchaur%I% = 1
    if (manchaur%j% == 10)
      if (vecino%j%  >= %vecino_10%) manchaur%I% = 100 /* engorda los 100's
```

```
      else if (vecino%j%  >= %vecino_carretera_10%
                  && dist_vias <= %buffer_carretera_10%)
                  manchaur%I% = 100
      else manchaur%I% = 10
    if (manchaur%j% == 100) manchaur%I% = 100

if (randomgrid <= .0003 && manchaur%j% == 0 ) manchaur%I% = 1
  end
  else manchaur%I% = manchaur%j%
end

&r borra randomgrid
&r borra dist_pueb
&end
/* end do (for)
q /*grid

&r mide %grid_meta%, manchaur%iteraciones%, %profundidad_del_indice%
```

A.2 mide.aml

Este código calcula el indice de similitud morfologica descrito en la seccion 2.5.

```
&ARGS fact, try, n
grid
  &r borra mancha_try
  &r borra mancha_fact
  &do m = 1 &to %n%
    &r borra al_cuadrado_%m%
  &end
  mancha_try = con (isnull (%try%) , 0, con (%try% >= 1, 1, 0))
  mancha_fact = con (isnull (%fact%), 0, con (%fact% >= 1, 1, 0))
  &sv ancho = [show $$NCOLS]
  &sv alto = [show $$NROWS]
  &sv dos_ala_i = .5
  &do i = 1 &to %n%
  &sv dos_ala_i = %dos_ala_i% * 2
  &sv alto_block = [truncate [calc %alto% / %dos_ala_i%] ]
  &sv ancho_block = [truncate [calc %ancho% / %dos_ala_i%] ]
```

```
  &sv maximo = %alto_block% * %ancho_block%
  al_cuadrado_%i% =
     (sqr (float (
     (float (blocksum (mancha_try, rectangle, %ancho_block%, %alto_block%))
      -float (blocksum (mancha_fact, rectangle, %ancho_block%, %alto_block%)))
      / %maximo% )) )
&end
q /*grid


tables
  &do j = 1 &to %n%
    select al_cuadrado_%j%.sta
      &sv este = [ sqrt  [show record 1 mean] ]
      &sv indice_%j% = %este%
      &ty Indice_%j% = %este%
  &end
  /*&sv indice = %indice_4% * 0.25 + %indice_3% * 0.25
                + %indice_2% * 0.25 + %indice_1% * 0.25
  &sv indice = %indice_1% * 0.5 + %indice_4% * 0.5
  //&sv indice = %indice_4%
  /* sends signal
  &ty ____________________________________________________________________
  /* manda indice
  &ty %indice%
q /* tables
```

A.3 Borra.aml

Este código borra los grids auxiliares.

```
&ARGS file
KILL %file% ALL
```

Apéndice B

Algoritmo Evolutivo

El Algoritmo Genético esta programado en java y se constituye de ocho clases a continuación el código:

B.1 Evolucionan.java

Este código se encarga de evolucionar los parámetros del autómata.

```
import java.io.File;
import java.util.Collections;
import java.util.Random;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

/*
 * Created on 22-jul-2004
 *
 * @author Fidel Serrano
 *
 * cuantas   = {0-.7}    .12
 * vecino_0   = {1-24}   12
 * vecino_pueblo_0     = {1-24}    12 *NUEVO*
 * vecino_carretera_0  = {1-24}   11
 *                 el minimo numero de vecino necesario para que:
 *                 si estas en el bufer de la carretera, puedas cambiar
 *                 al siguiente estado
 * buffer_carretera_0  = {10-300}  150
 * el tamanio de buffer de la carretera en metros
 * buffer_pueblo       = {10-444}      *NUEVO*
```

```
 * buffer_carr_pueb    = {10-555}      *NUEVO*
 * primer intento....SoloUnos
 *
 *  los reales (0-1)los convertire en enteros (0-100)
 */


public class Evolucionan {
  ListaDeIndividuos vivos;
  ComparaIndividuos comparador;
  CodigoGenetico adn;
  Ventana ventana;
  Ventana chidos;
  int radio;
  double[] estosDatos;
  double indiceDelBueno, indiceUnBueno;
  private String gridSalida;
  private String gridMeta;
  private int iteraciones;
  private int profundidadDelIndice;
  private String quetanto;
  public Evolucionan() {
    adn = new CodigoGenetico();
    comparador = new ComparaIndividuos();
    vivos = new ListaDeIndividuos();
    int[] unAdn = {19, 7, 3, 8, 5, 237, 5};
    int[] otroAdn = {19, 10, 1, 2, 5, 333, 5};
    vivos.add(0, new Individuo(unAdn));
    vivos.add(1, new Individuo(otroAdn));
    //vivos.add(0, new Individuo(-3));
    //vivos.add(1, new Individuo(3));
    vivos.add(2, new Individuo());
    vivos.add(3, new Individuo());
    vivos.add(4, new Individuo());
    vivos.add(5, new Individuo());
    vivos.add(6, new Individuo());
    vivos.add(7, new Individuo());
    ventana = new Ventana("Guardar Todo", "Todos");
    chidos = new Ventana("Guardar Resultados", "Los Buenos");
    ventana.show();
    chidos.show();
  }
  private void empiezen() {
```

```
  // TODO Auto-generated method stub
  inicializate();
  int vueltas=0;
  indiceDelBueno = 1.0;
  vivos.getI(0).indice = 2;
  while (vivos.getI(0).indice > .01){
    vueltas++;
    //aparea();
    // esto debe generar hijo1 e hijo2 como dos
    //combinaciones diferentes de los papas
    aparea();
    muta();
    noTePases();
    quienSobrevive();
    chidos.escribe(vivos.getI(0));
  }
  ventana.escribe("en " + vueltas + " vueltas");
}
public static void main(String[] args) {
  Evolucionan saltinbanquis = new Evolucionan();
  saltinbanquis.empiezen();
}
private void quienSobrevive() {
  ponIndices();
  Collections.sort(vivos, comparador); // ordena los 8
  int[] uno = vivos.getI(0).ADN;
  String quien = "" + uno[0];
  for (int esteGen = 1; esteGen < uno.length; esteGen++){
    quien = quien + ", " + uno[esteGen];
  }
  Corre.huella(quetanto, quien);
}
private void ponIndices(){
  Individuo este=null;
  indiceUnBueno = 1.0;
  Gen gen;
  for (int i = 0; i < vivos.size(); i++){
    este = vivos.getI(i);
    vivos.getI(i).indice = Corre.automata(
                      este.ADN, gridSalida, gridMeta,
                      profundidadDelIndice, iteraciones);
    if (vivos.getI(i).indice < indiceUnBueno){
      indiceUnBueno = vivos.getI(i).indice;
```

```
      Corre.guardar("unBueno");
      if (vivos.getI(i).indice < indiceDelBueno){
        indiceDelBueno = vivos.getI(i).indice;
        Corre.guardar("elBueno");
      }
    }
    ventana.escribe(vivos.getI(i));
  }
  ventana.escribe("---------------------------------" + "\n");
  String aqui ="";
  String uno_mas ="";
  quetanto ="";
  for (int j = 0; j < adn.tamao; j++){
    uno_mas = "" + Integer.toString(este.ADN[j]) + "_";
    aqui = aqui.concat(uno_mas);
  }
  chidos.escribe("!!!!!!!!!!!!!!!!!!" + aqui);
  quetanto = Double.toString(indiceDelBueno);
}

private void muta() {
  for (int i = 0; i < 4; i++){
    muta(i, i + 4);
  }
}
private void muta(int este, int aqui){
  ventana.escribe("mutare " + este + " " + aqui + "\n");
  Random escupe= new Random();
  Gen gen;
  int que_Tanto;
  for (int i=0; i<adn.tamao; i++){
    if (escupe.nextBoolean()){
      gen = adn.getGen(i);
      radio = (int)(gen.suIntervalo/(3*gen.brinco)) + 1;
      System.out.print("el radio es " + radio);
      que_Tanto = escupe.nextInt(radio) + 1;
      if (escupe.nextBoolean()){
        vivos.setADN(aqui, i, vivos.getI(este).ADN[i]
         + ((que_Tanto) * gen.brinco) );
      }else{
        vivos.setADN(aqui, i, vivos.getI(este).ADN[i]
        - ((que_Tanto) * gen.brinco) );
      }
```

```
      ventana.escribe("mutare el cromosoma " + i
      + " en " + que_Tanto + " brincos de "
      + gen.brinco + "\n")
    }else{
      vivos.setADN(aqui, i, vivos.getI(este).ADN[i]);
    }
  }
}
public void crossover(){
  Random escupe = new Random();
    int hastaAqui = escupe.nextInt(adn.tamao);
    for (int i = 0; i < hastaAqui; i++){
      vivos.setADN(2, i, vivos.getI(0).ADN[i]);
      vivos.setADN(3, i, vivos.getI(1).ADN[i]);
    }
    for (int i = hastaAqui; i < adn.tamao; i++){
      vivos.setADN(2, i, vivos.getI(1).ADN[i]);
      vivos.setADN(3, i, vivos.getI(0).ADN[i]);
    }
  }
public void aparea() {
  //Este metodo genera dos hijos seleccionando al azar cada
  //gen (de cualquiera de los papas)
  //y haciendo cross over... osea que si el hijo1.gen[i]
  //es de un papa => hijo2.gen[i] es del otro papa
  Random escupe = new Random();
  if (vivos.sonIguales(0,1)){
    ventana.escribe("AAAAAAAAA!!!!....MUTARE SI DIOS QUIERE \n");
    muta(0, 2);
    muta(1, 3);
  }else{
    for (int i = 0; i < adn.tamao; i++){
      switch (escupe.nextInt(5)){
        case 0 : {
          vivos.setADN(2, i, vivos.getI(0).ADN[i]);
          vivos.setADN(3, i, (int)((vivos.getI(0).ADN[i]
           + vivos.getI(1).ADN[i]) / 2));
          break;
        }
        case 1 :
        case 2 :
        case 3 : {
          vivos.setADN(2, i, vivos.getI(0).ADN[i]);
```

```
          vivos.setADN(3, i, vivos.getI(1).ADN[i]);
          break;
        }
        case 4 :
        case 5 :{
          vivos.setADN(2, i, vivos.getI(1).ADN[i]);
          vivos.setADN(3, i, vivos.getI(0).ADN[i]);
          break;
        }
      }
    }
  }
}
private void noTePases() {
  //para evitar que salga de sus limites
  for (int l=0; l<vivos.size(); l++){
    for (int o=0; o<adn.tamao; o++){
      if (vivos.getI(l).ADN[o]>adn.gen[o].max){
        vivos.setADN(l, o, adn.gen[o].max);
      }
      if (vivos.getI(l).ADN[o]<adn.gen[o].min){
        vivos.setADN(l, o, adn.gen[o].min);
      }
    }
  }
}
private void inicializate() {
  JFileChooser gridChooser = new JFileChooser();
  gridChooser.setFileSelectionMode(
                JFileChooser.DIRECTORIES_ONLY);
  gridChooser.setCurrentDirectory(
                new File("C:/eclipse/workspace/Limpio"));
  gridChooser.setDialogTitle("Escoje el grid de partida");
  int returnVal = gridChooser.showOpenDialog(ventana);
  if(returnVal == JFileChooser.APPROVE_OPTION) {
    gridSalida = gridChooser.getSelectedFile().getName();
    System.out.println("partiremos del grid " + gridSalida);
  }
  gridChooser.setDialogTitle("Escoje el grid meta");
  returnVal = gridChooser.showOpenDialog(ventana);
  if(returnVal == JFileChooser.APPROVE_OPTION) {
    gridMeta = gridChooser.getSelectedFile().getName();
    System.out.println("intentaremos crecer hasta el grid "
```

```
                        + gridMeta);
    }
    //  Get the message.
    iteraciones =Integer.parseInt(
         JOptionPane.showInputDialog("Numero de iteraciones"));
    profundidadDelIndice=Integer.parseInt(
                JOptionPane.showInputDialog(
                    "Profundidad del indice de similitud"));
  }
}
```

B.2 Corre.java

Este código es la interfaz para correr los scripts de AML como procesos desde el programa en java.

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;

/*
 * Created on 28-jul-2004
 *
 * @author Fidel Serrano
 *
 */

public class Corre {
//cuantas, vecino_0, vecino_carretera_0, vecino_pueblo,
//buffer_carretera_0, buffer_pueblo, buffer_carr_pueb
  public static double automata(int[] args, String gridSalida,
        String gridMeta, int profundidadDelIndice, int iteraciones) {
    double indice=1.0;
    Process elAutomata = null;
    try {
      System.out.println("aqui venimos ");
      //cuantas, vecino_0, vecino_carretera_0, vecino_pueblo,
      //buffer_carretera_0, buffer_pueblo, buffer_carr_pueb
      double arg0 = (double)args[0]/100.0;
      elAutomata = Runtime.getRuntime().exec("arc &r hazme_prueba "
```

```
    + arg0 + ", " +args[1]+ ", " +args[2]+ ", " +args[3]+ ", "
    + args[4]+ ", " + gridSalida + ", " + gridMeta + ", "
    + profundidadDelIndice + ", " + iteraciones);
    int que = 0;
    String laSenial=new String("_____________________________________
                               ____________________________________");
    String loQueEscupio = "nada";
    BufferedReader siDime = new BufferedReader(
                new InputStreamReader(elAutomata.getInputStream()));
    loQueEscupio = siDime.readLine();
    while (loQueEscupio.compareTo(laSenial)!= 0 ){
      loQueEscupio = siDime.readLine();
      System.out.println(loQueEscupio);
    }
    loQueEscupio = siDime.readLine();
    indice = Double.parseDouble(loQueEscupio);
    System.out.println(indice);
  } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
  elAutomata.destroy();
  try {
    elAutomata.waitFor();
  } catch (InterruptedException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
  }
  elAutomata.destroy();
  return indice;
}
public static void guardar(String aqui){
  Process guardando = null;
  try {
    guardando = Runtime.getRuntime().exec(
                                "arc &r Guardamelo " + aqui);

    BufferedReader siDime = new BufferedReader(
          new InputStreamReader(guardando.getInputStream()));

    String loQueEscupio = siDime.readLine();
    while (loQueEscupio!= null ){
      loQueEscupio = siDime.readLine();
```

```
      System.out.println(loQueEscupio);
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
  guardando.destroy();
  try {
    guardando.waitFor();
  } catch (InterruptedException e1) {
    e1.printStackTrace();
  }
  guardando.destroy();
}

public static void guarda(String aqui){
  Process guardame = null;
  try {
    System.out.print("estoy intentando guardar on guarda \n");
    guardame = Runtime.getRuntime().exec("arc &r guarda " + aqui);
    BufferedReader siDime = new BufferedReader(
              new InputStreamReader(guardame.getInputStream()));
    String loQueEscupio = siDime.readLine();
    while (loQueEscupio!= null ){
      loQueEscupio = siDime.readLine();
      System.out.println(loQueEscupio);
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
  guardame.destroy();
  try {
    guardame.waitFor();
  } catch (InterruptedException e1) {
    e1.printStackTrace();
  }
  guardame.destroy();
}
public static void estos(int i){
  Process guardame = null;
  try {
    System.out.print("estoy intentando guardar " + i + "\n");
    guardame = Runtime.getRuntime().exec("arc &r estos " + i);
    BufferedReader siDime = new BufferedReader(
```

```
                new InputStreamReader(guardame.getInputStream()));
      String loQueEscupio = siDime.readLine();
      while (loQueEscupio!= null ){
        loQueEscupio = siDime.readLine();
        System.out.println(loQueEscupio);
      }
      System.out.print("creo que pude guardar " + i + "\n");
    } catch (IOException e) {
      e.printStackTrace();
    }
    guardame.destroy();
    try {
      guardame.waitFor();
    } catch (InterruptedException e1) {
      e1.printStackTrace();
    }
    guardame.destroy();
  }
  public static void huella(String quetanto, String quien){
    Process guardame = null;
    BufferedWriter pon = null;
    try {
      pon = new BufferedWriter(new FileWriter(new File(
                    "C:/eclipse/workspace/Limpio/registro/"
                    + quetanto + "/parametros.txt") ));
      pon.write( quien );
      pon.close();
    } catch (IOException e1) {
      System.out.println("NO PUDE GUARDAR");
      e1.printStackTrace();
    }
    try {
      guardame = Runtime.getRuntime().exec(
                    "arc &r huella " + quetanto + ", " + quien);
      BufferedReader siDime = new BufferedReader(
                new InputStreamReader(guardame.getInputStream()));
      String loQueEscupio = siDime.readLine();
      while (loQueEscupio!= null ){
        loQueEscupio = siDime.readLine();
        System.out.println(loQueEscupio);
      }
    } catch (IOException e) {
      e.printStackTrace();
```

```
    }
    guardame.destroy();
    try {
      guardame.waitFor();
    } catch (InterruptedException e1) {
      e1.printStackTrace();
    }
    guardame.destroy();
  }
}
```

B.3 CodigoGenetico.java

Esta es la clase que contiene la estructura del código genético que tendran los Individuos

```
/*
 * Created on 22-jul-2004
 *
 * @author Fidel Serrano
 *
 */
public class CodigoGenetico{
  Gen[] gen = new Gen[5];
  int genSize = gen.length;
  public CodigoGenetico() {
    gen[0] = new Gen(20, 30);
    gen[1] = new Gen(2, 36);
    gen[2] = new Gen(2, 36);
    //gen[3] = new Gen(2, 36);
    //gen[4] = new Gen(44, 555);
    gen[3] = new Gen(10, 111);
    gen[4] = new Gen(55, 555);
    /*gen[7] = new Gen(10, 300);
    gen[8] = new Gen(10, 2400);
    gen[9] = new Gen(10, 300);
    gen[10] = new Gen(10, 300);
    gen[11] = new Gen(0, 100);
    gen[12] = new Gen(0, 100);*/
  }
  public Gen getGen(int j) {
    return (Gen)gen[j];
```

```
  }
}
class Gen {
  int suIntervalo;
  int min, max, brinco;
  public Gen(int min, int max) {
    try {
      this.min = min;
      this.max = max;
      suIntervalo = max - min;
      brinco = Math.round(suIntervalo/20);
      if (brinco == 0) brinco = 1;
    } catch (RuntimeException e) {
      e.printStackTrace();
    }
  }
}
```

B.4 Individuo.java

Este código define un Individuo que tiene su CodigoGenetico y sabe que tan exitoso es, entre los demas Individuos

```
import java.util.Random;

/*
 * Created on 22-jul-2004
 */
public class Individuo {
  CodigoGenetico adn = new CodigoGenetico();
  int[] ADN;
  public int lugar;
  public double indice;
  public Individuo() { //individuo en blanco
    ADN = new int[adn.genSize];
  }
  public Individuo(int[] estosCromosomas) {  //individuo con un adn
                              // determinado por el array de enteros
                              //que recibe el constructor
    //ADN = new int[7];
```

```
    ADN = estosCromosomas;
  }
  public Individuo(int tantito) {// individuo generado
                                  //random + tantito

    ADN = new int[adn.numDeGenes];
    Random escupe = new Random();
    CodigoGenetico genes = new CodigoGenetico();
    Gen gen;
    for (int i = 0; i < ADN.length; i++){
      gen = genes.getGen(i);
      ADN[i] = escupe.nextInt(gen.suIntervalo) + gen.min + tantito;
    }
  }
}
```

B.5 ListaDeIndividuos.java

Este código crea un ArrayList de Individuos

```
import java.util.ArrayList;
/*
 * Created on 24-jul-2004
 *
 * @author Fidel Serrano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class ListaDeIndividuos extends ArrayList {
  public Individuo getI(int arg0) {
    return (Individuo)super.get(arg0);
  }
  public void setADN(int individuo,int gen, int asi) {
    Individuo esteGuey = getI(individuo);
    esteGuey.ADN[gen]= asi;
    super.remove(individuo);
    super.add(individuo, esteGuey);
  }
}
```

B.6 Ventana.java

Este código se encarga de la operacion y propiedades de las ventanas que despliegan los resultados

```
import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Point;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Vector;
import javax.swing.JFileChooser;

/*
 * Created on 31-jul-2004
 *
 * @author Fidel Serrano
 *
 * Window - Preferences - Java - Code Style - Code Templates
 */

public class Ventana extends Frame implements ActionListener{
  TextArea yoDigo;
  boolean hayDisplay= false;
  Process guardando = null;
  Grafica graf;
  Vector datos;
  public Ventana(String pal_Boton, String titulo) {
    super();
    graf = new Grafica();
    datos = new Vector();
    setTitle(titulo);
    setSize(400,400);
    setResizable(false);
    yoDigo = new TextArea("", 20, 40,
```

```
                        TextArea.SCROLLBARS_VERTICAL_ONLY );
  yoDigo.setFont(new Font("u", Font.BOLD,13));
  Button salvar = new Button(pal_Boton);
  Button graficar = new Button("Ver Grafica");
  salvar.addActionListener(this);
  ActionListener ver = new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            graf.pinta(datos);
            graf.show();
            graf.setVisible(true);
          }
      };
  graficar.addActionListener(ver);
  setLayout(new BorderLayout());
  add(graficar, BorderLayout.NORTH);
  add(yoDigo, BorderLayout.CENTER);
  add(salvar, BorderLayout.SOUTH);
  setLocation(new Point(606,333));
}
public void escribe(String esteRenglon){
  yoDigo.append(esteRenglon);
}
public void escribe(Individuo esteGuey) {
  Double d = new Double(esteGuey.indice);
  datos.addElement(d);
  if (graf.isVisible()){
    graf.pinta(datos);
  }
  int[] ADN = esteGuey.ADN;
  for (int i = 0; i < ADN.length; i++){
    escribe(ADN[i] + "   ");
  }
  escribe(esteGuey.indice + "\n");
}
public void actionPerformed(ActionEvent e) {
  JFileChooser aqui = new JFileChooser();
  aqui.showSaveDialog(this);
  BufferedWriter pon = null;
  try {
    pon=new BufferedWriter(new FileWriter(aqui.getSelectedFile()));
    pon.write( this.yoDigo.getText() );
    pon.close();
  } catch (IOException e1) {
```

```
      System.out.println("NO PUDE GUARDAR");
      e1.printStackTrace();
    }
  }
}
```

B.7 ComparaIndividuos.java

Esta es una clase comparadora de Individuos

```
import java.util.Comparator;
/*
 * Created on 22-jul-2004
 *
 * @author Fidel Serrano
 *
 */
public class ComparaIndividuos implements Comparator {
  public int compare(Object arg0, Object arg1) {
    Individuo i0 = (Individuo)arg0;
    Individuo i1 = (Individuo)arg1;
    if (i0.indice < i1.indice)return -1;
    else if (i0.indice == i1.indice)return 0;
    else return 1;
  }
}
```

B.8 Grafica.java

Este código se encarga de desplegar los resultados en gráficas dinámicas

```
import java.awt.*;
import java.awt.event.*;
import java.util.Enumeration;
import java.util.Vector;

import JSci.awt.*;
```

```
import JSci.swing.*;

/**
* Sample demonstrating use of the Swing/AWT graph components.
* @author Mark Hale
* @modified by Fidel Serrano
* @version 1.1
*/
public class Grafica extends Frame {
  private DefaultGraph2DModel datos;
  Panel graph3;
  Label title;
  JLineGraph esta;
  public static void main(String arg[]) {
    new Grafica();
  }
  public Grafica() {
    super("Una Grafica");
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent evt) {
        dispose();
        //System.exit(0);
      }
    });
    setLayout(new BorderLayout());
    setSize(350,250);
    Font titleFont=new Font("Default",Font.BOLD,14);
    // datos
    datos=new DefaultGraph2DModel();
    double[] unos = {0.1,0.0};
    datos.setXAxis(0.0f,2.0f,33);
    datos.addSeries(unos);
    datos.setSeriesVisible(0,true);
    // value graphs
    //datos=createValueData();
    // line graph
    graph3=new Panel();
    graph3.setLayout(new JGraphLayout());
    title=new Label("Que Tanto Falla",Label.CENTER);
    title.setFont(titleFont);
    graph3.add(title,"Titulo");
    esta=new JLineGraph(datos);
    graph3.add(esta,"Graph");
```

```
    // layout
    add(graph3,"Center");
    //setVisible(true);
  }
  public void pinta(double[] losValores){
    DefaultGraph2DModel otros = new DefaultGraph2DModel();
    try {
      otros.setXAxis((float)1.0,(float)losValores.length,
                                                losValores.length);
      otros.addSeries(losValores);
      otros.setSeriesVisible(0,true);
      graph3.remove(esta);
      esta=new JLineGraph(otros);
      esta.setYExtrema((float)0.0,(float)0.15);
      int dx = (int)((float)losValores.length/(float)5.0);
      esta.setXIncrement((float)dx);
      esta.setYIncrement((float).02);
      graph3.add(esta,"Graph");
      setVisible(true);
      esta.redraw();
    } catch (RuntimeException e) {
      e.printStackTrace();
      System.out.println("uuuu");
    }
    System.out.println("no hay faya");
  }
  public void pinta(Vector datos){
    double[] array = new double[ datos.size() ];
    int count = 0;
    Enumeration e = datos.elements();
    while( e.hasMoreElements() ){
      Double wrapper = (Double) e.nextElement();
      array[ count ] = wrapper.doubleValue();
      count += 1;
    }
    pinta(array);
  }
}
```

Bibliografía

[1] P. Allen. *Cities and Regions as Self-Organizing Systems: Models of Complexity (Environmental Problems Social Dynamics).* Gordon Breach Science Publishers, Amsterdam, 1997.

[2] G. Garza. *La Urbanización de México en el Siglo XX.* El Colegio de Mexico, Mexico, 2003.

[3] M. Batty, Y. Xie, and Z. Sun. The dynamics of urban sprawl. *Working Paper Series, Centre for Advanced Spatial Analysis, University College London*, 15, 1999.

[4] C. G. Langton. *Life at the edge of chaos, Artificial Life II, Santa Fe Studies in the Sciences of Complexity, vol. X.* Addison-Wesley, Redwood City, California, 1992.

[5] F. Wu and C. Webster. Simulating artificial cities in a gis environment: urban growth under alternative regulation regimes. *Int. J. Geographical Information Science*, 14 (7):625–648, 2000.

[6] R. Conte, R. Hegselmann, and P. Terna. *Simulating Social Phenomena.* Springer, Heidelberg, 1997.

[7] F. Wu. Calibration of stochastic cellular automata: the application to rural-urban land conversions. *Int. J. Geographical Information Science*, 16 (8):795–818, 2000.

[8] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55:601–644, 1983.

[9] S. Wolfram. *A New Kind of Science.* Wolfram Media, Inc., Canada, 2002.

[10] J. Von Neumann. Theory of self-reproducing automata. *University of Illinois Press*, 1966.

[11] O. Martin, A. Odlyzko, and S. Wolfram. Algebraic properties of cellular automata. *Commun. Math. Phys.*, 93:219, 1984.

[12] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[13] F. Wu. Complexity and urban simulation: towards a computational laboratory.

[14] R. White and G Engelen. Cellular automata and fractal urban form: a cellular modeling approach to the evolution of urban land use patterns. *Environmental Planning A*, 25:1175–1199, 1993.

[15] D. Vanbergue, J-P. Treuil, and A. Drogoul. Modelling urban phenomena with cellular automata. *Advances in Complex Systems Journal*, 3:1–4, 2000.

[16] R. Mansilla and J. Gutierrez. Deterministic site exchange cellular automata models for the spread of diseases in human settlements. *Complex Systems*, 13, 2001.

[17] M. Phipps. Dynamical behavior of cellular automata under the constraint of neighborhood coherence. *Geographical Analysis*, 21(3):197–215, 1989.

[18] D. O´Sullivan and P. Torrens. Cellular models of urban systems. *CASA Fourth International Conference on Cellular Automata for Research and Industry, Karlsruhe Germany*, 2000.

[19] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming. An Introduction.* Kaufmann Publishers, San Francisco, 1998.

[20] J. Raper and R. Krzanowski. *Spatial Evolutionary Modeling.* Oxford University Press, Oxford, 2001.

[21] B.W. Luscombe and T.K. Poiker. Strabo: An alternative gis approach to decision making for planning applications in data scarce environments. *Proceedings of the Sixth International Symposium On Automated Cartography*, 1:264269, 1983.