
# Xarxa neuronal amb topologia evolutiva per resoldre el problema del camí més curt d'un graf

**Joan-Antoni Vilaseca Giralt**
Grau en Enginyeria Informàtica
Àrea d'Intel·ligència Artificial

**Consultor/a: Dr. David Isern Alarcón**
**Professor/a responsable: Dr. Carles Ventura Royo**

Data Lliurament: 31/12/2019

Als meus fills, Núria, Guillem i Marcel
que m'han regalat el seu temps per
fer possible aquest Treball i tot el
Grau.

## FITXA DEL TREBALL FINAL

| | |
|---|---|
| **Títol del treball:** | *Xarxa neuronal amb topologia evolutiva per resoldre el problema del camí més curt d'un graf.* |
| **Nom de l'autor:** | *Joan-Antoni Vilaseca Giralt* |
| **Nom del consultor/a:** | *Dr. David Isern Alarcón* |
| **Nom del PRA:** | *Dr. Carles Ventura Royo* |
| **Data de lliurament (mm/aaaa):** | *12/2019* |
| **Titulació o programa:** | *Grau en Enginyeria Informàtica* |
| **Àrea del Treball Final:** | *Intel·ligència Artificial* |
| **Idioma del treball:** | *Català* |
| **Paraules clau** | *NEAT, especiació, algoritme Dijkstra* |


**Resum del Treball (màxim 250 paraules):** *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

En aquest TFG s'ha fet un estudi del funcionament de les xarxes neuronals neuroevolutives i de les propietats i característiques del mètode NEAT que bàsicament consisteix en l'evolució de la topologia a partir d'una estructura mínima i del manteniment d'un registre històric mitjançant un nombre d'innovació per tal de preservar les innovacions amb la classificació de les xarxes en diferents classes que anomenem espècies. L'objectiu era crear una implementació en Java d'aquest mètode que ens serveixi per a l'estudi del seu funcionament i per a aplicar-lo al problema de trobar el camí més curt d'un graf. Per això s'han estudiat les alternatives per resoldre el problema a través d'una xarxa neuronal i s'ha decidit implementar un nou algoritme que es basa en aplicar la xarxa Neat de forma recursiva i d'aquesta manera anar recorrent el graf. El projecte podria tenir interès en àrees on es treballi en grafs que varien de forma dinàmica, com l'enrutament de paquets per Internet o la conducció autònoma.

Per la implementació s'ha utilitzat el llenguatge Java i per adaptar-la al problema del camí més curt s'han modificat els mètodes originals per utilitzar-la de forma recursiva i afegir-hi algunes ajudes específiques del problema (no retorn pel mateix camí, control de cicles) que permetin millorar els resultats. Ens hem trobat que la implementació és capaç de solucionar el problema en grafs de mida petita mentre que en grafs de mida més gran, o amb grau dels vèrtexs major, és necessari millorar el seu comportament.

**Abstract (in English, 250 words or less):**

This TFG has performed a study of the functioning of neuro-evolutionary neural networks and the properties and characteristics of the NEAT method, which basically consists in the evolution of the topology from a minimum structure and the maintenance of a historical record through a number of innovations in order to preserve the innovations by classifying the networks into different classes called species. The goal was to create a Java implementation of this method that would allow us to study its details and apply it to the problem of finding the shortest path in a graph. That is why the alternatives to solve the problem through a neural network have been studied and it has been decided to implement a new algorithm that is based on applying the Neat network recursively and in this way go through the graph. The project could be of interest in areas where dynamically varying graphs are used, such as Internet packet routing or stand-alone driving.

The implementation has used the Java language and to adapt it to the problem of the shortest path the original methods have been modified to use it recursively and add some problem-specific help (no return on the same path, cycle control) to improve results. We have found that this implementation is able to solve the problem on small graphs, while on larger graphs or with higher grade on its vertices it is necessary to improve their behavior.

# Índex

## Llista de figures

# 1. Introducció

## 1.1 Context i justificació del Treball

En el camp de l'aprenentatge computacional, la combinació de les xarxes neuronals artificials amb els algoritmes genètics és una tècnica que permet resoldre una gran varietat de problemes. Per aconseguir aprenentatges més autònoms i independents del domini han aparegut nous mètodes que permeten modificar els pesos i ajustar la topologia de la xarxa simultàniament, essent NEAT un dels més destacats.

En aquest Treball de Final de Grau s'estudiaran les característiques de les xarxes neuronals i s'aprofundirà en les propietats i la implementació del mètode NEAT (NeuroEvolution of Augmented Topologies) estudiant-ne els canvis, requisits i avantatges respecte altres mètodes més coneguts. Per entrenar i provar la xarxa neuronal farem servir el problema per buscar el camí més curt d'un graf perquè per una banda podem calcular el resultat esperat aplicant l'algoritme de Dijkstra, i per l'altra ens aproxima el problema a camps més pràctics, com podria ser un assistent de navegació.

En el desenvolupament de noves tècniques per construir xarxes neuronals cal estudiar els avantatges que aporten aquest nou mètodes, tant en velocitat d'aprenentatge com en capacitat d'adaptació a nous problemes. Un dels problemes que intentarem resoldre en aquest TFG és la seva implementació en un llenguatge de programació i l'estudi de la dificultat a ajustar els paràmetres per a un problema en concret. A més a més farem la comparació dels resultats obtinguts per la nostra implementació del mètode NEAT resolent el problema del camí més curt d'un graf amb implementacions d'altres algoritmes com per exemple l'algoritme de Dijkstra.

Els camps d'aplicació d'aquest estudi són molt amplis ja que són molts els camps de la ciència i de la tècnica que utilitzen grafs per representar els seus elements, però la motivació inicial seria poder utilitzar aquests resultats per assistir a a agents de conducció autònoma, en ocasions en entorns dinàmics, on per exemple, poden canviar les connexions del graf en temps real.

## 1.2 Objectius del Treball

Els principals objectius que es pretenen assolir en aquest Treball Final de Grau i que es corresponen amb els resultats que es pretenen assolir en les dues entregues que es realitzaran durant el curs són:

- Implementació d'una xarxa neuronal NEAT (PAC2)
  - Propietats de les xarxes neuronals i tipus existents.
  - Estudi de les característiques i funcionament d'una xarxa neuronal basada en el mètode NEAT.
  - Implementació del mètode NEAT en un llenguatge de programació.
- Ajust de paràmetres per resoldre el problema del camí més curt d'un graf i comparació dels resultats obtinguts amb altres tipus de xarxes neuronals (PAC 3)
  - Ajust de paràmetres per utilitzar NEAT per resoldre el problema del camí més curt d'un graf.
  - Comparació dels resultats obtinguts amb la xarxa NEAT respecte dels obtinguts amb altres tipus de xarxes neuronals.

Altres objectius que es poden valorar en funció del temps disponible:

- Ús de la xarxa neuronal obtinguda per dirigir un agent dins d'un software MAS.
- Ús del mètode SUNA i anàlisis comparatiu de resultats.

## 1.3 Enfocament i mètode seguit

En aquest TFG implementarem una xarxa neuronal basada en el mètode NEAT, i després d'una fase d'investigació sobre les xarxes neuronals i el seu funcionament i l'estudi del comportament i dels requisits necessaris d'una NEAT passarem a programar-la des de zero definint les estructures de dades i les funcions que creguem necessàries per manejar-les.
Utilitzarem el llenguatge de programació Java, ja que és el que més hem utilitzat al llarg del Grau d'Enginyeria Informàtica. Per ajudar-nos en el desenvolupament utilitzarem Eclipse IDE.
Com que volem utilitzar les xarxes neuronals per resoldre el camí més curt d'un graf, necessitarem trobar un repositori de grafs, o estudiar la dificultat de crear un programa en Java que els pugui generar a partir d'un nombre aleatori nodes, arestes i pesos. També ens caldrà investigar les diferents tècniques per utilitzar una xarxa neuronal per resoldre el problema del camí més curt i analitzar i implementar un algoritme que ens permeti adaptar la nostra implementació al problema. Finalment necessitarem utilitzar algoritmes coneguts per comprovar si els resultats de la nostra xarxa són correctes.

## 1.4 Planificació del Treball

Per a la realització d'aquest TFG extraiem dels objectius principals que hem definit a l'apartat 1.2 diferents tasques que ens serviran per dur-los a terme.

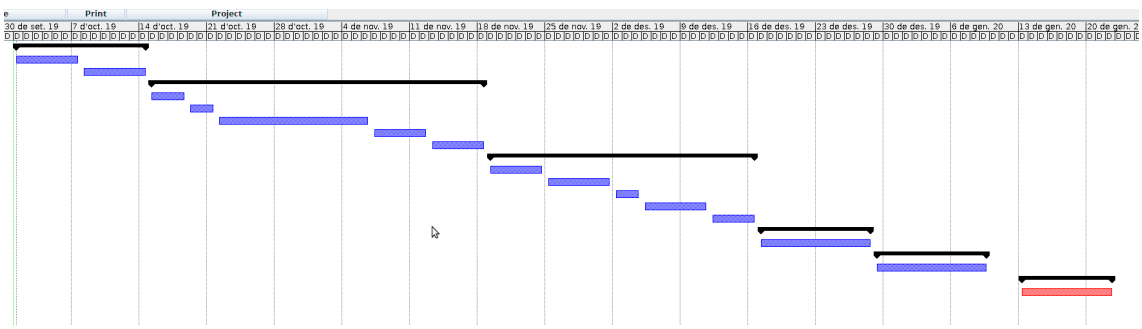
És important destacar que al final de cada fase deixem uns dies per dedicar a possibles imprevistos:

- En la fase d'implementació els principals riscos que ens podem trobar són problemes de programació i per tant, es poden mitigar preveient més temps per dedicar a la depuració del codi.
- En la fase de comparació d'algoritmes en la resolució del camí més curt ens podem trobar amb dos tipus de risc diferents:
  - Dificultats per aplicar la implementació al problema concret del camí més curt d'un graf: en aquest cas es podria valorar i consensuar amb el consultor l'aplicació de la xarxa neuronal en un domini més senzill.
  - Corba d'aprenentatge en la utilització d'altres algoritmes: es pot resoldre amb els dies que hem previst per mitigar imprevistos.

A continuació exposem les diferents tasques amb la seva planificació temporal i el Diagrama de Gantt associat:

| | Name | Start | Finish |
|---|---|---|---|
| 1 | **PLA DE TREBALL (PAC1)** | **1/10/19 8:00** | **14/10/19 17:00** |
| 2 | Definició i revisió del objectius | 1/10/19 8:00 | 7/10/19 17:00 |
| 3 | Descomposició en tasques i planificació temporal | 8/10/19 8:00 | 14/10/19 17:00 |
| 4 | **IMPLEMENTACIÓ D'UNA XARXA NEURONAL NEAT (PAC2)** | **15/10/19 8:00** | **18/11/19 17:00** |
| 5 | Estudi de les propietats de les xarxes neuronal i tipus existents | 15/10/19 8:00 | 18/10/19 17:00 |
| 6 | Estudi de les característiques i funcionament d'una xarxa neuronal basada en el mètode NEAT | 19/10/19 8:00 | 21/10/19 17:00 |
| 7 | Implementació d'una xarxa neuronal mitjançant el mètode NEAT | 22/10/19 8:00 | 6/11/19 17:00 |
| 8 | Definició i proves dels tests unitaris | 7/11/19 9:00 | 12/11/19 17:00 |
| 9 | Imprevistos o ús de la xarxa neuronal obtinguda per dirigir un agent dins d'un software MAS | 13/11/19 9:00 | 18/11/19 17:00 |
| 10 | **RESOLUCIÓ DEL CAMÍ MÉS CURT D'UN GRAF I COMPARACIÓ DE RESULTATS AMB ALTRES XN (PAC3)** | **19/11/19 9:00** | **16/12/19 17:00** |
| 11 | Especificació del conjunt de grafs de proves | 19/11/19 9:00 | 24/11/19 17:00 |
| 12 | Ajust de paràmetres de la implementació NEAT per resoldre el problema del camí més curt d'un graf | 25/11/19 9:00 | 1/12/19 17:00 |
| 13 | Cerca d'implementacions d'altres tipus de xarxes neuronals en biblioteques de software | 2/12/19 9:00 | 4/12/19 17:00 |
| 14 | Anàlisi i comparació de resultats resolent el mateix problema amb altres mètodes | 5/12/19 9:00 | 11/12/19 17:00 |
| 15 | Imprevistos o estudi del mètode SUNA | 12/12/19 9:00 | 16/12/19 17:00 |
| 16 | **REDACCIÓ DE LA MEMÒRIA (PAC4)** | **17/12/19 9:00** | **28/12/19 17:00** |
| 17 | Redacció i correcció de la memòria | 17/12/19 9:00 | 28/12/19 17:00 |
| 18 | **ELABORACIÓ DE LA PRESENTACIÓ (PAC 5a)** | **29/12/19 9:00** | **9/1/20 17:00** |
| 19 | Preparació de la presentació | 29/12/19 9:00 | 9/1/20 17:00 |
| 20 | **DEFENSA PÚBLICA (PAC 5b)** | **13/1/20 9:00** | **22/1/20 17:00** |
| 21 | Defensa del treball i respostes a les preguntes | 13/1/20 9:00 | 22/1/20 17:00 |

*Il·lustració 1: Planificació del TFG*




*Il·lustració 2: Diagrama de Gantt*

## 1.5 Breu sumari de productes obtinguts

En aquest TFG hem aconseguit una implementació d'una xarxa neuronal basada en el mètode NEAT programada en el llenguatge de programació JAVA.

Podem trobar el codi al repositori públic de GitHub:
https://github.com/jvilasecag/NEAT_UOC

També hem aconseguit aplicar aquesta implementació per resoldre el problema de trobar el camí més curt d'un graf amb diferents tipus de graf seguint una metodologia iterativa on s'aplica la xarxa a cada neurona per decidir quin camí ha de seguir. Finalment s'han comparat aquests resultats amb el resultat d'aplicar l'algoritme de Dijkstra.

## 1.6 Breu descripció dels altres capítols de la memòria

El capítol 2 està destinat a explicar les característiques de les xarxes neuronals i del funcionament i estructures necessaris per implementar el mètode NEAT. També s'analitza l'estat de l'art de les diferents implementacions d'aquest tipus de xarxes.

El capítol 3 ens explica els detalls i problemes que ens hem trobat en la implementació així com les estructures, classes i mètodes que hem utilitzat.

El capítol 4 es destina a analitzar els diferents algoritmes per calcular el camí més curt d'un graf i expliquem les dificultats per fer-ho amb una xarxa neuronal i diferents aproximacions que s'han utilitzat. Finalment expliquem un nou algoritme per adaptar el mètode NEAT al problema i com hem ajustat els mètodes de la nostra implementació per aconseguir-ho.

# 2. Xarxes neuronals i mètode NEAT

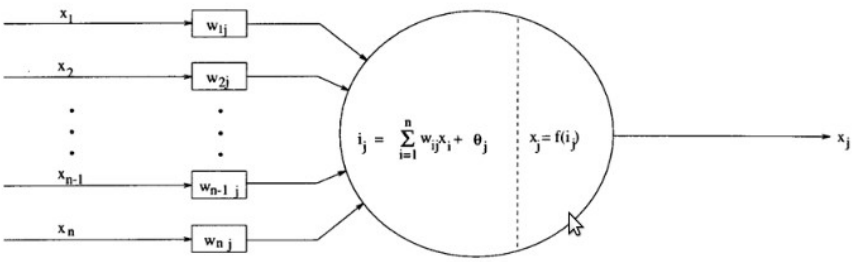
Les xarxes neuronals són un dels instruments principals per estudiar i comprendre com el cervell és capaç de resoldre problemes que fins fa poc semblaven reservats pels humans. [29] En el camp dels sistemes intel·ligents els investigadors han intentat resoldre els reptes que presenta la tecnologia informàtica: generalització, facilitat d'aprenentatge, flexibilitat i adaptació a nova informació i processos d'informació simultanis. Comparant la intel·ligència artificial amb la humana, el cervell humà és capaç de superar qualsevol sistema informàtic pel que fa a conceptes abstractes com com el reconeixement de patrons i el pensament creatiu. Els investigadors intenten resoldre aquest dèficit de la IA a través de xarxes neuronals artificials que intenten simular algunes de les característiques que presenta una complexa xarxa neuronal biològica. Una d'aquestes aproximacions és el mètode NEAT.

## 2.1. XARXES NEURONALS

Les xarxes neuronals són una simulació del funcionament i l'arquitectura del cervell humà i imiten la seva capacitat d'aprenentatge, de generalització i de processament ràpid de gran quantitat d'informació [3]. Aquestes xarxes reprodueixen les característiques de les sinapsis i de l'activació de neurones dels éssers vius i a partir d'uns estímuls (inputs) ens poden donar una resposta o valor de sortida (output). Cada element de procés representa una neurona i el conjunt de neurones connectades representa la xarxa neuronal. Cada neurona pot tenir diverses entrades i una única sortida. Si la suma d'inputs excedeix un cert valor límit la neurona s'activarà i emetrà un valor a la sortida. En el procés d'entrenament la xarxa s'adapta a través d'uns càlculs matemàtics i modifica el pes de les seves entrades.

Els elements principals d'una xarxa neuronal són [4]:

- Conjunt de neurones (o unitats de procés)
- Connexions entre unitats (definides per un pes ω).
- Funció que determina l'input total d'una neurona a partir del conjunt dels seus inputs.
- Funció d'activació que determina el nivell d'activació en funció del seu input.
- Un mètode per adquirir informació o regla d'aprenentatge.
- Unes dades que seran el senyal d'entrada i de les quals es pot calcular l'error de la sortida.




*Il·lustració 3: Activació d'una neurona*

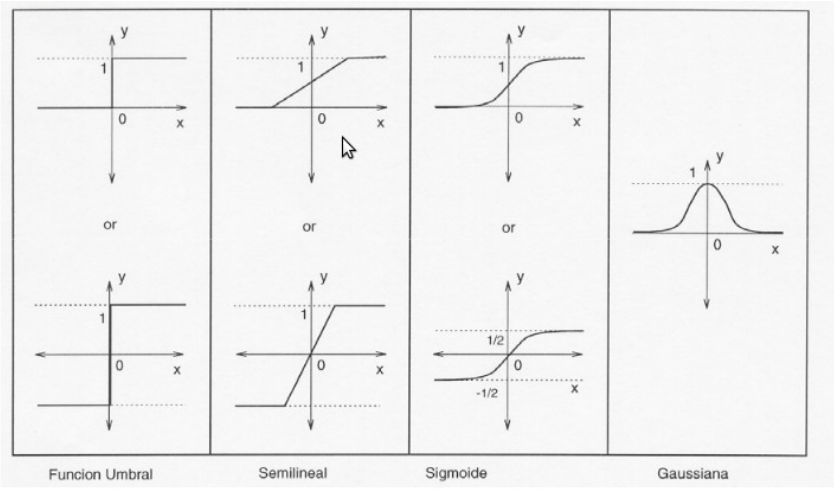
Les neurones reben els senyals d'entrada de les altres neurones i les utilitzen per calcular la seva senyal de sortida. A més a més s'encarreguen d'ajustar els pesos ω. Hi ha 3 tipus de neurones:

- input: reben les senyals de fonts externes de la xarxa.
- output: envien dades a l'exterior.
- hidden: reben i emeten senyals dins de la xarxa.

El comportament d'una unitat pot expressar-se a partir d'una equació que representa una suma ponderada dels seus inputs més un terme θ de biaix. El terme ω pot ser excitador (ω positiu) o inhibidor (ω negatiu).

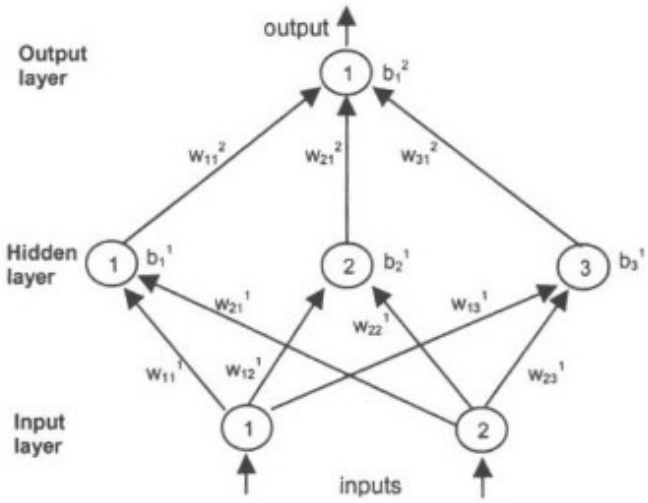
La funció d'activació, a partir del valor de l'input total de la neurona i de la seva activació actual, calcula el nou valor d'activació. Acostumen a ser funcions de tipus llindar acotades entre un valor màxim (activació total) i un de mínim (activació nul·la). Les funcions d'activació més utilitzades són [5]:

- Funció binària: l'activació és 1 si l'entrada supera el valor llindar o 0 o -1 si no el supera.
- Funció semilineal: la sortida valdrà 1 o 0 (o -1) si l'entrada es troba per sota o per sota d'uns valors llindar superior o inferior. Si es troba entre els dos valors la sortida es calcularà com una funció lineal a partir del seu valor d'entrada.
- Funció sigmoidal: funció acotada entre 0 i 1. El seu comportament és més suau en la zona de transició.
- Funció gaussiana: valors màxims quan l'entrada és propera a 0 i tendint a 0 a mesura que ens allunyem de 0.




*Il·lustració 4: Tipus de funcions d'activació*

Pel que fa a la topologia de la xarxa, les neurones s'acostumen a organitzar en capes i connectar-se a les neurones de la capa següent. El nombre de neurones, capes i connexions depèn del problema que s'ha de solucionar. Si totes les neurones es connecten exclusivament amb neurones de la capa posterior s'anomenen xarxes feedforward, mentre que si contenen connexions cap endarrere o cap a elles mateixes s'anomenen xarxes recurrents.




*Il·lustració 5: Topologia d'una xarxa neuronal*

El procés d'aprenentatge d'una xarxa neuronal consisteix en la modificació dels pesos de les connexions entre neurones per aconseguir el comportament desitjat. Aquest aprenentatge pot ser supervisat si s'aporten uns conjunts d'entrenament on tenim l'output esperat per cada conjunt d'entrada. Es calcula l'error a partir de la diferència entre la sortida obtinguda i l'esperada i a partir d'aquí es van ajustant els pesos fins a aconseguir un error més petit a un valor preestablert o que aquest valor sigui mínim. A partir d'aquest moment ja es podrà utilitzar aquesta xarxa funcionant amb els pesos aconseguits.

També es poden entrenar les xarxes mitjançant aprenentatge no supervisat on durant l'execució de la xarxa va adaptant els pesos de les seves connexions mitjançant la correlació existent entre les neurones implicades en les connexions. Durant l'aprenentatge per variar els valors dels pesos s'acostuma a utilitzar la retropropagació, on a partir de l'error calculat en cada output, es distribueix aquest error cap endarrere cap a totes les neurones que hi estan connectades i una de les tècniques més utilitzades per a l'optimització dels pesos és el mètode del descens del gradient que permet convergir cap a un mínim local o global de l'error mitjançant una tècnica computacional iterativa.

## 2.2. NEUROEVOLUCIÓ I MÈTODE NEAT

La neuroevolució és una forma de generar Xarxes Neuronals Artificials fent servir algoritmes evolutius que poden canviar els seus paràmetres, topologies i regles. El seu principal avantatge és que es poden utilitzar més àmpliament que els algoritmes d'aprenentatge supervisat on sempre es necessita un conjunt de dades correctes d'entrada i sortida, ja que els algoritmes neuroevolutius només necessiten una mesura del rendiment (fitness) de la xarxa a una tasca.

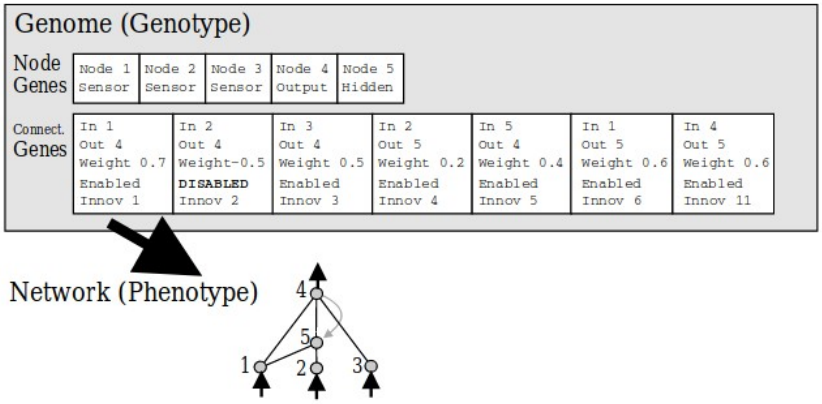
Per resoldre la problemàtica dins de la neuroevolució de desenvolupar la topologia de la xarxa al mateix temps que els seus pesos, Kenneth O. Stanley i Risto Mikkulainen, tots dos del departament de ciències de la Computació de la Universitat de Texas a Austin, van presentar el mètode NEAT (NeuroEvolution through Augmenting Topologies) [1][2]. Aquest mètode fa que sigui possible optimitzar i afegir complexitat a les solucions simultàniament fent que l'aprenentatge sigui més ràpid. En els mètodes tradicionals, s'escull una topologia (conjunt de nodes i connexions entre ells) abans de començar l'experiment. Habitualment consisteix en una capa de neurones ocultes que es troben connectades a cada entrada i a cada sortida de la xarxa. L'evolució busca en l'espai dels pesos d'aquestes connexions permetent reproduir-se a les xarxes amb un millor rendiment.

Tot i això, els pesos de les connexions no són l'únic aspecte de les xarxes neuronals que afecten al seu comportament, sinó que la mateixa estructura de la xarxa també afecta a la seva funcionalitat [24]. Modificar la seva estructura és una part efectiva en l'aprenentatge de la xarxa. Un dels avantatges de l'evolució de la topologia, consisteix en l'estalvi d'esforços humans per intentar decidir quina és l'estructura de la xarxa neuronal més adient per a un problema particular.

Darrerament s'han desenvolupats molts sistemes amb l'objectiu de desenvolupar simultàniament els pesos i la topologia de la xarxa [25]. Aquests mètodes s'agrupen sota la denominació TWEANNs (Topology and Weight Evolving Artificial Neural Networks). Habitualment utilitzen codificació directa, com els autors de NEAT han escollit pel seu mètode, on s'especifica en cada genoma totes les connexions i nodes que apareixeran a la xarxa, enlloc de la codificació indirecta on l'estructura es determina mitjançant un conjunt de regles.

El mètode NEAT es basa en tres tècniques principals[26]:

- Manteniment d'un registre històric per fer un seguiment dels gens i del seu orígens.
- Protecció de la innovació a través de l'especiació.
- Complexificació de la xarxa amb un creixement gradual a partir d'una estructura mínima.

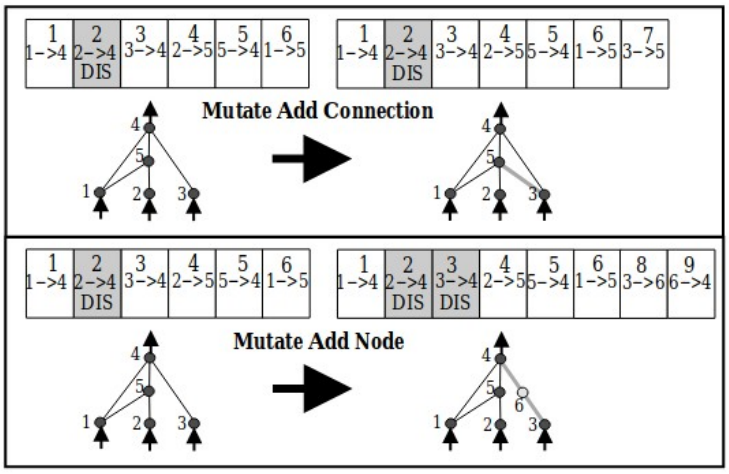



*Il·lustració 6: Codificació xarxa NEAT*

La innovació es crea a través de les mutacions que poden afegir noves estructures a la xarxa. Aquestes mutacions estructurals poden ser:

- Afegir un nou node a la xarxa: una connexió aleatòria és dividida i s'hi afegeix un nou node i dos noves connexions.
- Afegir una nova connexió: una nova connexió amb pes aleatori és afegida entre dos nodes aleatoris que prèviament no estaven connectats.

A més a més, l'operació de mutació també pot variar els pesos de les connexions de manera similar a altres sistemes neuroevolutius on cada connexió té una probabilitat de ser pertorbada a cada generació.



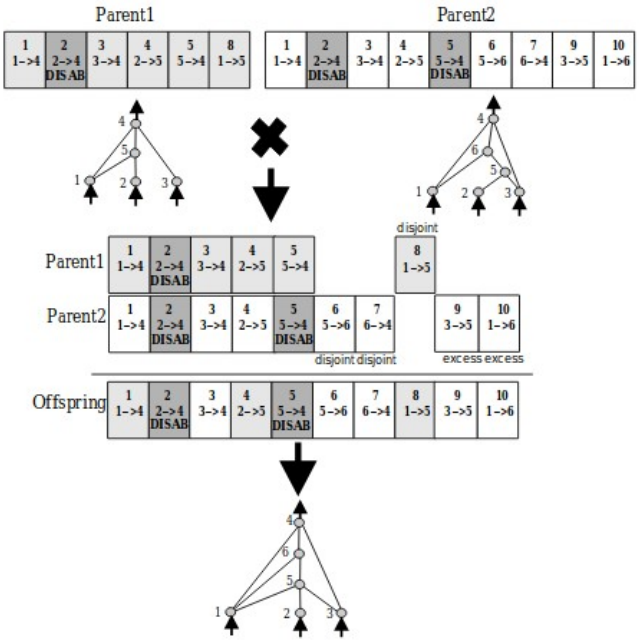

*Il·lustració 7: Mutacions estructurals NEAT*

**Seguiment dels gens a través d'un registre històric**

El mètode NEAT fa un seguiment històric de l'origen de cada gen. Cada vegada que hi ha una mutació estructural on es crea un gen nou, un nombre d'innovació global és incrementat i assignat a aquest gen. Per tant, els nombres d'innovació representen una cronologia de l'aparició de cada gen en el sistema.

El manteniment d'una llista de les innovacions que han aparegut a cada generació permet que es doni el mateix nombre d'innovació quan la mateixa estructura apareix idèntica més d'una vegada a la mateixa generació. Aquest nombre permet saber quins gens es poden alinear amb quins i en el procés de creuament de gens s'alineen els que tenen el mateix nombre d'innovació. Per tant els gens no alineats (excess o disjoints) representen les estructures que no estan presents en l'altre genoma.

Els descendents adopten un dels gens d'uns dels dos pares de forma aleatòria en els gens coincidents (matching genes), mentre que per la resta de gens s'agafen els gens del pare amb millor valor de fitness.




*Il·lustració 8: Encreuament de gens*

**Protecció de la innovació a través de l'especiació**

En els mètodes TWEANNs, la innovació consisteix en afegir noves estructures a la xarxa a través d'una mutació, però moltes vegades aquests canvis produeixen una davallada del seu «fitness» abans que els seus pesos tinguin l'oportunitat d'optimitzar-se. Això fa que aquestes innovacions estructurals tinguin poques oportunitats de sobreviure abans que la seva aptitud o fitness pugui millorar. Per això, NEAT protegeix les xarxes amb noves estructures a través de l'especiació perquè es puguin optimitzar abans de competir amb tota la resta de la població.

Per poder separar els genomes en espècies es necessita una funció de compatibilitat que ens digui si dos genomes haurien de pertànyer a la mateixa espècia. Com que és difícil de definir aquesta funció entre xarxes amb topologies diferents, es fa servir la informació històrica dels gens per definir una funció de distància entre els gens.

Considerem que el nombre de gens que una xarxa té de més (excess genes) o que no tenen les dos (disjoint genes) és una mesura natural de la seva compatibilitat. Per tant calculem la seva distància com una combinació lineal entre els excess (E) i els disjoint (D) genes i la mitjana de la diferència de pesos ($\overline{W}$) dels gens coincidents:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \bar{W}$$

amb N essent el nombre de gens del genoma més gran.

A cada generació els genomes són distribuïts de forma seqüencial en espècies i per tant cal gestionar i mantenir un llistat de les espècies actuals amb un representant que serà un genoma aleatori de la generació anterior. Cada genoma de la generació actual es classificarà dins de la primera espècie on la distància amb el seu representant sigui més petita que un llindar preestablert. En cas contrari es crearà una nova espècie per aquest genoma.

Com a mecanisme de reproducció per NEAT es fa servir «explicit fitness sharing» on totes les xarxes dins d'una mateixa espècie han de compartir el fitness del seu nínxol i d'aquesta manera cap espècie pot arribar a ser massa gran encara que molts dels seus genomes tinguin un bon rendiment i fa que sigui improbable que tota la població acabi essent una sola espècie.

La funció ajustada de fitness per cada genoma es calcula d'acord a la distància als altres organismes de tota la població. Com que la funció sh és igual a 0 quan la distància és superior a la que ens marca el llindar per separar espècies i 1 en els altres casos, la suma de sh es redueix al nombre de genomes que pertanyen a la mateixa espècie. A continuació les espècies es reprodueixen eliminant en primer lloc als seus membres menys adaptats. A cada espècie li correspondrà un nombre diferent de descendents en proporció a la suma dels f' de tots els seus membres:

$$f'_i = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))}$$

**Minimitzant la dimensionalitat començant amb una estructura mínima.**

Com que la reducció de la dimensionalitat en una xarxa és desitjable per tal de reduir el nombre de paràmetres de l'espai de cerca i millorar el rendiment i el temps de resposta de la xarxa, molts TWEANN incorporen la mida de la xarxa en el càlcul de la seva funció d'ajust o fitness.

En el mètode NEAT, enlloc de començar a partir d'una població inicial de topologies aleatòries, s'intenta esbiaixar el resultat cap a un espai de mínimes dimensions iniciant la població com un conjunt de neurones uniformes amb zero nodes ocults i tots els nodes d'entrada connectats amb els nodes de sortida. Les noves estructures s'afegeixen de forma gradual al ritme que van succeint les diverses mutacions estructurals i sempre són justificades ja que sempre afavoreixen la solució del problema.

Com que es comença des d'un espai de dimensionalitat mínima, el mètode NEAT sempre es troba cercant en un espai de menys dimensions que altres mètodes TWEANN o de topologia fixa afavorint el seu rendiment.

## 2.3. ESTAT DE L'ART

Avui en dia, l'algorisme NEAT ha demostrat que pot ser molt útil per resoldre problemes mitjançant xarxes neuronals, especialment quan no es coneix prèviament quina topologia ha de tenir,i hi ha varis estudis que demostren que pot ser més eficient que altres mètodes coneguts. Per aquest motiu no és estrany que trobem moltes implementacions del mètode, en molts llenguatges de programació diferents (C#, C++, Java, Python, Go , Delphi, Matlab,..), moltes d'elles provinents d'universitats i que s'ofereixen en codi obert. Algunes d'elles també inclouen experiments com el XOR o el de l'equilibri d'un pèndol.

Podem trobar un recull molt interessant d'aquestes fet pel grup de recerca de la University of Central Florida. [7] Una de les més interessants és la que va desenvolupar Kenneth Stanley ja que amb Miikkulainen són els autors que van descriure originalment els principals punts per implementar un algoritme genètic basat en NEAT en els seus assajos [1] i [2]. Es tracta d'implementació en C++ i el van utilitzar per jugar al joc de taula GO. Aquest algorisme el podem trobar a la universitat de Texas. [8]

Una de les implementacions més utilitzades i populars és SharpNEAT [9] escrita per Colin Green en C#. Algunes de les seves principals característiques és que incorpora mutacions d'eliminació, a més de les d'addició, i d'aquesta manera ajuda a preservar una topologia de la xarxa més simple. Un altre característica a destacar és que fa servir l'algorisme d'agrupament K-means

per a l'especiació dels genomes de l'algoritme. Una de les seves utilitzacions ha estat per la creació d'un programa anomenat Genetic Art que a partir de les seleccions que fa un usuari l''algoritme crea imatges que són del gust de l'usuari. També l'ha utilitzat la Universitat de Nevada per crear solucions per gestionar i proporcionar un comportament intel·ligent als diferents elements dinàmics que participen en un joc d'estratègia a temps real. [17]

Pel que fa a les implementacions Java, que és el llenguatge en el qual volem implementar la nostra versió de NEAT, cal destacar la de Ugo Vierucci també de la Universitat de Texas i basat en el paquet original de la implementació escrita en C++ per Kenneth Stanley. Està implementada en llenguatge Java i s'anomena JNEAT [10]. S'ha utilitzat per determinar el comportament automàtic dels bots al joc Toribash. [18]

Una altra implementació molt popular de l'algorisme escrita en Java és ANJI (Another NeatJava Implementation). Utilitza el paquet JGAP (Java Genetic Algorithm Package) per implementar bona part de les funcions informàtiques evolutives. També té una variant per implementar l'extensió HyperNEAT. També s'ha utilitzat per controlar elements d'un joc, en aquest cas de Robocode [15], i també té un extensió que es pot fer servir per sitemes de visió 2D (Action Vision Systems) [11].

Finalment, també cal destacar la implementació de Cesar G. Miguel i Carolina Feher da Silva anomenada NEAT 4J que permet una execució multifil i possibilitant d'aquesta manera una evolució distribuïda. Els mateixos autors l'han utilitzat per l'anàlisi del Forex Trading [12]. També s'ha utilitzat per controlar cotxes en jocs de carrera com Torcs. [14]

| Taula resum implementacions mètode NEAT | | | |
|---|---|---|---|
| Nom | Llenguatge | Característiques | Aplicacions |
| Neat-C | C++ | Implementació original | Joc de taula Go<br>Pèndols balancejats |
| SharpNEAT | C# | Mutacions d'eliminació<br>K-means per a l'especiació | Genetic Art<br>Strategy Game Micromanagement |
| JNEAT | Java | Basada en la implementació original | XOR problem<br>Comportament dels bots de Toribash |
| ANJI | Java | Fa servir el paquet JGAP | Robocode<br>Active Vision System |
| NEAT 4J | Java | Execució multifil<br>Evolució distribuïda | Forex Trading Analisys<br>Control de cotxes a Torcs |

*Il·lustració 9: Taula implementacions NEAT*

També existeixen implementacions en altres llenguatges de les extensions de l'algoritme NEAT com HyperNEAT o Adaptative HyperNEAT. Alguns d'ells implementen l'algorisme de cerca anomenat Novelty Search que consisteix en premiar la innovació més que la proximitat a l'objectiu final. Un d'ells escrit en Java és AHNI (Another HyperNeat Implementation) desenvolupat per Oliver Coleman de la University of New South Wales. [13]

Tot i ser un algoritme amb múltiples implementacions disponibles, ens centrarem a efectuar una nova implementació d'aquesta tècnica amb les següents particularitats respecte a les ja existents:

- Simplificació de tots els mecanismes d'activació de la neurona fent servir una sola funció d'activació.
- Diferenciació de les diferents mutacions que porten a l'evolució de la topologia en classes específiques.

Aquests canvis pretenen aconseguir els objectius següents en la nostra implementació:

- Destacar les característiques pròpies del mètode NEAT per ajudar al seu estudi i a la seva comprensió, tant a nivell personal com per qualsevol grup de recerca o d'estudi interessats en comprendre el funcionament del mètode NEAT.
- Preparar una implementació que sigui fàcilment adaptable al problema del camí més curt d'un graf que volem resoldre i que ens permetrà desenvolupar un mètode NEAT interactiu variant alguns dels mètodes que hem implementat per la solució genèrica de la primera fase del treball.

La segona fase del Treball de Final de Grau consisteix en utilitzar aquesta implementació per intentar resoldre el problema del camí més curt d'un graf. Resoldre el problema més curt d'un graf és un problema complex ja que els mètodes tradicionals no escalen bé quan es troben amb grafs molt grans. Com a exemple tenim l'algoritme de Dijkstra amb una complexitat O(m + n.log(n)) on n és el nombre de nodes i m el nombre d'arestes. Altres tècniques que utilitzen heurístiques per aproximar els resultats a camps de cerca més petits poden arribar a aconseguir millors rendiments com l'aproximació anomenada landmark-based on es trien un conjunt de nodes com a fita i es precalcula la distància entre ells per aconseguir una complexitat lineal respecte al nombre de fites. [30] També s'han utilitzat xarxes neuronals per intentar resoldre el problema i també utilitzen diverses aproximacions per adaptar-se al problema. A la secció 4.1 tenim trobem una exposició més detallada d'aquestes tècniques i de les seves característiques.

# 3. Implementació del mètode NEAT en Java

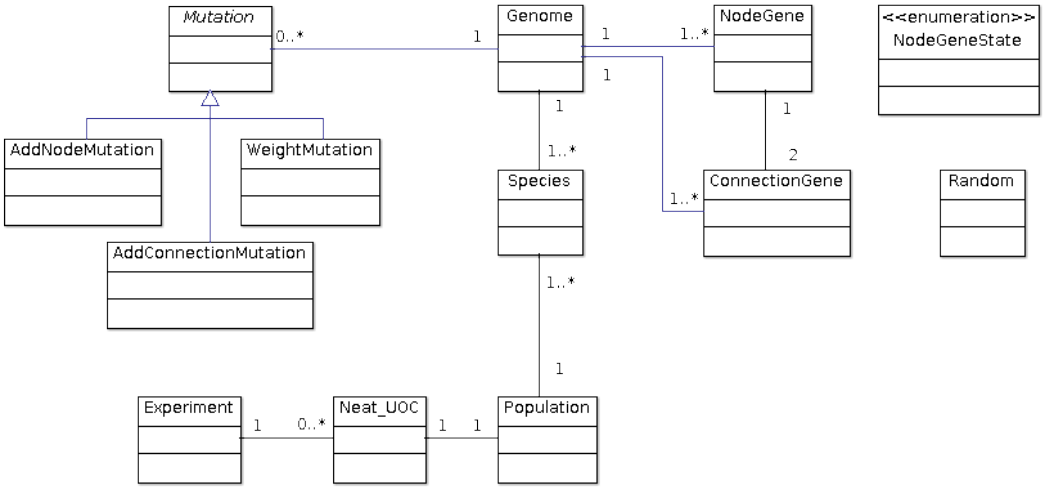
En la nostra implementació intentarem simplificar les complexitats de la implementació d'una xarxa neuronal, per tal de centrar-nos en les característiques específiques del mètode NEAT i que pugui servir com a exemple per estudiar el seu funcionament. Per tant, els mecanismes utilitzats per aconseguir aquests objectius han estat:

- Simplificació funcionalitats xarxa neuronal:
  - Funció d'activació binària sense mecanismes per utilitzar-ne d'altres.
  - No hi ha retropropagació del fitness dins les neurones de la xarxa.
  - Mètode recursiu per calcular el valor d'activació de cada neurona.
  - No s'utilitzen neurones de biaix.
- Característiques específiques NEAT:
  - La mutació és una classe abstracta de la qual hereten les mutacions específiques del mètode:
    - Afegir node.
    - Afegir connexió.
  - Gestió específica del nombre d'innovació.
  - Cada genoma guarda la representació de la xarxa (mitjançant una llista de nodes i una altra de connexions).
  - Les xarxes neuronals es guarden dins d'espècies i la població total de xarxes només contindrà un llistat d'espècies.

Podem trobar el codi al repositori de Github:
https://github.com/jvilasecag/NEAT_UOC

## 3.1. DECISIONS DE DISSENY I DEFINICIÓ DE LES CLASSES

A continuació fem una explicació de la funció de les diferents classes de la implementació i l'explicació de perquè s'han definit d'aquesta manera en la fase de disseny. L'explicació de tots els seus mètodes es pot trobar en la documentació que acompanya el codi font i que s'ha generat mitjançant JavaDoc. També incloem el diagrama de classes de la implementació:




*Il·lustració 10: Diagrama de classes implementació*

**Genome**
Aquesta és la classe central de tota la implementació i representa tant el genoma com la Xarxa Neuronal generada per aquest genoma. Els seus atributs principals són el valor d'ajust del seu rendiment (fitness) i una llista dels nodes que conté i una altra de totes les connexions de la xarxa.

En aquest punt aprofitem que el mètode NEAT manté un registre històric d'aquests gens mitjançant un nombre que anomenem innovationNumber per implementar la llista de connexions com un TreeMap per tal de disposar d'una clau que serà el mateix innovationNumber i poder executar les tasques de creuament de gens (crossOver) en la reproducció de genomes de manera més eficient. També serà l'encarregat de calcular les distàncies entre genomes o d'iniciar l'activació dels seus nodes.

**NodeGenes**
Representen les diferents neurones de la xarxa. Hem decidit mantenir a cada node una llista de les connexions entrants i una altra amb els nodes origen d'aquestes connexions. Això ens facilita la gestió de les estructures de la xarxa, sobretot a l'hora de modificar connexions degut a que afegim un nou node a la xarxa i a l'hora d'activar les neurones. Els altres atributs de la classe són un enter que representa la seva id, el valor resultant de la suma de les activacions de les connexions entrants i el seu estat. Els seus possibles valors d'estat són input, output o hidden depenent de si es tracta d'una neurona d'entrada, de sortida o oculta i l'implementem amb una classe d'enumeració anomenada NodeGeneState.

Aquesta classe conté el mètode recursiu que ens permet calcular el valor d'activació de cada neurona. El mètode recursiu finalitza quan arriba a una senyal d'entrada o quan detecta un cicle en la xarxa. Per detectar els cicles

mantenim una llista de nodes visitats durant la recursivitat i quan es troba amb un node repetit atura la recursivitat per evitar continuar l'activació en un bucle infinit. La funció d'activació que hem escollit és la funció binària per simplificar la implementació.

**ConnectionGenes**

Aquesta classe representa les connexions entre neurones. La connexió queda definida pels atributs node inicial, node final i pes (weight). A més a més li hem definit una variable estàtica innovationCount que representa el marcador històric i que incrementem cada vegada que creem una nova connexió i guardem el valor de cada una a l'atribut innovationNumber. També guardem en un atribut booleà si la connexió està activa. El mateix constructor de la classe és l'encarregat d'actualitzar les llistes de nodes i connexions entrants del node que es troba a la sortida de la connexió.

**Population**

Classe que representa la població o el conjunt de xarxes amb que treballem per evolucionar i escollir les que millor s'adapten al problema. Com que en NEAT els genomes s'agrupen en espècies, la classe Population només tindrà un llistat amb les espècies (Species) existents en cada moment. El seu constructor és l'encarregat de crear un nombre d'espècies predeterminat i poblar-les amb genomes de dimensionalitat mínima (sense nodes ocults i amb totes les entrades connectades a totes les sortides).

A més a més conté un atribut enter per portar un registre del número de generació que ens trobem per poder determinar el final de l'evolució. Conté els mètodes encarregats de mutar tots els genomes de la població, de reproduir-los i d'especiar-los (separar-los en diferents espècies en funció de la seva distància). També conté un mètode per eliminar les espècies que s'hagin quedat buides.

**Species**

Aquesta classe representa les diferents espècies on es classifiquen els genomes. Per tant conté una llista amb tots els genomes que en formen part. Com que aquestes llistes són l'únic contenidor de genomes que es troba en tota la implementació, tot genoma que existeixi en la nostra població sempre es trobarà ubicat dins d'una espècia. A més a més s'identifica amb un representant que és un genoma que no necessàriament ha d'existir ja que pertany a una generació anterior.

Cada espècie guarda el valor del fitness mitjà dels seus genomes, el mètode per calcular-lo i els mètodes per ordenar-ne el seu llistat en funció del valor del seu fitness, el mètode per calcular el nombre de genomes que han de sobreviure a cada generació i un altre per eliminar-ne els sobrants. A més a més en els seus atributs guardem el valor del millor fitness i quants passos o generacions han passat des de la darrera vegada que s'ha modificat per poder

determinar que una espècia ja no s'espera que millori més. Com que el fitness dels genomes és la diferència quadràtica de les sortides obtingudes amb els valors esperats, els fitness més ajustats es correspondran amb els que tenen el valor més petit.

**Mutation**
Classe abstracta que representa una mutació i que conté el genoma que ha de mutar. Aquesta classe, així com totes les que hereten d'ella potser no seria necessària ja que tots els seus mètodes es podrien incloure a la classe Genome, però d'aquesta manera em va semblar que s'explicitava millor les característiques i tipus de canvis estructurals que permet el mètode NEAT.

**AddNodeMutation**
Classe que hereta de Mutation i que defineix una mutació on una connexió és separada i un nou node és introduït. El pes de les noves connexions és el mateix que tenia la connexió que hem trencat per la connexió que es troba abans del nou node i és igual a 1 per la que surt del nou node.

**AddConnectionMutation**
Classe que hereta de Mutation i que representa una mutació estructural on s'afegeix una nova connexió entre dos nodes existents. Com que el seu mètode mutation() ha de comprovar que la connexió no existeixi, que el node origen no sigui un output o que el node destí no sigui un input i que l'origen i destí de la connexió no sigui el mateix node, s'ha afegit un comptador per descartar l'intent de mutació si no es troben dos nodes que es puguin connectar després d'un determinat nombre d'intents. Hi poden haver casos on realment no existeixi la possibilitat d'introduir una connexió perquè totes les connexions possibles ja es troben en el genoma. Hem de tenir en compte que en la nostra implementació hi poden haver dos connexions diferents, una en cada sentit, entre dos nodes.

**WeightMutation**
Classe que hereta de Mutation i que permet la mutació dels pesos de totes les connexions d'un genoma. En cada connexió es calcula aleatòriament si s'ha de modificar el seu pes en funció d'una probabilitat predefinida.

**Experiment**
Aquesta classe representa les característiques de l'entorn o de la tasca on apliquem la nostra implementació de la xarxa neuronal. Guarda una llista amb els valors de les entrades i unes llistes amb els valors de les sortides esperades i de les sortides reals si existeixen. Aquesta classe és necessària per donar un context a l'activació de les neurones de la xarxa neuronal.

**NEAT_UOC**
Aquest classe representa els passos principals de l'algoritme NEAT i és el que fa evolucionar la xarxa neuronal. Per aquest motiu és el que conté un mètode main i al que se li poden passar els inputs i outputs (falta implementar) de la tasca per la qual volem entrenar la xarxa.

**Random**
Classe auxiliar per definir els mètodes que ens permeten calcular valors aleatoris de diferents tipus de dades.

## 3.2. TESTS UNITARIS I PROBLEMA DE LA SUMA 3+2

Per provar el correcte funcionament de la implementació hem creat dos tipus diferents de proves:

- Tests unitaris JUnit: la classe TestNEAT proporciona un conjunt de tests per comprovar el correcte funcionament dels constructors de les diferents classes i dels seus mètodes utilitzant el framework JUnit. L'èxit en els tests ens garanteix el correcte funcionament de la implementació així com la seguretat de que els canvis que efectuem en el codi no modifiquen altres parts que ja funcionaven correctament. Abans de cada test hem de reinicialitzar la variable estàtica innovationCount.
- Problema de la suma 3+2: Problema molt simple però que ens permet entrenar una xarxa fàcilment i després utilitzar-la per predir el resultat. Com que ens trobem que els resultats predits es troben dins d'un error d'un 20% i molts dins d'un 10% podem determinar ràpidament que la xarxa serveix per solucionar el problema.

| Error | 0%-5% | 5%-10% | 10%-20% | >20% |
|---|---|---|---|---|
| Nombre prediccions | 67% | 20% | 13% | 0 |

*Il·lustració 11: Taula resultats Experiment 3+2*

## 3.3. AJUST DE PARÀMETRES

En aquesta implementació definim un conjunt de paràmetres que es poden ajustar en funció del problema que s'intenti resoldre i del comportament que esperem de l'algoritme. Aquests paràmetres es troben definits com atributs estàtics de la classe NEAT_UOC i s'han de modificar en el mateix fitxer font si es volen canviar.

A la següent taula donem una explicació de la funció de cada paràmetre i del valor per defecte utilitzat:

| Taula de paràmetres NEAT_UOC | | |
|---|---|---|
| Paràmetre | Significat | Valor |
| maxGenomes | Nombre màxim de genomes i que serveix per inicialitzar la població | 50 |
| inputNodes | Nombre de neurones de la capa d'entrada | 4 |
| outputNodes | Nombre de neurones de la capa de sortida | 1 |
| maxInitialSpecies | Nombre d'espècies en què dividirem els genomes de la població inicial | 10 |
| maxGenerations | Nombre de generacions fins que finalitzi l'algoritme | 200 |
| fitnessThreshold | | 0.9 |
| SurvivalThreshold | Llindar de supervivència que ens determina el percentatge de genomes que sobreviuran a cada espècie en cada generació | 0.5 |
| WeightMutationProb | Probabilitat de que a cada generació el pes d'una connexió variï | 1 |
| AddNodeMutationProb | Probabilitat de que a cada generació s'afegeixi un node a la topologia de la xarxa neuronal | 0.5 |
| AddConnectionProb | Probabilitat de que a cada generació s'afegeixi una nova connexió a la topologia de la xarxa neuronal | 0.5 |
| c1 | Coeficient pels ExcessGenes per calcular la distància entre genomes | 1 |
| c2 | Coeficient pels DisjointGenes per calcular la distància entre genomes | 1 |
| c3 | Coeficient per avgWeightDifference per calcular la distància entre genomes | 0.4 |
| speciationThreshold | Llindar per la distància entre genomes en què es considera que formen part de la mateixa espècia | 0.5 |
| CountAddConnection Mutation | Nombre d'intents per intentar trobar aleatòriament dos nodes vàlids per afegir una nova connexió | 50 |

*Il·lustració 12: Taula paràmetres implementació NEAT*

Els creadors del mètode, K.O. Stanley i R.Miikkulainen van fer servir uns paràmetres [1] per comprovar el mètode NEAT amb els seus experiments de prova i aquests seran els que utilitzarem en la nostra implementació. Com que la principal característica del mètode és que s'ha d'adaptar a qualsevol experiment, també seria òptim que el mateix ajust de paràmetres servís per tots ells.

Es fa servir una població amb 150 xarxes NEAT,. Els coeficients per mesurar la compatibiltat són c1=1, c2= 1 i c3=0,4. La distància llindar és δt=3. Si el fitness màxim d'una espècie no millora en 15 generacions, les xarxes en aquesta espècie no es continues reproduint. A més el genoma amb millor fitness de cada espècie amb més de 5 xarxes és copia sense canvis a la següent generació.

Les probabilitats de cada genoma de canviar els pesos de les seves connexions a través d'una mutació és del 80%, i en aquest cas cada pes té un 90% de probabilitats de ser pertorbat i un 10% de ser canviat per un valor aleatori. La probabilitat d'afegir un nou node i una nova connexió és de 0,3 (0,03 i 0,05 en poblacions petites). Els autors fan servir una funció de transferència sigmoidal a tots els nodes. També estipulen un 75% de probabilitats que un gen heretat sigui deshabilitat si es troba deshabilitat en tos dos gens pares. A més a cada generació un 25% dels descendents provenen d'una mutació sense creuament de genomes.

# 4. Problema del camí més curt d'un graf

Trobar el camí amb distàncies més curtes entre els nodes d'un graf és un problema molt important en moltes aplicacions. Per exemple, el nombre de salts d'una persona a una altra en una xarxa social ens indica el seu nivell de confiança. [30] Els mètodes tradicionals per calcular la distància entre nodes no escalen eficientment amb la mida dels grafs. A continuació repassem les tècniques més importants i acabem estudiant de quina manera podríem utilitzar la xarxa neuronal que hem implementat per resoldre aquest problema.

## 4.1. ALGORITMES D'ENCAMINAMENT

Un algoritme d'encaminament consisteix en el procés de trobar un camí en un graf. [19] Un graf consisteix en un conjunt de nodes connectats mitjançant enllaços. Les codificacions més habituals dels grafs acostuma a ser mitjançant una matriu d'adjacències o a través d'una llista d'adjacències que conté tots els nodes amb les connexions als següents nodes. Les connexions poden ser dirigides, no dirigides, amb pes o sense. Un camí consisteix un conjunt de nodes dels graf connectats que van des del node inicial al final.

Una dels problemes principals dels grafs és el de trobar el millor camí des d'un node inicial a un altre node. En el cas de grafs amb pesos a les connexions el millor camí acostuma a ser el camí amb un cost o suma de pesos mínim. Un altre dels problemes clàssics dels grafs és el problema dels ponts de Könisgsberg que consisteix en trobar un camí que recorri totes les connexions però utilitzant-les una sola vegada. Un altre problema clàssic és el del «Travelling-Salesman» on l'objectiu és el de trobar un camí on cada node es visiti almenys una vegada.

### 4.1.1. Algoritme de Dijkstra

L'algoritme de Dijkstra va ser publicat per E.W.Dijkstra l'any 1959. L'objectiu és trobar el camí òptim amb un mínim de nodes analitzats o amb la creació d'un arbre de cerca mínim. L'arbre de cerca pot ser desenvolupat mitjançant la tècnica de cerca en amplada o «Breadth-First Search» (BFS) o de cerca en profunditat, «Depth-First Search» (DFS).

L'algoritme de Dijkstra es basa en el BFS intentant troba el millor camí d'una forma més ràpida abans que tot l'arbre de cerca sigui desenvolupat. L'algoritme BFS utilitza tres llistes: una llista amb tots els nodes, una anomenada oberta amb els nodes actius i una altra llista anomenada llista tancada i que conté tots els nodes passius que ja han estat analitzats. Inicialment la llista oberta i la tancada es troben buides i l'algoritme passa el node inicial de la llista amb tots els nodes a la llista oberta i a continuació es mouen també tots els nodes connectats a aquest node al final de la llista oberta i es passa el node analitzat a la llista tancada. Aquest procediment es repeteix mentre quedin node a la llista oberta. L'algoritme de Dijkstra també utilitza aquestes tres llistes per

guardar els nodes. A més defineix una funció de cost g(X) que es calcula cada vegada que un node és traspassat a la llista oberta com la funció recursiva:

$$g(X)=g(C)+v(C,X)$$

La funció g(X) ens indica la llargada del camí del del node X al node inicial com la suma de la llargada al node inicial del node anterior C més la llargada des del node anterior C a X. A més a més en l'algoritme de Dijkstra la llista oberta és ordenada en funció del cost g de cada node. D'aquesta manera l'algoritme de cerca sempre s'esbiaixarà cap al node amb un camí més curt fins al node inicial. L'algoritme s'acaba quan el node final es trobi al capdamunt de la llista oberta i en aquell moment la llista tancada contindrà el millor camí.

### 4.1.2. Algoritme A*

L'algoritme A* és una extensió de l'algoritme de Dijkstra que intenta millorar-lo mitjançant una heurística. Aquesta intenta dirigir el camí cap a l'espai de cerca correcte. El tipus d'heurística utilitzada depèn del problema en concret on l'utilitzem:

- Millor node primer: En aquest cas es determina quin node s'hauria d'analitzar primer per tal d'aconseguir la millor recompensa enlloc d'utilitzar un BFS o DFS estricte.
- Selecció d'enllaços: L'heurística determina quina connexió de les que surten d'un node cap a un node no explorat s'ha de triar.
- Poda: Aquesta heurística pot podar parts del graf i eliminar aquells subgrafs que no contenen el node final i d'aquesta manera reduir l'espai de cerca.

Per exemple, si utilitzem l'heurística del millor node primer, cal definir una nova funció de cost que complementi la funció g de l'algoritme de Dijkstra i que tingui en compte la distància estimada fins al node objectiu:

$$f(X)=g(C)+v(C,X)+h(X)$$

El cost o distància exacte fins al node final no es pot calcular i com es defineixi h(X) perquè sigui òptim dependrà del problema. Per exemple, si els nodes representen punts sobre la superfície terrestre, la distància Euclídea pot ser una bona aproximació.

### 4.1.3. Algoritme d'optimització de colònia de formigues

L'algoritme d'optimització de colònia de formigues es basa en les característiques d'interacció social dels animals a la natura. D'aquesta manera es pot aconseguir un objectiu global sense una entitat central que controli tots els processos.

L'algoritme imita el comportament de les formigues que surten en gran quantitat del formiguer a buscar menjar. Al principi les formigues trien un camí aleatori

dins del graf. Per comunicar-se i coordinar-se amb les altres, les formigues secreten feromones al llarg del seu recorregut. Aquestes feromones es volatilitzen a poc a poc. Les formigues acostumen a triar els camins amb una concentració més alta de feromones, però sempre hi ha una probabilitat de no triar el camí amb més concentració. Aquesta característica afavoreix l'ús de nous camins més curts que poden aparèixer en entorns dinàmics. La probabilitat p inclou un factor $\alpha$ (pes de la concentració de feromones) i una altre $\beta$ (atractiu de la ruta). La freqüència més gran de formigues la trobarem en el camí més curt perquè poden retornar més ràpidament i tornar a començar més sovint. D'aquesta manera la concentració de feromones en el camí òptim s'intensifica i la majoria de formigues l'acaben escollint i podrem deduir que hem trobat el camí més curt.

### 4.1.4. Xarxes Neuronals

El problema de resoldre el camí més curt mitjançant una xarxa neuronal artificial es concentra en la dificultat d'escollir l'estructura de nodes ocults de la ANN que vulguem implementar. Les diferents solucions que trobem acostumen a estructurar aquesta capa de manera que contingui la informació del graf que volem resoldre i d'alguna manera és un reflex parcial de la seva topologia [22].

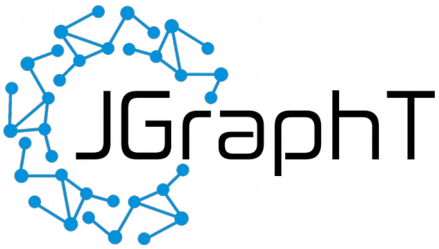
Un dels models utilitzats és la Xarxa Neuronal de Hopfield [20] que consisteix en un model recurrent que es pot utilitzar com a sistema de memòria associativa. Es defineix una funció d'energia que cal que la xarxa minimitzi mentre que la informació de la xarxa es troba guardada en els seus nodes de biaix.

Una altra solució similar és la que proposen Saeed Behzadi i Ali Aleskeikh de fer servir la xarxa directament com algoritme per trobar el millor camí. [21] Totes les neurones contenen el node inicial com a entrada i el node objectiu com a sortida. La resta de gens s'escullen aleatòriament però amb coneixement dels nodes i de les connexions del graf. A continuació cada neurona conté una proposta de solució i s'escolliran les neurones que presenten una solució amb el camí més curt. Aquestes es creuaran i mutaran fins a trobar el camí més curt.

Una altra aproximació consisteix en la creació d'un camp artificial d'energia [22] potencial (o gravitatòria) i que es fa servir més habitualment en el camp de navegació robòtica. En aquesta aproximació es crea un camp de potencial en l'entorn on es crea una força atractiva que prové del punt objectiu o destí i unes forces repulsives provinents dels obstacles.

## 4.2. BIBLIOTECA JgraphT

Per la creació i manipulació de grafs utilitzarem la biblioteca JgraphT [28] que consisteix en una biblioteca de classes Java que implementen estructures de dades i algoritmes de teoria de grafs.




*Il·lustració 13: Logotip JGraphT*

Algunes de les característiques de la biblioteca són:

- Permet fer servir qualsevol objecte com a tipus pels vèrtexs i les arestes del graf.
- Definició d'arestes dirigides, no dirigides, amb pes o sense.
- Diferents tipus de graf: complet, bipartit, multigraf, pseudograf,..
- Iteradors per recórrer el graf (DFS, BFS,..)
- Implementació de diferents algoritmes:
  - Camí més curt
  - Detecció de cicles
  - Detecció de cliques
  - Connectivitat
  - Detecció d'isomorfismes
  - Coloració

Pel nostre treball utilitzarem grafs on els nodes seran objectes de tipus enter i les arestes una classe especial de la llibreria anomenda DefaultWeightedEdge que ens permet associar un pes a cada aresta. Generarem grafs del tipus SimpleWeightedGraph.

La utilització de la llibreria ens permet crear grafs i els mètodes que necessitem per recórrer-los i manipular-los ràpidament i a més ens proporciona mètodes ja implementats d'algoritmes coneguts per trobar el camí més curt. En el nostre cas utilitzarem la implementació proporcionada per l'algoritme de Dijkstra, que a més ens permetrà comparar els resultats obtinguts amb el nostre model, i el de l'algoritme A* amb la definició d'una heurística (en aquest cas poc òptima per la forma com generem els grafs amb la inclusió d'arestes aleatòriament entre qualsevol parell de vèrtexs) consistent en la diferència entre els valors enters de cada vèrtex per tal de comparar els resultats amb els obtinguts mitjançant Dijkstra.

## 4.3. NEAT RECURSIU

En un primer moment hem fet un estudi i valoració de diferents possibilitats per tal d'aplicar una xarxa neuronal NEAT al problema de trobar el camí més curt d'un graf. Com hem explicat en l'apartat anterior les diferent solucions que han proposat els diferents autors per la resolució del problema acostumen a basar-se en xarxes neuronals amb topologies construïdes per cada graf en concret adaptant les seves capes ocultes de neurones a l'estructura del graf o a l'espai d'estats per intentar trobar la solució mitjançant una funció potencial o energia que ens permet trobar la proximitat a la solució i d'aquesta manera propagar l'error i corregir els pesos de les connexions de la xarxa.

En el nostre cas, intentem solucionar el problema del camí més curt d'un graf des d'una perspectiva diametralment oposada, ja que utilitzarem una xarxa neuronal que utilitza el mètode NEAT i una de les seves característiques principals és que s'inicia amb una estructura de neurones o topologia mínima i a partir d'aquí va evolucionant la seva estructura cap a les que presenten un millor comportament. [22]

Per tant, es poden seguir diferents camins per intentar resoldre el problema:

- L'entrada és la matrius de pesos de la xarxa i la sortida ens indica els nodes que cal recórrer. En aquest cas caldrà implementar algun mecanisme per assegurar que aquest nodes formen un camí, per exemple fent una comprovació en la matriu d'adjacències.
- L'entrada i la sortida son iguals al grau del graf del graf. Les entrades representen els possibles camins d'un node a l'altre i les sortides ens indiquen quin camí s'ha de seguir. Aquest procés ens permet resoldre el problema utilitzant-lo iterativament per a un mateix graf.
- Les entrades representen un camí aleatori i es simula un problema de colònia de formigues entrenant la xarxa per tal de formar una matriu de probabilitats amb els resultats.

En la següent taula mostrem els avantatges i desavantatges de les diferents tècniques que es podrien utilitzar per resoldre el problema:

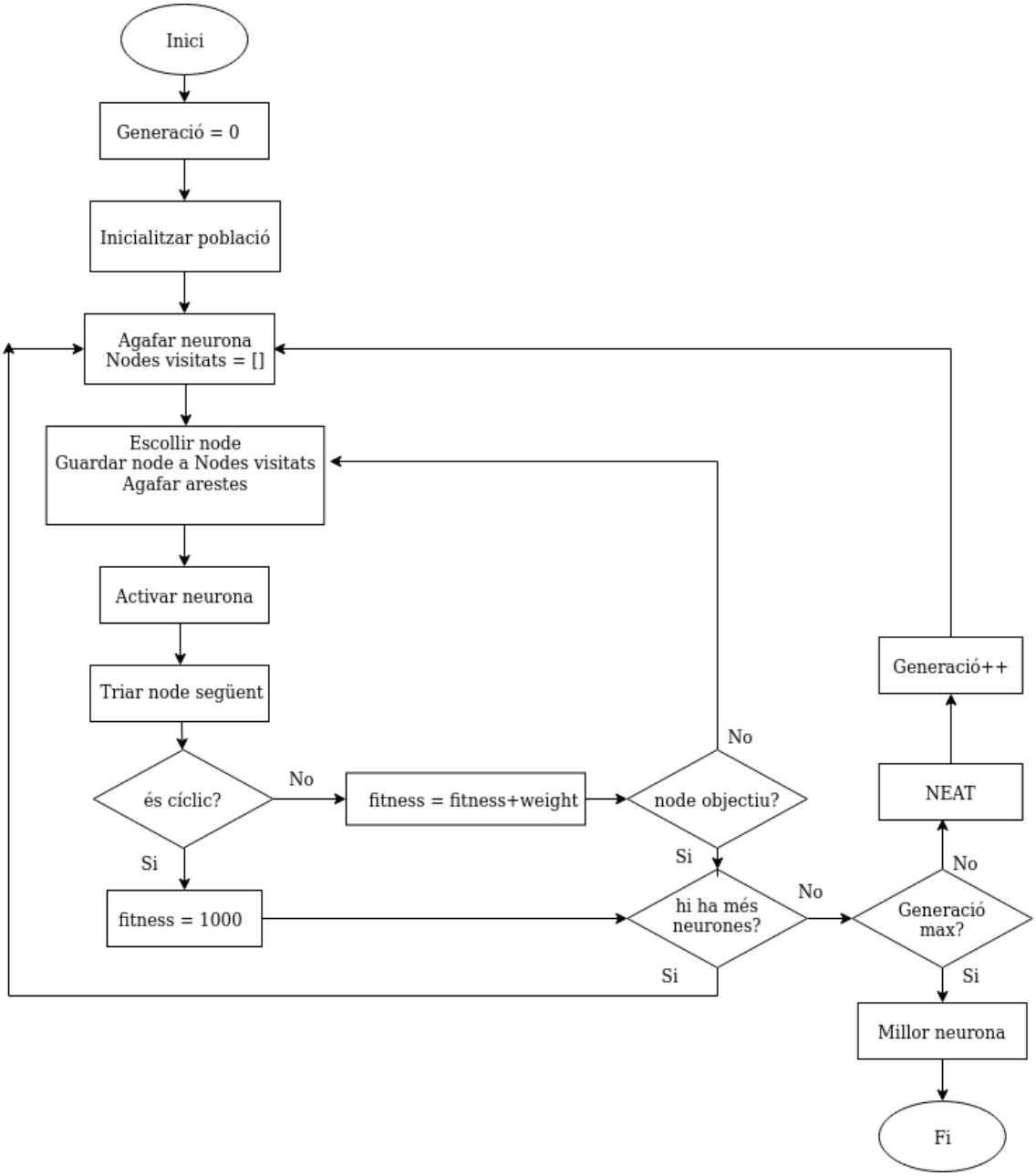
| Tècnica | Avantatges | Desavantatges |
|---|---|---|
| Matriu d'adjacències | Implementació directa amb xarxa neuronal<br>Representació compatible amb representació del graf | Mecanisme de control per assegurar que els nodes solució són un camí |
| Mètode recursiu | S'adapta a entorns dinàmics<br>Pot ser útil en sistemes multiagents | Cal modificar l'algoritme per convertir-lo en recursiu |
| Colònia de formigues | La solució serveix per tots els camins existents al graf | Dificultat per trobar el fitness de la neurona si no calculem abans els resultats. |

*Il·lustració 14: Taula comparativa tècniques*

Si fem l'anàlisi d'aquestes possibilitats, i tenint en compte que el mètode NEAT s'ha utilitzat per entrenar sistemes multiagents per simular sortides d'emergència [23], o que una de les motivacions inicials de l'àrea del treball era el camp de la conducció autònoma i la seva planificació de rutes en funció de l'estat del trànsit (carrers tallat, congestió de trànsit,..) hem optat per adaptar el mètode NEAT a un algoritme iteratiu on en cada node s'analitza la millor opció i quan s'arriba el destí es valora la seva eficiència.

Per convertir el mètode NEAT tal i com l'hem definit en el punt 2 de la memòria a un funcionament recursiu considerarem que en cada iteració ens trobem en un node del graf i que podem veure les diferents arestes que en surten (o camins que pot prendre des d'aquell node). Les entrades de la neurona seran aquestes arestes i la millor sortida serà el camí que recorrerà cap al següent node. Aquest node es repetirà fins que arribi al destí on se li donarà a la neurona el valor de fitness resultant de la suma de tots els passos recorreguts. En cas de que l'algoritme detectés un cicle finalitzaria el camí i li atorgaria el pitjor fitness possible per penalitzar la reproducció de la neurona. Amb aquests valors de fitness obtinguts després de la iteració aplicaríem el mètode NEAT, amb les mutacions i encreuaments característics del mètode, fins a completar les generacions predefinides o les condicions de finalització del mètode (no hi ha canvis en un nombre determinat de generacions).

El comportament de l'algoritme NEAT recursiu seria el següent:




*Il·lustració 15: Algoritme NEAT recursiu*

Per aplicar aquest algoritme ens cal canviar alguns mètodes de la nostra implementació de NEAT perquè s'adaptin al procés recursiu. Els mètodes modificats els inclourem en una nova classe anomenada NEAT_Recursiu que també inclourà algun mètode auxiliar que ens permetrà aplicar-lo al problema del camí més curt d'un graf:

**NEAT_Algorithm(ShortestPathNEAT shortestPath)**: seqüència de passos de l'algoritme NEAT que fa servir les versions recursives dels mètodes de l'algoritme.

**train(ShortestPathNEAT sPN)**: mètode que s'encarrega d'iterar sobre tots els genomes de la població i els entrena utilitzant el mètode recursiu.

**trainRecursive(ShortestPathNEAT sPN, Genome activatingGenome)**: és l'encarregat d'executar el procés recursiu i mantenir un registre del camí provisional actual fins arribar al destí o detectar un cicle en el camí provisional. Assigna el fitness a la neurona.

**predictRecursive(ShortestPathNEAT sPN, Genome activatingGenome)**: és l'encarregat d'executar el procés recursiu i mantenir un registre del camí provisional actual fins arribar al destí o detectar un cicle en el camí provisional i retornar-lo per tal de predir el millor camí.

**activateGenome(Genome genome, ArrayList<Double> inputs)**: mètode per trobar la sortida d'una neurona a partir d'un ArrayList d'entrades per una iteració en concret.

**activateNode(NodeGene node, ArrayList<Double> inputs, ArrayList<NodeGene> visited)**: mètode recursiu per calcular el valor d'activació d'un node.

**isCyclic(Integer node, ArrayList<Integer> visited)**: mètode auxiliar per detectar cicles en un camí provisional.

**bestGenome()**: recorre tots els genomes per trobar quin té el millor valor de fitness.

Al package uoc.graph hi afegim una classe AstarHeuristic que implementa una heurística per aplicar amb el mètode A* que consisteix en la diferència entre el valor de la id dels dos nodes. A més a més també hem creat una classe ShortestPathNEAT que seria l'equivalent a la classe Experiment de NEAT i que representa el cas sobre el que volem aplicar el mètode recursiu. En els seus atributs hi trobem el graf que volem estudiar, i els nodes inicial i final del camí que busquem. Els seus mètodes són:

**getMaxGrade()**: ens proporciona el grau màxim del graf i ens servirà per conèixer les entrades i sortides que necessitaran les neurones per ser capaces de resoldre tots els casos en tots el nodes.

**getWeightEdges(Integer vertex)**: ens proporciona els pesos de totes les arestes d'un node i ens servirà com a entrades per les neurones.

**createNEATPopulation()**: crea la població inicial de neurones amb les entrades i sortides necessàries per tractar el graf del problema que volem resoldre. També comprova que els nodes inicial i final es trobin dins del graf.

## 4.4. RESULTATS

El primer experiment per la utilització de la implementació del mètode NEAT recursiu per a trobar el camí més curt d'un graf serà utilitzant un graf de 10 nodes i connexions aleatòries i l'entrenem per trobar el camí entre dos nodes escollits aleatòriament. A la primera prova que fem ens trobem que més del 90% dels tests ens donen resultats erronis. Analitzem els camins escollits per la millor xarxa neuronal que hem obtingut entrenant la població amb el nostre mètode i ens trobem que moltes vegades després d'escollir un camí (o aresta) i arribar a un nou vèrtex, en aplicar de nou la nostra xarxa neuronal amb totes les arestes que hi ha connectades a aquell vèrtex, ens torna a escollir l'aresta per la qual ha vingut i retorna al vèrtex anterior fent que el procés finalitzi en detectar un cicle. Modifiquem la implementació per evitar que la neurona pugui triar un camí que retorni pel camí pel qual ha vingut.

Tornem a realitzar l'experiment i ens trobem amb els següents resultats. Hem de tenir en compte que no s'ha implementat cap mecanisme per detectar que no hi ha cap camí en el cas de grafs no connectats, per tant els considerarem falsos negatius. En el cas de que la xarxa trobi un camí però no es correspongui amb el camí més curt ho considerarem com un fals positiu. Utilitzarem grafs de diferents mides per comprovar si hi ha diferències. Efectuarem 100 proves amb cada tipus de graf per obtenir els percentatges directament.

| Resultats camí més curt d'un graf amb NEAT Recursiu | | | | | |
|---|---|---|---|---|---|
| Nombre nodes | Grau màxim | Positius | Negatius | Falsos positius | Falsos negatius |
| 10 | 3 | 45 | 25 | 5 | 25 |
| 10 | 8 | 13 | 63 | 24 | 0 |
| 30 | 3 | 0 | 78 | 9 | 13 |
| 30 | 25 | 0 | 100 | 0 | 0 |

*Il·lustració 16: Taula resultat camí més curt per diferents grafs*

Podem veure que si apliquem la xarxa neuronal a un graf petit, 10 nodes i grau màxim 3, troba el camí més curt en un percentatge força alt de casos d'un 45%, i que aquest incrementa arriba al 70% si hi afegim els casos de falsos negatius degut a que el graf és partit i el camí no existeix. A més a més cal tenir en compte que els falsos positius ens proporcionen camins que tot i no ser el més curt, s'hi aproxima bastant.

En canvi, a mida que augmentem la mida del graf, ja sigui incrementant el nombre de connexions entre vèrtexs (grau) o el seu nombre de nodes, els resultats empitjoren ràpidament.

Per tant, tot i que veiem que l'algoritme és capaç de trobar el camí més curt en alguns casos, deixa de ser eficient quan el problema es torna més complex. Si

revisem els camins que ens tria l'algoritme en els errors podem veure que hi ha una tendència a escollir més sovint els nodes situats en les primeres posicions de la llista d'entrades. Per resoldre aquest tema caldria revisar la implementació. Com que en els vèrtexs on el seu grau és més petit que el grau màxim del graf hem omplert les entrades que ens sobraven amb un pes molt elevat per penalitzar el seu valor de fitness, i aquestes entrades buides sempre les hem deixat al final de l'ArrayList d'entrades, les nostres xarxes tenen tendència a escollir les primeres posicions de l'ArrayList d'inputs. Caldria estudiar com tractem els inputs que ens sobren quan ens trobem en un node que no té el grau màxim de la neurona. Un altre mecanisme que s'hauria de revisar és la manera com calculem el fitness de cada xarxa, sobretot en els casos que ens trobem amb un cicle.

Finalment també cal comentar que tal i com hem implementat el nostre algoritme recursiu per resoldre el problema, tots els errors es deuen a que la xarxa neuronal s'ha trobat amb un cicle i s'ha aturat ja que és evident que aquell no és el camí més curt, però es podria adaptar l'algoritme per tal de que continués provant fins a arribar al vèrtex objectiu.

# 5. Conclusions

En aquest TFG hem aconseguit una implementació d'una xarxa neuronal basada en el mètode NEAT i l'hem adaptat per aplicar-lo recursivament a un graf per trobar el camí més curt entre dos vèrtexs del graf.

Ens hem trobat que la implementació d'una xarxa neuronal és una tasca complexa, especialment pel maneig de les diferents estructures de dades que implementen la representació de la xarxa i la dificultat de gestionar l'activació de les neurones i de definir el seu fitness i de propagar-lo per la xarxa.

Pel que fa al mètode NEAT hem après la capacitat d'un sistema neuroevolutiu d'adaptar-se al problema, amb un interès especial per NEAT que ho fa a partir d'una xarxa amb topologia mínima, amb l'estalvi que representa no haver de fer proves i assajos per determinar la topologia de la xarxa que es necessita en una xarxa amb topologia fixa.

També hem après la dificultat per trobar una solució aproximada a un problema NP-Difícil com el de trobar el camí més curt entre dos punts d'un graf. Hem vist que tot i que hi ha diversos algoritmes que intenten resoldre el problema, representen aproximacions diferents al problema que es centren en característiques diferents del problema.

També hem vist que la transformació d'una xarxa neuronal a un algoritme recursiu comporta algunes dificultats d'implementació i que per tal d'adaptar-la al problema del camí més curt d'un graf es necessita modificar la implementació i afegir en alguns mètodes algunes ajudes, en podríem dir heurístiques senzilles, que l'ajudin per trobar el millor camí. Finalment hem comprovat que per a grafs més complexos la xarxa que hem creat és massa simple i no aconsegueix trobar les solucions correctes.

Els objectius inicials del treball s'han assolit gairebé la majoria. En el capítol 2 trobem resultat de l'estudi de les xarxes neuronals i de l'estudi de les característiques i funcionament de la xarxa neuronal basada en el mètode NEAT.

La implementació del mètode NEAT ha estat una de les tasques principals d'aquest TFG i que tot i ser una tasca més feixuga del que s'havia previst, podem dir que hem aconseguit implementar-la, fer els tests i provar que funciona amb un experiment simple. El resum de les decisions preses en el disseny del codi i de les classes el podem trobar en el capítol 3 de l'informe del Treball de Final de Grau.

L'objectiu d'utilitzar el mètode NEAT per resoldre el problema de trobar el camí més curt d'un graf ha estat un repte inesperat degut a la necessitat de trobar i dissenyar una solució específica. Per aquest motiu, l'estudi i anàlisis de les diferents possibilitats de resolució ha ocupat un temps que s'ha allargat més del que s'havia previst i ha concentrat tots els esforços de la fase final del treball.

Podem trobar els resultats al capítol 4 de la memòria així com la solució i detalls de la implementació realitzada.

El darrer objectiu, la comparació dels resultats obtinguts amb la xarxa NEAT respecte dels obtinguts amb altres tipus de xarxa neuronal, no ha estat possible principalment per la no existència d'implementacions de xarxa neuronal que siguin fàcils d'utilitzar per resoldre aquest problema i simplement ens hem centrar a comparar els resultats amb els que ens proporciona l'algoritme de Dijkstra.

Durant la realització del Treball hi ha hagut algunes desviacions sobre la planificació inicial, principalment degudes a la dificultat tant d'implementar la xarxa en Java des de zero en la primera fase com de trobar i adaptar la implementació a un logaritme que sigui capaç de trobar el camí més curt en la segona fase. Les dues entregues representaven productes que presentaven molts aspectes amb molt camí per fer-hi millores i amb la motivació per presentar el millor resultat possible, la feina de redacció de l'informe i documentació de la implementació s'ha retardat en els terminis d'entrega.

També crec que en aquest TFG es barrejaven dos temes molt complexos i que els terminis que hi havien a la planificació eren massa curts per aprofundir-hi més. Aquest és el motiu pel qual no s'ha pogut treballar en els objectius secundaris i s'han hagut d'utilitzar les accions de mitigació d'utilitzar el temps que s'havia reservat per a imprevistos per acabar les implementacions dels algoritmes treballats.

En el futur, es presenten camins molt interessants per continuar amb la tasca iniciada amb aquest TFG. Per una banda, la implementació del mètode NEAT es podria generalitzar per tal de que els arguments del mètode main() siguin les dades que es necessiten (llistat d'inputs i outputs) per definir l'experiment. També es podria refinar la implementació per tal de que el funcionament d'alguns mètodes s'adaptin millor a les funcionalitats descrites pel mètode Neat, especialment en el control del valor de fitness i del nombre de generacions sense canvis en aquest valor i en alguns detalls de com s'ha implementat la classificació en espècies i de la reproducció de genomes (evitar l'encreuament de genomes amb si mateix,..).

Pel que fa a l'estudi del camí més curt, per una banda es podria fer l'estudi de com s'han d'implementar i quins resultats ens donen les tècniques descartades: utilitzar matrius d'adjacències o colònies de formigues, que gairebé seria un tema per un TFG cada una per separat. Per altra banda, el mètode NEAT Recursiu es podria continuar millorant, especialment pel que fa al seu comportament quan detecta un cicle i en la manera com es calcula el fitness de la xarxa, per tal de poder millorar els resultat en grafs de mida més gran.

# 6. Glossari

**especiació**: mètode que separa els diferents genomes en grups en funció de la seva distància.

**funció d'activació**: funció que calcula el valor d'activació d'una neurona a partir del valor de la suma de totes les entrades de la neurona.

**genoma**: conjunt de gens que defineixen l'estructura d'una xarxa neuronal.

**gens**: cada un dels elements que formen un element de la topologia d'una xarxa.

**hidden**: neurones d'una xarxa que es troben entre la capa d'entrada i de sortida.

**inputs**: neurones d'una xarxa que reben les senyals de fonts externes de la xarxa.

**mutació**: canvis en els paràmetres (canvis en els pesos de les connexions) o en l'estructura de la topologia (afegir nodes o connexions) d'una xarxa neuronal.

**NEAT**: Acrònim de NeuroEvolution of Augmented Topologies. Mètode creat per Stanley i Mikkulainen per crear xarxes neuronals neuroevolutives.

**neuroevolució**: capacitat de les xarxes de variar els seus paràmetres o la seva topologia.

**neurona**: cada unitat de processament o node de la xarxa.

**outputs**: neurones d'una xarxa que emeten senyals a l'exterior.

**reproducció**: creació de noves xarxes neuronals combinant gens de diferents xarxes neuronals.

**topologia de xarxa**: conjunt de nodes i connexions entre aquests que ens descriuen com és la xarxa. S'acostuma a representar mitjançant un graf.

**xarxa neuronal**: simulació del funcionament del cervell humà consistent en un conjunt de neurones que es poden utilitzar i entrenar per resoldre problemes.

# 7. Bibliografia

[1] Stanley, Kenneth O. and Miikkulainen, Risto. *Evolving Neural Networks Through Augmenting Topologies*. In: Evolutionary Computation 10(2): 99-127, 2002.

[2] Stanley, Kenneth O. and Miikkulainen, Risto. *Efficient Evolution of Neural Network Topologies*. In: Proceedings of the 2002 Congress on Evolutionary Computation,
CEC2002, 2002.

[3] Wikipedia: https://ca.wikipedia.org/wiki/Xarxa_neuronal_artificial. Visitada el 15/10/2019

[4] Materials UPC: https://upcommons.upc.edu/bitstream/handle/2099.1 6483 05.pdf ?sequence=6&isAllowed=y *. Visitada el 18/10/2019*

[5] Materials UPC: https://upcommons.upc.edu/bitstream/handle/ 2099.1/ 5600/06.pdf ?sequence=7&isAllowed=y . Visitada el 18/10/2019

[6] *Stanford Artificial Intelligence Laboratory:* http://www.cs.stanford.edu/ research/ai. Visitada el 20/10/2019

[7] Evolutionary Complexity (EPlex) Research Group at the University of Central Florida. *Find the Right Version of NEAT for Your Needs:* http://eplex.cs. ucf.edu/neat_software. Visitada el 20/10/2019

[8] Neat-c: http://nn.cs.utexas.edu/?neat-c. Visitada el 20/10/2019

[9] SharpNEAT: https://sharpneat.sourceforge.io. Visitada el 20/10/2019

[10] Jneat: http://nn.cs.utexas.edu/soft-view.php?SoftID=5. Visitada el 20/10/2019

[11] Anji: http://anji.sourceforge.net. Visitada el 20/10/2019

[12] Neat4J: http://neat4j.sourceforge.net . Visitada el 20/10/2019

[13] Ahni: https://github.com/OliverColeman/ahni. Visitada el 20/10/2019

[14] L.Cardamone, D.Loiacono i P.L.Luca. *Evolving competitive car controllers for racing games with neuroevolution.* In: GECCO'09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. Juliol 2009. Pàgines 1179-1186

[15] R.Liang i P.Zhao. *Applying and Comparing Evolutionary Algorithms for Robot Tanks*. In: Swarthmore College papers.

[16] J.Gomes, P.Urbano i A.L.Christensen. *Evolution of Swarm Robotics Systems with novelty* In: Search. Arxiv:1304.3362v1. 11 d'abril de 2013

[17] Aavaas Gajurel. *Neuroevolution for Realtime Strategy Game Micro management.* In: Thesis of Master of Science in Computer Science and Engineering. University of Nevada, Reno.

[18] Forum Toribash game: http://forum.toribash.com/showthread.php?t=10899. Visitada el 24/10/2019

[19] Christoph Oberndorfer. *Research on new Artificial Intelligence based Path Planning Algorithms with Focus on Autonomous Driving* In: (Master Thesis). Munich University of Applied Sciences.

[20] Esmaeili Aliabadi, Danial. *Modeling Shortest Path Routing Problem with Hopfield Neural Network.* In: 10.13140/RG.2.1.3133.2960. 2015/07/13

[21] Seed Behzadi and Ali A. Alesheikh. *Developing a Genetic Algorithm for Solving Shortest Path Problem.* In: WSEAS International Conference on urban planning and transportation (UPT'07), Heraklion, Crete Island, Greece, July 22-24

[22] Tianyu Zhou. *Deep Learning Models for Route Planning in Road Networks.* In: Degree Project Computer Science and Engineering. University of Stockholm, Sweden 2018

[23] Mehmet Erkan Yuksel. A*gent-based evacuation modeling with multiple exits using NeuroEvolution of Augmenting Topologies.* In: Advanced Engineering Informatics 35 (2018) 30–55.

[24] Danilo Vasconcellos Vargas and Junichi Murata. *Spectrum-Diverse Neuroevolution With Unified Neural Models.* In: IEEE transations on neural networks and learning systems, *Vol. 28, no. 8, August 2017*.

[25] João Nadkarni, Rui Ferreira Neves. *Combining NeuroEvolution and Principal Component Analysis to trade in the financial markets.* In: Expert Systems With Applications 103 (2018) 184–195.

[26] Neuroevolution of augmenting topologies: https://en.wikipedia.org/wiki/Neuroevolution_of_augmenting_topologies. Visitada el 19/10/2019

[27] NEAT: An Awesome Approach to NeuroEvolution: https://towardsdatascience.com/neat-an-awesome-approach-to-neuroevolution-3eca5cc7930f. Visitada el 19/10/2019

[28] JGraphNeat: https://jgrapht.org. Visitada el 20/11/2019

[29] Jack M. Haglin, Genesis Jimenez, Adam E. M. Eltorai. *Artificial neural networks in medicine*. In: Health and Technology (2019) 9:1-6

[30] Fatemeh Salehi Rizi, Joerg Schloetterer, Michael Granitzer. *ShortestPath Distance Aproximation using Deep Learning Techniques*. In: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysisi and Mining. pp 1007-1014