Generador d'Escenaris Sintètics Particulars amb Tècniques Heurístiques per Algoritmes Oportunistes

Marçal Torres Codina

Resum— Aquest treball és un programa realitzat en Java que genera nodes i els moviments en el temps d'aquests en un escenari sintètic. L'objectiu del treball és donar valors d'entrada al programa, aquest valors són densitat, dispersió i centralitat basada en clústers, on apliquem la base d'algoritmes genètics per tal de millorar les característiques del escenari. La millora de les característiques de l'escenari s'aconsegueix alterant el recorregut dels nodes que el conformen. La primera alteració és trobar el punt mig de la posició de dos nodes en tots els temps i així es crea un nou recorregut per al node fill. La segona alteració és la del camí compartit, això significa que del node pare obtenim la meitat del recorregut que ha fet, per exemple, dos posicions a la dreta, tres posicions a baix... fins la meitat del recorregut i fem el mateix amb el node mare i així el fill té la meitat del recorregut del pare i l'altre meitat del recorregut de la mare. Es realitzen aquestes funcions amb tots els nodes de l'escenari i es tornen a comprovar les característiques de l'escenari. Aquests mètodes els anomenem generacions, realitzem tantes generacions com siguin necessàries, el programa et retorna un document de text amb els nodes per TheOne Simulator.

Paraules clau—Java, algoritmes oportunistes, xarxes oportunistes, programa generador d'escenaris, algoritmes genètics, tècniques heurístiques, projecte java, TheOne Simulator.

Abstract—This work is a program made in Java that generates nodes and their moviments in time in a synthetic scenario. The aim of the work is to give input values to the program, these values are density, dispersion and centrality based on clusters, where we apply the basis of genetic algorithm in order to improve the characteristics of the scenario. Improving the characteristics of the scenario is achived by altering the path of the nodes that make it up. The first alteration is to find the midpoint of the position of two nodes at all times and thus a new path is created for the child node. The second alteration is that of the shared path, this means that from the parent node we get half of the path that has made, for exemple, two positions on the right, three positions down... to the middle of the path and we do the same with the mother node and so the child has half of the father's path and the other half of the mother's path. These functions are performed with all nodes in the scenario and the characteristics of the scenario are checked again. We call these methods generations, we perform as many generations as necessary, the program retorns you a text document with the nodes by TheOne Simulator.

Index Terms—Java, opportunistic algorithm, opportunistic network, scenario generator program, genetic algorithm, heuristic technique, java project, TheOne Simulator.

1 Introducció

AQUEST treball té com objectiu crear un programa generador d'un arxiu de text pel TheOne Simulator[5], aquest arxiu es compon de nodes que pertanyen a un escenari sintètic. L'arxiu serveix per provar algoritmes oportunistes en xarxes oportunistes, ja que no tots els algoritmes obtenen els mateixos resultats en diferents escenaris. Aquest programa creat en Eclipse Java[1][2] té l'aplicació d'obtenir valors d'entrada com la quantitat de nodes, el temps del escenari i les característiques de l'escenari. A partir d'aquests valors generar els nodes i els seus recorreguts per tal de complir els requisits especificats.

- *E-mail de contacte: Marcal.Torres@e-campus.uab.cat*
- *Menció realitzada: Tecnologies de la Informació.*
- *Treball tutoritzat per: Diego Freire*
- *Curs 2019/20*

La primera creació és totalment aleatòria, per tal d'aconseguir els valors desitjats comprovarem les característiques de l'escenari, si no passen els filtres introduïts anteriorment passem a fer les generacions. Cada generació modifica els recorreguts dels nodes per veure si milloren o no les característiques. La idea de cada generació es basa en algoritmes genètics[4].

La metodologia del treball ha sigut de prototipatge, des de la simple creació de l'objecte node, a la creació de l'objecte escena que conté n objectes Node, escenaris que es conformen d'escenes i fins a les generacions, es va fent prova i error, amb uns objectius a curt termini per anar comprovant les funcionalitats de casa pas donat.

Aquest document comença explicant cada objecte i els mètodes que el conformen, característiques dels escenaris, continua amb l'explicació de les dues formes de creuar els nodes per crear generacions, explicació i utilització d'algoritmes genètics i per acabar cada resultat del prototipatge fins el resultat final i les conclusions del treball.

2 OBJECTIUS

2.1 Generals

L'objectiu principal del treball és crear un programa generador d'escenaris des de zero amb el llenguatge Java.

Per tal de que es puguin provar diferents algoritmes oportunistes calen diferents tipus d'escenaris ja que els algoritmes no donen els mateixos resultats en tots els escenaris. En aquest treball l'objectiu és crear escenaris sintètics, donant uns valors d'entrada que determinaran com serà l'escenari i retornant un fitxer de text amb l'escenari desitjat.

2.2 Específics

El programa es crea amb el llenguatge Java ja que aquest llenguatge esta orientat a objectes i per aquest treball va perfecte. Tenim Nodes, Escenes i Escenaris que representats com objectes són fàcils de donar característiques i manipular-los.

Un altre objectiu és estudiar i aplicar a la mesura del possible els algoritmes genètics. Aquests algoritmes estan orientats a la millora de la població i del individu, és un dels objectius que tenim, generar un escenari amb les característiques desitjades.

3 METODOLOGIA

Els primers passos en aquest treball són la investigació d'escenaris per xarxes oportunistes i la funcionalitat dels algoritmes genètics.

A continuació comença el disseny de prototips, anar posant petits objectius de programació per avançar pas per pas. Al començament serà crear nodes, escenes i escenari i poder calcular les seves característiques.

Un cop tenim escenaris ja ens podem centrar en els algoritmes genètics i com implementar-los en el codi i en les necessitats del programa. Amb els algoritmes genètics implementats continuem amb el prototipatge i ens tornem a posar petits objectius en les millores dels escenaris.

4 ESTAT DE L'ART

4.1 Resum

La disponibilitat i el rendiment de les actuals tecnologies sense fils com el Wifi, el 4G o ara el 5G poden donar problemes importants de congestió i propagació[6], especialment en espais amb molta activitat o pel contrari amb poca densitat de població.

L'ús de xarxes oportunistes en aquests tipus d'escenaris poden ser la solució als problemes. Aquestes xarxes es basen en l'oportunitat d'intercanviar missatges utilitzant alguna tecnologia de comunicació directa entre dispositius mòbils com el Bluetooth o el WiFi.

El rendiment de les xarxes oportunistes depenen principalment en la mobilitat dels nodes i els protocols d'encaminament utilitzats. Són aquests últims els encarregats de decidir com son intercanviats els missatges quan es produeix un contacte, intentant trobar la millor ruta per arribar al seu destí.

Per altre part l'eficàcia de la difusió depèn especialment de la mobilitat dels usuaris i al comportament humà. Llavors per l'anàlisi i l'avaluació del rendiment de les xarxes oportunistes és necessari considerar tant els aspectes tècnics relacionats amb els protocols d'encaminament com els aspectes de la mobilitat humana.

4.2 Definició

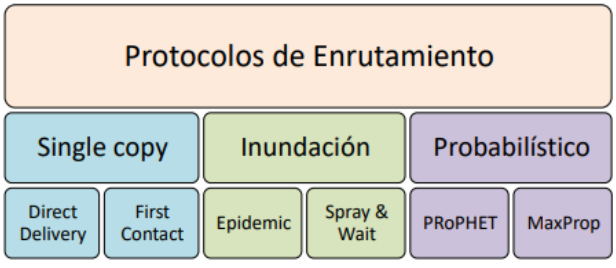
Les xarxes oportunistes en l'oportunitat d'establir un contacte entre parells de nodes per propagar missatges. L'efectivitat depèn principalment amb el número de contactes i la duració d'aquests, per tant la mobilitat.

Els principals factors en una xarxa oportunista són la velocitat, el model de mobilitat, la densitat dels nodes i els llocs per on es mouen. Així ho expliquen autors que estudien i avaluen la difusió de missatges utilitzant diferents algoritmes oportunistes.[8]

4.3 Protocols d'encaminament

Els protocols d'encaminament o algoritmes oportunistes són utilitzats en xarxes oportunistes per realitzar el procés d'intercanvi d'informació quan els nodes estan en contacte.

Poden ser classificats depenent de diferents aspectes, com l'ús o no d'infraestructura, aplicacions, quantitat d'informació que poden manipular i altres. (figura 1)



Protocols segons el número de missatges
Figura 1

Single Copy

Els protocols de "Copia Simple" són els que mantenen una sola copia del missatge a la xarxa fins que és entregat al seu destí.

- *First Contact:* quan els nodes estan utilitzant l'algoritme de primer contacte transfereixen el missatge al primer node que es trobin, repetint el procés fins que arribi al node destí.[9]

- *Direct Delivery:* aquest protocol d'entrega directe es pot considerar com la forma més simple de transmetre un missatge. El missatge només es transmetrà al node amb qui faci contacte si aquest és el node destí. Per tant la possibilitat que arribi al destí depèn del contacte directe entre el node origen i el node destí.[10]

Utilitzant aquest tipus de protocols només un missatge existeix a la xarxa, per tant la sobrecarrega de la xarxa es redueix al mínim, el problema es que la probabilitat d'entrega és baixa i el temps molt alt.

Inundació
Els protocols d'inundació propaguen un missatge per la xarxa. Pot ser tant a nivell de no control com amb n número de còpies.

- *Epidemic:* el protocol epidèmic realitza una copia del missatge per a tots els nodes que contacta, incrementant la possibilitat d'entrega. El principal problema és la sobrecàrrega, això demana molt ample de banda i memòria als nodes.[11]

- *Spray & Wait:* és un protocol de disseminació controlada, per tant es redueix la sobrecàrrega a la xarxa si el comparem amb *l'Epidemic*. Amb aquest algoritme els nodes d'origen assignen un nombre màxim de còpies del missatge que es poden crear. La primera fase és "Spray" on les còpies del missatge són distribuïdes als contactes, la segona fase és "Wait", com diu la paraula consisteix en esperar a que alguna còpia arribi al destí.[12]

Probabilístics
Aquests tipus de protocols es basen en la transmissió de les còpies del missatge als nodes que tenen més possibilitats de trobar-se amb el destinatari.

- *PRoPHET:* és el primer protocol probabilístic creat per xarxes oportunistes. Aquest algoritme es basa en determinar quins nodes són els millors per reenviar el missatge i arribi al destinatari. Utilitzant aquests nodes reduïm la quantitat de còpies que hi ha a la xarxa i augmentat la probabilitat de que el missatge arribi al destí.
  Concretament aquest protocol estima la probabilitat de que el node contactat pugui entregar el missatge basant-se en el historial de contactes, reenviant el missatge només als nodes que tinguin alta probabilitat d'entrega.[13]

- *MaxProp:* aquest protocol fa difusió del missatge sobre la xarxa, fent una validació de totes les rutes possibles per a cada missatge, guarda el número de salts i la probabilitat que el missatge arribi al destí basada en contactes anteriors.
  Si el missatge arriba al destí només es té en compte aquesta ruta i s'envia un missatge de resposta perquè totes les còpies errònies s'eliminin.[14]

5 ALGORITMES GENÈTICS

5.1 Historia i definició

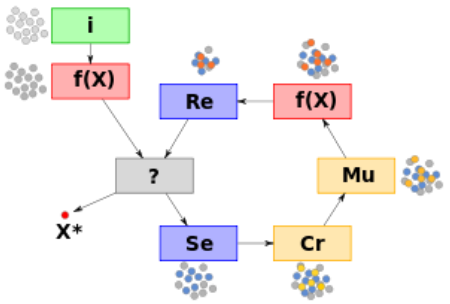
Als anys 1970 de part de John Henry Holland va sorgir una branca per a la intel·ligència artificial, els algoritmes genètics[7], s'anomenen així perquè s'inspiren en l'evolució biològica i genètica molecular.

Aquests algoritmes fan evolucionar la població mitjançant mutacions, recombinacions genètiques i la selecció segons algun criteri, aquest criteri determina quins individus són els més aptes, que sobreviuen, i quins no, que són descartats.

5.2 Informàtica

Els algoritmes genètics s'implementen normalment com una simulació informàtica (figura 2), en la qual a una població li apliquem una sèrie de solucions representada com una cadena, generalment binaria, anomenada cromosoma. Quan la representació del cromosoma es fa amb cadenes de dígits es coneix com a genotip.

Els individus evolucionen a través d'iteracions, anomenades generacions. A cada generació els individus són avaluats utilitzant una mesura d'aptitud. Per cada generació hi ha unes regles anomenades operadors genètics, aquests operadors són: selecció, creuament, mutació i reemplaçament.



i: inicialització; f(x): avaluació; ?: condició d'avaluació; Se: selecció; Cr: creuament; Mu: mutació; Re: reemplaçament; X: millor individu;*
Figura 2

5.3 Quan s'utilitzen

Els algoritmes genètics tenen una eficàcia provada, es poden utilitzar pràcticament amb qualsevol funció.

S'han de tenir en compte algunes consideracions; si la funció a optimitzar té molts màxims i mínims caldran més generacions per aconseguir un resultat òptim; si la funció té molts punts propers al òptim només es pot assegurar de trobar-ne un, i potser no ser el més òptim de tots.

5.4 Funcionament

Qualsevol algoritme genètic pot presentar variacions, ja que depenent de com s'apliquen els operadors genètics, com són el creuament i la mutació, de com es realitza la selecció i de com es decideix com substituir els individus menys aptes per la nova població.

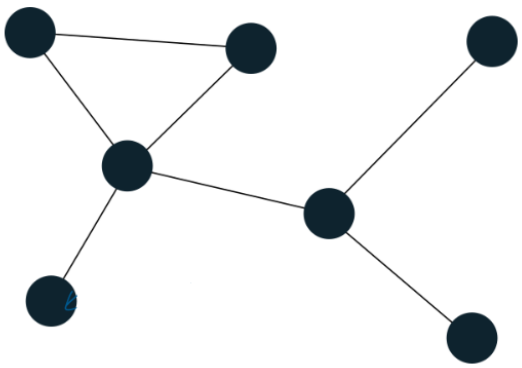
El primer pas és la inicialització, es genera aleatòriament la població inicial, si no es fa aleatòriament hem d'assegurar que hi ha diversitat estructural.
Seguidament ve l'avaluació, a cada un dels individus de la població se'ls aplica una funció d'aptitud per saber com de bo és i com de bona es la seva possible genètica.

A continuació s'apliquen els operadors genètics, selecció dels millors individus per creuar i formar la següent generació, com millor aptitud més possibilitats de ser seleccionats. Creuament, agafant dos individus com si fos la reproducció sexual, s'opera sobre aquest individus per crear un o dos individus fills de la combinació d'aquests pares. La mutació és modificar al atzar una part de les característiques d'un individu per arribar a zones de cerca que no estaven presents en la població actual. Finalment la substitució o reemplaçament, un cop s'apliquen els anteriors operadors genètics es seleccionen els millors individus per crear la següent generació.

6 Objectes principals

6.1 Node

En aquest programa l'objecte Node és la base de tot el treball, es compon de quatre variables, Id, posició X, posició Y i el temps. Per cada creació d'un objecte Node li donem una Id, les posicions X i Y i en quin temps del escenari es crea.
Quan avancem en el temps obtenim els valors del node amb la Id que toca, modifiquem els valors i creem un nou node amb la mateixa Id, posicions modificades per un RandomWalk i amb el temps avançat a la següent iteració. (figura 3)



Exemple Nodes
Figura 3

6.2 Escena

L'objecte escena és on es generen tots els objectes Node que toquen, el mètode per fer-ho és **public void generarEscena(ArrayList<Node> nodes)**. Aquest mètode genera n nodes aleatòriament quan el time = 0.0, quan el temps és time = 0.1 fins time = t el mètode utilitza un RandomWalk per moure els nodes creats anteriorment.

A part de la creació l'objecte també té tres mètodes per calcular la densitat (**public void calculateDensity(int[][] matrix)**), la dispersió (**public void calculateSparsity(int[][] matrix))** i la centralitat basada en clústers (**public void calculateCentrality(int[][] matrix))**.

Aquests càlculs es fan per a cada escena, s'agafa el valor més alt que ens doni fent el càlcul per quadrant. Després es fa el càlcul a l'objecte escenari per saber les característiques del escenari que seran les que han de passar els filtres.

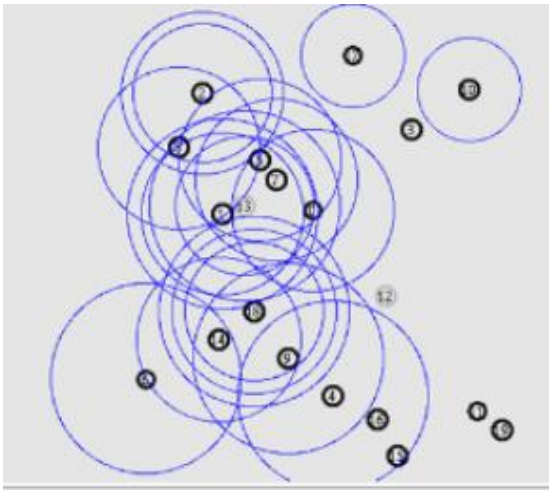
En aquest objecte també guardem el número de quadrants i **blocs = quadrants * quadrants**, aquestes variables ajuden a fer càlculs als mètodes.

6.3 Escenari

Aquest objecte junta totes les escenes amb el mètode **public ArrayList<Node> generadorEscenari(ArrayList<Escena> escenes, ArrayList<Node> nodes)**, una escena per cada temps, i fa els càlculs de les característiques. El càlcul de les característiques són la suma de cada una de les característiques en qüestió dividides per la quantitat d'escenes que tenim, una mitja.

Aquest objecte té dos mètodes més, un és el que comprova les característiques amb els valors d'entrada **public boolean sceneFilter(double densityFilter, double sparsityFilter)** que retorna un valor booleà i per últim el mètode que ens crea el fitxer de text que donarà la sortida del programa **public void crearFitxer(ArrayList<Escena> escenes, ArrayList<Node> nodes).** [15] (figura 4)

Exemple escenari



Exemple escenari amb nodes comunicant-se
Figura 4

6.4 Generació

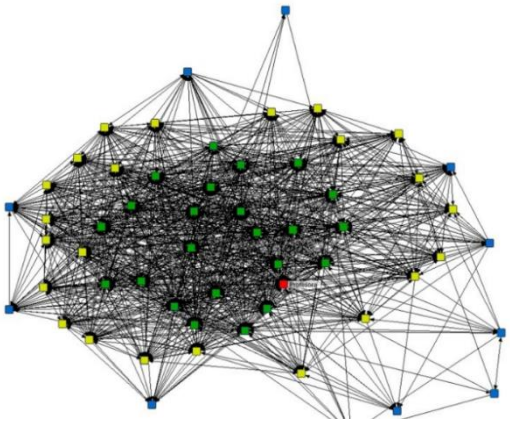
Per últim, a part de l'objecte Program que és el main del programa i només crida a altres objectes i mètodes, tenim l'objecte que crea generacions. Aquest objecte conté dos mètodes que tenen la funció de ajuntar els nodes per crear fills i reunir una nova generació.

El primer mètode és **public static void fiftyFifty(ArrayList<Node> nodes)** i el segon mètode és **public static void halfWay(ArrayList<Node> nodes, ArrayList<Escena> escenes, Escenario scenario, double time),** més endavant explicarem quines funcionalitats tenen.

7 CARACTERÍSTIQUES

7.1 Densitat

La densitat, com ja s'ha parlat abans, és una de les tres característiques que ens limiten el filtratge dels escenaris.
El càlcul de la densitat és realitza per a cada escena i per l'escenari resultant. (figura 5 i 6)



Exemple densitat
Figura 5

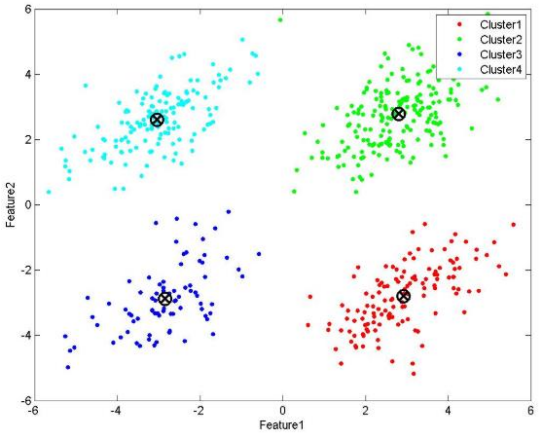
Cada escena es divideix el mapa en quadrants, i es realitza el càlcul de cada una de les àrees, un cop calculades ens quedem amb la millor. La densitat del escenari és el valor mitjà dels millors quadrants de cada escena.
La densitat és una porció entre els individus que hi ha en una àrea i la quantitat total d'individus. Representa les connexions actuals entre les possibles de tot el mapa.

$$D = \frac{Nº\ Individus}{Superfície}$$

Fórmula densitat
Figura 6

7.2 Centralitat

La centralitat basada en clústers mesura el grau de connectivitat en una xarxa. La quantitat de nodes que estan dins dels límits per comunicar-se entre si. (figura 7)



Centralitat de clústers
Figura 7

Es calcula per a cada quadrant com la densitat sense tenir en compte si un node veí d'un altre quadrant pot estar dins dels límits per la comunicació. Podem modificar el càlcul per la quantitat clústers per escena, la distància entre clústers o el mida dels clústers.

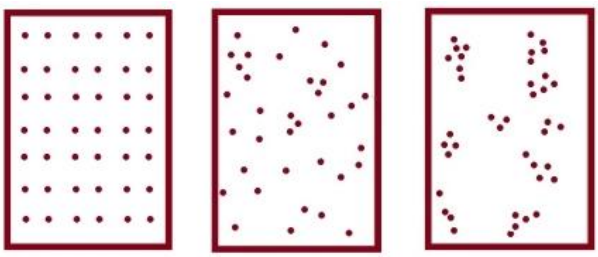
7.3 Dispersió

La dispersió mostra la distribució física dels nodes sobre l'àrea de l'escena, tenint en compte quantes àrees tenen baixa presencia de nodes, que és un 20% menys del esperat. La fórmula del càlcul és la següent. (figura 8)

$$Sparsity = \frac{sum(low\_presence\ blocks)}{Blocks}$$

Fórmula dispersió
Figura 8

Com totes les característiques es calcula per a cada quadrant i es divideix quants quadrants tenen baixa presencia entre els quadrants totals que dona un percentatge de dispersió de l'escena. (figura 9)

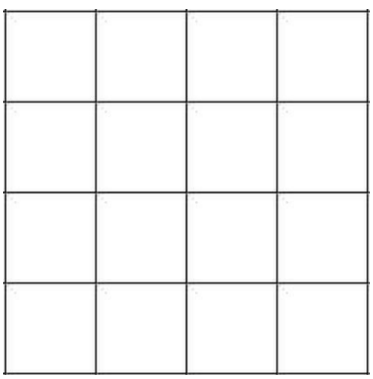


Diferents tipus de dispersió
Figura 9

7.4 Quadrant

El quadrant és una característica que comparteixen totes les escenes i per tant l'escenari, es un valor int de 1 a 6 que ens determina en quina quantitat de blocs dividirem el nostre escenari sintètic. El valor donat el multipliquem per ell mateix per saber quants blocs tindrem, per tant el valor 2 ens farà dividir entre 4 parts iguals l'escenari, el 5 ens el divideix en 25 parts, i així amb tots els valors. (figura 10)

Aquesta característica no varia entre generacions ni escenes, és una característica constant per cada escenari.



Quadrants = 4; Blocs = 16
Figura 10

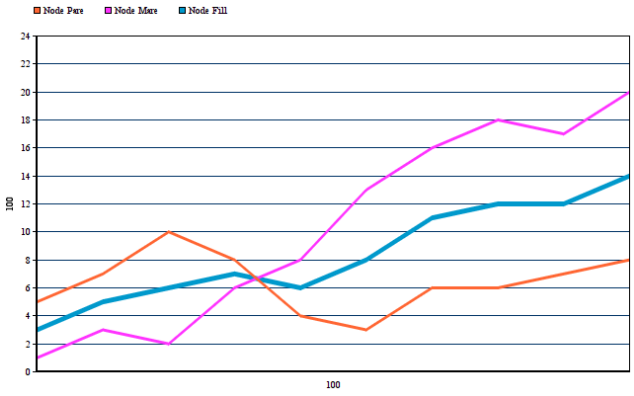
8 FÓRMULES GENERACIONS

8.1 Punt Mig

Com ja sabem els Nodes tenen posició X i posició Y, amb aquesta funcionalitat agafem un escenari amb bones característiques i calculem el punt mig per a cada dos Nodes en cada instant de temps.

Per exemple (figura 11):
Node1: x = 4, y = 60 i t = 1 i Node2: x = 8, y = 30 i t = 1
El resultat serà NodeFill: x = 6, y = 45 i t = 1

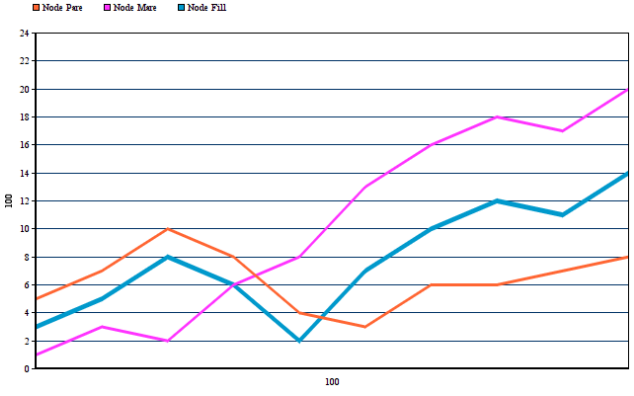
Es calcula per a tots els nodes, per tal d'arribar al número de nodes que volem, podem ajuntar diferents nodes fins al màxim o bé crear-ne alguns de nous.
Tornem a comprovar les característiques per veure si ha millorat el nou escenari en algun aspecte.



Node Pare; Node Mare; Node Fill
Figura 11

8.2 FiftyFifty

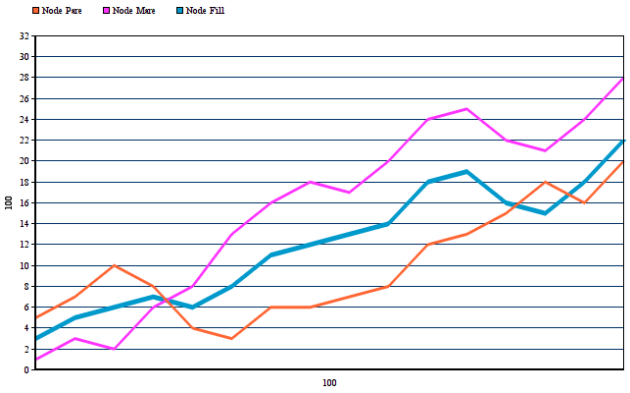
En aquesta funcionalitat tenim en compte el recorregut que ha fet cada node, per al NodeFill tenim en compte el principi del recorregut de la NodeMare fins la meitat i agafem de la meitat fins al final del recorregut del NodePare. També arribem fins als nodes màxims o afegim nous per arribar al màxim. (figura 12)
Com en el mètode anterior comprovarem les característiques per veure si l'escenari nou ha obtingut millores en les característiques.



Node Pare; Node Mare; Node Fill
Figura 12

8.3 Mixt

Com diu la paraula és agafar un percentatge del mètode de Punt Mig i la resta del percentatge de Meitat i Meitat.
Si fem 50-50% fem el 50% del recorregut amb el mètode de Punt Mig i el 50% el realitzem amb la funció de Meitat i Meitat. (figura 13)



Node Pare; Node Mare; Node Fill
Figura 13

8.4 Mutacions

En tots els casos podem afegir la mutació, això significa que podem modificar alguna part del recorregut a voluntat.

El mètode obté el llistat de nodes i aleatòriament decideix quants nodes de la llista seran modificats. Seguidament al atzar obté un 10, 20 o 30% de modificació. Apliquem als nodes que han sigut escollits el percentatge de modificació per a les seves posicions X i Y, també aleatòriament decidim si serà augment o disminució.

Un cop aplicat es torna a calcular les característiques del escenari i si encara no es prou bo es torna a aplicar els mètodes anteriors.

9 FUNCIONALITAT DEL PROGRAMA

9.1 Recorregut del programa

El recorregut del programa comença a la classe Program() que és l'equivalència a la funció main().

Al iniciar pregunta pel temps total del escenari, el número de nodes que hi haurà al escenari i a cada escena, el número de quadrants (recordem que el valor del quadrant es multiplica per ell mateix per crear n blocs, p.e. 3x3, 5x5), el mida X i el mida Y del nostre mapa.

A continuació un bucle de 0 fins al $temps_{max}$ crida a la funció del objecte Escenari.generadorEscenari, aquest retorna la llista de nodes de cada una de les escenes.
Dins el mètode generadorEscenari es crea un nou objecte Escena() per cridar al mètode Escena.generarEscena, on també es retorna una llista de nodes però en aquest cas de la escena en qüestió.

El mètode generarEscena és un dels més importants, en aquest mètode si el temps és 0 es creen N nodes (valor obtingut per entrada) amb posicions aleatòries, a partir del temps 1 fins al $temps_{max}$ tenim implementat un RandomWalk on obtenim les posicions de cada node en el temps anterior i aleatòriament hi ha un moviment de -3 fins a 3 per a la posició X i per a la posició Y, això significa una llibertat de direcció total.

Per a cada iteració de temps es genera una escena i, per tant, es realitzen els càlculs de les característiques.
Un cop acabat el bucle de temps ja tenim totes les escenes amb els seus valors de les característiques i el llistat de nodes en cada temps i posicions.

En aquest punt el programa calcula les característiques del escenari i les mostra, un cop fet els càlculs pregunta quins valors mínims volem per a cada característica, si hi ha algun valor del escenari més baix que el desitjat entrem a la part de generacions.

Dins un while amb una condició booleana que les característiques siguin superiors o iguals a les desitjades tenim els dos mètodes per crear generacions, halfWay i fiftyFifty.
Podem triar quin utilitzar, després d'algunes generacions sense bons resultats podem utilitzar l'altre o bé continuar esperant millors resultats.

9.2 Generacions

Si entrem a la funció halfWay de l'objecte Generation() la funcionalitat que trobarem serà la de trobar el valor mitjà. Per cada dos nodes calculem el valor mitjà entre les posicions X i les posicions Y en cada instant de temps, aquests valors mitjans els hi donem a un nou node que serà el node fill.

Això es fa per a cada node i cada instant de temps, on hem de calcular les característiques noves de cada escena, fins que la llista de nodes esta complerta amb nous nodes i calcular finalment les característiques del nou escenari, retornem la llista i l'escenari tornarà a ser avaluat.

En el cas que entrem al mètode fiftyFifty la funcionalitat és molt diferent. Per a cada node calculem els moviments que ha fet, això vol dir que si en el temps 0 estava a la posició (10, 45) i en el temps 1 a la posició (8, 48) obtenim que el seu recorregut entre el temps 0 i 1 ha sigut de (-2, 3), aquest valor del recorregut el guardem en una llista i obtenim tots els recorreguts de tots els nodes.

Després es crea una generació nova en el temps 0 amb valors de posició aleatoris inicialment. Seguidament per a cada node li imposem el recorregut que ha de realitzar, la meitat dels moviments del node pare i la meitat dels moviments del node mare, amb el resultat que la meitat del camí del nou node és la meitat de cada un dels recorreguts.

Quan tenim la nova llista amb la nova generació fem els càlculs per a cada escena i finalment per a l'escenari. Retornem la nova llista de nodes.

Els anteriors mètodes es poden repetir tantes generacions com siguin necessàries, però si veiem moltes generacions sense millora podem aplicar el mètode mutationNode().

Aquest mètode obté la llista de nodes de l'última generació i la modifica, aleatòriament augmenta o disminueix les posicions en algun moment de temps d'alguns nodes. Això serveix per intentar que les generacions no entrin en un bucle d'empitjorament.

Finalment sortim del while, podem posar un límit de generacions per sortir del programa, ja que hem aconseguit una bona generació o bé ens hem rendit. En tot cas l'última generació es passa al mètode crearFitxer i generem un arxiu de text amb el llistat de nodes per poder integrar-lo al TheOneSimulator.

10 PROTOTIPATGE I RESULTATS

10.1 Nodes temps 0

El primer objectiu del projecte era crear una escena, per tant un llistat de N nodes en el temps 0 i aquest és el resultat. (figura 14)

```
Node [id=0 positionX=41, positionY=4, time=0.0]

Node [id=1 positionX=24, positionY=46, time=0.0]

Node [id=2 positionX=75, positionY=14, time=0.0]

Node [id=3 positionX=2, positionY=18, time=0.0]

Node [id=4 positionX=10, positionY=85, time=0.0]
```

Primer prototip
Figura 14

Amb aquests resultats el següent pas és crear per a cada temps, això vol dir que per a cada escena i per tant ja tenim la creació del primer escenari.

10.2 Càlculs característiques

Seguidament crear els mètodes dels càlculs per a cada escena i els càlculs del escenari, ja que amb el resultat anterior ja podíem crear escenes per a cada temps i per tant l'escenari. (figura 15)

```
Escena [numNodes 50/ density 0.007346938775510204/ sparsity 66.66
Escena [numNodes 50/ density 0.007346938775510204/ sparsity 77.77
Escena [numNodes 50/ density 0.006530612244897959/ sparsity 66.66
Escena [numNodes 50/ density 0.006530612244897959/ sparsity 88.88
Escena [numNodes 50/ density 0.006530612244897959/ sparsity 77.77
Escena [numNodes 50/ density 0.006530612244897959/ sparsity 66.66
Escena [numNodes 50/ density 0.007346938775510204/ sparsity 55.55
Escena [numNodes 50/ density 0.006530612244897959/ sparsity 77.77
Escena [numNodes 50/ density 0.007346938775510204/ sparsity 55.55
Escena [numNodes 50/ density 0.007346938775510204/ sparsity 55.55
Escena [numNodes 50/ density 0.00816326530612245/ sparsity 66.666
```

Segon prototip
Figura 15

10.3 Generacions

A continuació el següent repte i objectiu era crear els mètodes de generacions, un per un. Aquests mètodes estan formats per molts bucles i moltes condicions per a que no sorgeixin errors.

Finalment al crear les mètodes falta saber si milloren les característiques i arriben als límits desitjats. (figura 16)

```
Escenario [sparsity=77.20588235294117 nodes=200 time=5.0]Primer escenario
Dispersio minima desitjada?
87
Escenario [sparsity=90.19607843137256 nodes=200 time=5.0]Mejor Generación
Se han hecho 3 generaciones
```

Prototip final
Figura 16

Com podem veure amb només tres generacions el programa ha obtingut fins i tot un 3% més de dispersió que la dispersió mínima desitjada.

10.4 Mutació

Quan veiem que les generacions no estan donant els resultats desitjats apliquem a una generació mutació.
En aquest cas quan portàvem 50 generacions apliquem el mètode i en només 7 generacions més aconseguim els valors desitjats. (figura 17)

```
Escenario [sparsity=81.55940594059406 nodes=300 time=10.0]Primer escenario
Dispersio minima desitjada?
98
Escenario [sparsity=81.25 nodes=300 time=10.0]Con mutación
Escenario [sparsity=99.62871287128714 nodes=300 time=10.0]Mejor Generación
Se han hecho 57 generaciones
```

Resultat final amb tots els protocols aplicats
Figura 17

11 Conclusió

Podem concloure que s'han aconseguit tots els objectius principals i secundaris que havíem definit. El programa està programat de forma modular per canviar quantitat de nodes, mida del escenari i quadrants, podem crear molts escenaris diferents.

Més específicament la funcionalitat dels algoritmes genètics aplicats al programa han donat bons resultats la majoria de cops, si demanem uns valors massa alts no sempre troba els valors desitjats però sempre millora l'escenari creat inicialment.

La realització del treball ha estat una bona eina per profunditzar més en la programació en Java, per saber més sobre xarxes oportunistes i per descobrir i aprendre sobre els algoritmes genètics. Inicialment el tema a tractar era un tant desconegut per mi, només per una o dues assignatures durant la carrera, i tenia una mica de por per si no m'agradava. Al final ha sigut un molt bon aprenentatge tant de part de programació com la part d'investigació i els resultats obtinguts han sigut molt satisfactoris.

11.1 Problemes durant la realització del treball

Al principi del projecte havia de tornar a familiaritzar-me amb el llenguatge de Java, no va el problema més complicat però em va fer investigar bastant per realitzar el programa correctament.

Més enllà del llenguatge va haver algun problema a l'hora de modular tot el programa, les variables no depenien de mi i per tant els mètodes s'havien de comportar sempre de la mateixa manera per diferents valors, tot i així va ser satisfactori aconseguir que el programa estigui modulat.

Finalment l'últim repte ha sigut desenvolupar els mètodes dels algoritmes genètics. El mètode de punt mig no em va donar gaires problemes, però el mètode de fiftyFifty de recorreguts compartits si va ser un repte major. Molts bucles i molts condicionants en molts passos que dificultaven saber on estava l'error.

Tot i tenir aquests problemes he pogut seguir endavant i estic orgullós del resultat obtingut.

Agraïments

Agraeixo al meu tutor Diego Freire per estar sempre disponible per qualsevol dubte i per compartir coneixements sobre les xarxes oportunistes i els escenaris.

Agraïments també al meu amic Pablo Hervás que em va ajudar i dirigir a un estil de programació més net i a resoldre dubtes sobre el llenguatge i les funcionalitats.

Bibliografia

[1] A. Walton. Javadesdecero, [Online]. Disponible: https://javadesdecero.es/fundamentos/mejores-ide-para-programadores-java/
[2] campusMVP. CampusMVP, [Online]. Disponible: https://www.campusmvp.es/recursos/post/5-motivos-por-los-que-utilizar-java-para-desarrollar-tus-aplicaciones.aspx
[3] -. Tutorialesprogramacionya, [Online]. Disponible: https://www.tutorialesprogramacionya.com/javaya/
[4] F. Wilhelmstötter. Synopsys, [Online]. Disponible: https://www.openhub.net/p/jenetics
[5] TheOne. Akeranen, [Online]. Disponible: http://akeranen.github.io/the-one/
[6] L. Chancay. Dialnet, [Online]. Disponible: https://dialnet.unirioja.es/servlet/tesis?codigo=250265
[7] -. Wikipedia, [Online]. Disponible: https://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico
[8] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass y J. Scott. "Impact of human mobility on opportunistic forwarding algorithms". En: IEEE Transactions on Mobile Computing 6.6 (2007)
[9] S. Jain, K. Fall y R. Patra. Routing in a delay tolerant network. Vol. 34. 4. ACM, 2004
[10] M. Grossglauser y D. Tse. "Mobility increases the capacity of ad-hoc wireless networks". En: INFOCOM 2001.
[11] A. Vahdat, D. Becker y col. Epidemic routing for partially connected ad hoc networks. Inf. téc. Duke University, 2000
[12] T. Spyropoulos, K. Psounis y C. S. Raghavendra. "Spray and wait: an efficient routing scheme for intermittently connected mobile networks". ACM. 2005
[13] A. Lindgren, A. Doria y O. Schelén. "Probabilistic routing in intermittently connected networks". En: ACM SIGMOBILE
[14] J. Burgess, B. Gallagher, D. D. Jensen, B. N. Levine y col. "MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks." En: Infocom. 2006
[15] -. Chuwiki, [Online]. Available: http://chuwiki.chuidiang.org/index.php?title=Lectura_y_Escritura_de_Ficheros_en_Java