CAMÍ MÉS CURT UTILITZANT INTEL·LIGÈNCIA D'EIXAM

Llorenç GELABERT MARÍ
Màster de Bioinformàtica i Bioestadística
Intel·ligència Artificial

Tutor: Dr. Brian JIMÉNEZ GARCÍA
PRA: Dra. Maria Jesús MARCO GALINDO

30 de juny de 2018

Copyright © 2018 Llorenç Gelabert Marí.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

Fitxa del treball final

Títol del treball:	Camí més curt utilitzant intel·ligència d'eixam
Nom de l'autor:	Llorenç Gelabert Marí
Nom del consultor/a::	Dr. Brian Jiménez García
Nom del PRA:	Dra. Maria Jesús Marco Galindo
Data d'entrega (mm/aaaa):	06/2018
Titulació:	Màster de Bioinformàtica i Bioestadística
Àrea del Treball Final:	Intel·ligència artificial
Idioma del treball:	Català
Paraules clau:	swarm intelligence, shortest path, decentralized system

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

L'objectiu final d'aquest Treball Final de Màster (TFM) serà el d'obtenir un algoritme que, a partir d'un graf i dos nodes, origen i destí, sigui capaç de trobar una ruta amb pes mínim que els uneixi.

Per aconseguir-ho serà necessari provar múltiples configuracions per als diferents paràmetres d'entrada del nou algoritme implementat. Així, una bona conjunció entre una bona definició i implementació de l'algoritme i la configuració utilitzada ens conduirà a completar el projecte de forma satisfactòria.

També necessitarem generar, o obtenir d'altres projectes, la definició d'un o més grafs sobre els quals aplicar l'algoritme.

Pel fet que el nostre algoritme no és determinista, sinó que es basa en probabilitats, no s'espera obtenir sempre la millor ruta, de pes mínim, com ho faria un algoritme determinista com el de Dijkstra.

El nostre algoritme té alguns avantatges sobre aquests algoritmes, i és l'adaptabilitat o facilitat per a reconstruir una nova millor ruta en entorns canviants sense necessitat de començar el càlcul dels camins des de l'inici.

S'han aconseguit un conjunt de configuracions que presenten bons resultats. Els seus camins assolits es troben al voltant d'un 10% per sobre del valor de la millor ruta.

Donat que el nostre algoritme pot aplicar-se sobre qualsevol tipus de graf, és molt complicat obtenir una configuració que funcioni sempre. Algunes configuracions tenen un bon comportament amb grafs amb un gran nombre d'arestes, i d'altres amb grafs amb molts de cicles.

Abstract (in English, 250 words or less):

The final objective of this Master's Final Project is to obtain an algorithm, based on a graph and two nodes, origin and destination, which can find a route with a minimum weight that connects them.

To achieve this, it will be necessary to test multiple configurations for the different input parameters of the new implemented algorithm. Thus, a good combination between a good definition and implementation of the algorithm and the configuration used will lead us to successfully complete the project.

We will also need to generate, or obtain from other projects, the definition of one or more graphs on which to apply the algorithm.

Because our algorithm is not deterministic, but based on probabilities, it is not always expected to get the best, least weighted route, as a deterministic algorithm like Dijkstra's algorithm would do.

Our algorithm has some advantages over these algorithms, and it is adaptability or the capacity to rebuild a new best route in changing environments without having to start the path calculation from the beginning.

A set of successful configurations has been achieved. Their achieved paths are around 10% above the value of the best route.

Since our algorithm can be applied to any type of graph, it is very difficult to get a configuration that always works. Some configurations have a good behavior with graphs with a large number of edges, and others with graphs with many cycles.

Al Màster he d'agrair l'oportunitat d'haver-te conegut. En poc temps has passat a ser algú important per a mí. Durant aquest mesos hem compartit moltes hores d'estudi, i ens hem robat hores de son. Fer-ho amb tu ha sigut un plaer, has convertit una cosa aparentment avorrida en desitjada. No hi ha res clar del que passarà, però passi el que passi, gràcies per tots els moments i somriures.

Que res acabi. Que tot comenci.

Sumari

Índex de figures

Índex de taules

1 Introducció

A l'hora de cercar un projecte per al TFM no tenia una idea concreta sobre el tema sinó sobre quina branca volia tractar: la Intel·ligència Artificial (IA). És un tema que sempre m'ha interessat. Al grau d'Enginyeria Informàtica vaig realitzar el Treball Final de Màster (TFG) sobre el reconeixement del color de pell en imatges utilitzant màquina de vector de suport (SVM), que són una branca de la IA.

Així, després de llegir les diferents propostes em semblà que la que més s'adequava als meus desitjos era la proposta de realitzar un algoritme utilitzant Intel·ligència d'Eixam (IE) per a resoldre el problema del camí més curt entre dos nodes d'un graf.

Aquest treball consta de tres grans parts per tal de que s'adeqüi als requisits del Màster:

- Formació: etapa d'estudi sobre IE i tot el que envolta el problema del camí més curt entre dos nodes.
- Programació: desenvolupament i implementació de l'algoritme.
- Anàlisi: aplicació dels diferents coneixements estadístics assolits al Màster per tal d'analitzar els resultats obtinguts amb l'execució de l'algoritme.

Finalment, hi ha multitud de problemes que es presenten en biologia que requereixen d'un algoritme que obtingui la ruta més curta entre dos nodes, com poden ser: interacció entre proteïnes, xarxa de regulació gènica, etc.

Així, el projecte resulta adequat com a TFM, ja que cobreix les diferents competències a assolir.

1.1 Context i justificació del treball

L'objectiu principal del TFM és implementar un nou algoritme que resolgui el problema del camí més curt entre dos nodes d'un graf [1] utilitzant tècniques d'IE [1]. Per aconseguir-ho, serà necessari primer estudiar el concepte d'IE i alguns exemples de problemes ja resolts. A continuació, s'implementarà un nou algoritme utilitzant els coneixements adquirits. Finalment, aquest algoritme es compararà amb d'altres ja coneguts per tal d'obtenir la seva eficiència. A més, amb l'anàlisi dels resultats també s'espera poder definir en quins sistemes o situacions són millors els algoritmes descentralitzats.

[1] El **camí més curt** és aquell on la suma dels pesos de les arestes que el forma és mínima.

1.1.1 Definició del problema

El problema a resoldre és una reducció del problema del viatjant [2], que cerca connectar tots els nodes d'un graf amb un pes mínim. En el nostre cas, ens centrarem a solucionar-ho per a trobar el camí més curt donats dos nodes [2].

Sembla un problema senzill, però quan el nombre de nodes i enllaços entre aquests augmenta, pot resultar molt lent si s'utilitzen algoritmes deterministes. A més, en entorns on l'estructura del graf pot variar, els algoritmes deterministes han de recalcular gran quantitat d'informació, o començar de nou. En canvi, amb la nostra proposta els agents deixen informació per a tot l'entorn, sent capaços de substituir un camí per un altre amb relativa facilitat.

El nostre algoritme serà estocàstic, fet que no ens assegura obtenir la millor solució, però si s'espera una bona solució en un temps assumible.

El nostre objectiu és implementar un nou algoritme basat en la IE i els principis de l'autoorganització.

1.1.2 Justificació del treball

La IA [3] és una ciència molt popular els darrers temps. Les millores en el software, l'aparició de grans clústers, i la gran quantitat d'informació, permeten aplicar algoritmes d'IA que abans eren massa lents o poc efectius. Així, molts problemes es poden enfocar des d'una perspectiva diferent per tal d'obtenir una solució, o millorar la ja existent utilitzant altres metodologies.

Encara que el concepte de *swarm intelligence* té dècades (finals dels anys 80) [4, p.7], es troba en evolució contínua. En molts casos, els algoritmes que implementa es basen en l'observació del comportament d'alguns animals: camí seguit per les formigues per a l'obtenció d'aliments, construcció dels ruscs d'abelles, etc. S'ha comprovat que aquests ofereixen solucions imaginatives i, en molts casos, donat un problema com la selecció d'una ruta, s'obté un bon rati entre temps de càlcul i solució obtinguda.

Finalment, el problema del camí més curt és un problema típic en informàtica: ruta entre dos nodes d'una xarxa, joc de paraules [5], xarxes d'interacció proteïna-proteïna, etc. A més, amb el fet d'utilitzar la IE és cerca solucionar el problema aplicant coneixements biològics, estudiant el comportament de diferents grups d'insectes, i estadístics, utilitzats amb la selecció de decisions.

[2] El **camí més curt** és aquell on la suma dels pesos de les arestes que el forma és mínima.

1.2 Objectius

1.2.1 Generals

Ja es disposa d'un algoritme base sobre el qual treballar. A partir d'aquí, els objectius generals són:

- **Implementació** de diferents versions sobre l'algoritme base: és l'objectiu principal del projecte, ja que ens permetrà mitjançant l'estudi dels resultats conèixer els efectes de cada petita modificació.
- ***Benchmarking***. S'implementarà un algoritme determinista, com és Dijkstra [6], amb el qual obtindrem la millor solució a l'hora de calcular el camí més curt i ens permetrà saber quant s'hi apropa la solució proporcionada amb el nostre algoritme. En molts casos, no només és important obtenir la millor solució sinó fer-ho en un temps i consum de recursos mínims. Per això, s'hauran de definir mètriques per a tenir-ho en compte.
- **Anàlisi** de resultats i conclusions. És un punt fonamental del projecte que ens permetrà entendre a què es deuen els resultats obtinguts i definir possibles millores. S'ha de tenir en compte que les diferents versions de l'algoritme base poden obtenir millors resultats sobre un tipus concret de configuració, com ara un graf amb gran quantitat de cicles.

Addicionalment, s'implementarà una interfície gràfica amb la qual es pugui visualitzar el funcionament de l'algoritme.

1.2.2 Específics

- Implementació.
  - Definir diferents versions de l'algoritme base.
- *Benchmarking*.
  - Implementar Dijkstra.
  - Definir les diferents mètriques.
- *Anàlisi*.
  - Obtenir els resultats.
  - Analitzar els resultats obtinguts.
  - Definir les mancances i possibles millors del nostre algoritme.

1.3 Metodologia

S'aplicarà la metodologia àgil que ens permet modificar el producte al llarg de tot el projecte. Així, en qualsevol moment es poden corregir errors o aplicar millores que hagin sorgit en tasques posteriors. Amb això s'aconsegueix, per exemple, que si a l'hora de realitzar el *benchmarking* es detecten possibles correccions o millores en l'algoritme proposat, aquestes siguin aplicades.

El cronograma serà una guia per a facilitar l'organització del projecte, que a més ens permetrà controlar el temps de desenvolupament, però mai serà un impediment en cas de ser necessari tornar a una etapa anterior.

Com s'ha pogut comprovar durant la realització del projecte, ha sigut necessari poder redefinir, o ampliar, el temps de les tasques.

1.4 Planificació

1.4.1 Tasques

Desenvolupament.

- Descripció del model natural.
- Revisió de l'algoritme base.
- Revisió de la implementació de graf.
- Implementació de les diferents versions sobre l'algoritme base.

***Benchmarking*.**

- Estudi i implementació de Dijkstra.
- Definició de les mètriques.
- Codificació de les mètriques.

Execució i obtenció de resultats.

- Implementació d'un algoritme de creació automàtica de grafs.
- Definició de diferents configuracions de grafs: nombre de nodes i arestes, dispersos i densos, amb gran nombre de cicles, etc.
- Càrrega de grafs típics sobre els quals realitzar proves.
- Execució i obtenció de resultats.

Anàlisi.

- Estudi dels resultats obtinguts.

- Conclusions.

Visualització i aplicació.

- Implementació d'una interfície gràfica que permeti mostrar el funcionament de l'algoritme.
- Aplicació de l'algoritme sobre un problema real.

Documentació.

- Redacció de la memòria.
- Elaboració de la presentació.

L'assoliment de les tasques anteriors són el mínim suficient per a completar satisfactòriament el projecte. Pel que fa a l'apartat **Visualització i aplicació**, és un objectiu addicional que cerca mostrar el funcionament i una aplicació real que pugui tenir el nostre algoritme.

1.4.2 Fites

Per a cada part del projecte es crea una fita d'inici i una de final d'aquella part. Així, a la taula 1 podem veure un resum de les fites que s'han definit en el projecte.

Fita	Data inici	Data fi
Projecte	26/03/2018	13/06/2018
Desenvolupament	26/03/2018	16/04/2018
Benchmarking	17/04/2018	23/04/2018
Entrega fase 1	-	23/04/2018
Execució i obtenció de resultats	24/04/2018	30/04/2018
Anàlisi	03/05/2018	09/05/2018
Visualització i aplicació	10/05/2018	18/05/2018
Entrega fase 2	-	21/05/2018
Documentació	22/05/2018	13/06/2018

Taula 1: Resum de les fites que formen el projecte.

1.4.3 Calendari

A la taula 2 podem veure un resum del calendari amb les diferents etapes, tasques, i les hores de dedicació. Els nombres són els referenciats a la figura 1, en la que podem veure el diagrama de Gantt resultant.

Etapa	Tasca	Hores	Hores fita	Hores totals
1. Desenvolupament	1.1	5	95	335
	1.2	20		
	1.3	10		
	1.4	60		
2. Benchmarking	2.1	5	30	
	2.2	10		
	2.3	15		
3. Obtenció resultats	3.1	15	40	
	3.2	7.5		
	3.3	2.5		
	3.4	15		
4. Anàlisi	4.1	22.5	30	
	4.2	7.5		
5. Visualització i aplicació	5.1	30	40	
	5.2	10		
7. Documentació	7.1	65	100	
	7.2	35		

Taula 2: Resum de les tasques que formen el projecte.

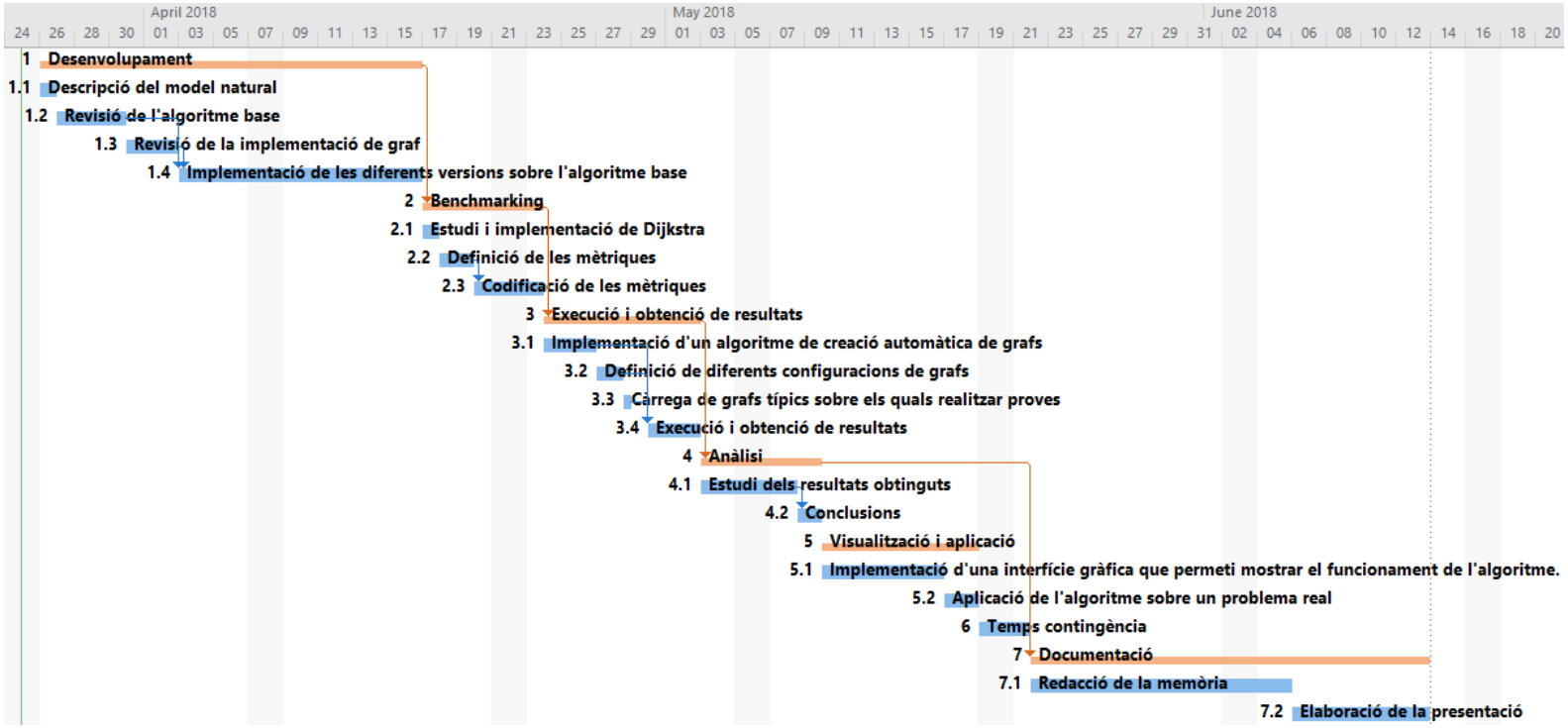


Figura 1: Diagrama de Gantt resultant després de la planificació de les tasques.

1.4.4 Anàlisi de riscs

Els riscs principals són dos:

- Risc crític: no aconseguir desenvolupar diferents versions funcionals del nostre algoritme base, ja que és el nucli del projecte i totes les tasques es basen en aquesta.
- Pel que fa a la resta de tasques, l'únic problema que pot sorgir és una mala planificació temporal.

Per intentar pal·liar els riscs, la planificació s'ha realitzat de forma que hi hagi dos dies, deu hores, com a temps de contingència. A més, les tasques de **Visualització i aplicació** són addicionals, i es podrien utilitzar les quaranta hores que s'han planificat per aquestes tasques en acabar les tasques fonamentals.

Finalment, les tasques de **Documentació** s'han sobredimensionat en nombre d'hores.

1.5 Productes obtinguts

Els productes obtinguts al final del projecte són:

- **Memòria**: document final que recull tot el procés de desenvolupament del projecte.
- **Codi font** que es divideix en dues parts:
  - Codi del projecte, això és l'algoritme i les seves diferents versions, i la definició d'un graf.
  - *Scripts* per tal de poder realitzar l'anàlisi dels resultats obtinguts amb les diferents execucions de l'algoritme.
- **Presentació** amb la qual mostrar el projecte realitzat.

1.6 Estructura de la memòria

La memòria es troba estructurada en els següents capítols:

- **Capítol 1: Introducció.** Definició del problema a resoldre i gestió del projecte.
- **Capítol 2: Introducció al tema**. Visió general teòrica sobre els temes que impliquen el nostre problema, IA i IE.

- **Capítol 3: Planificació i seguiment.** Cronograma de tasques i com s'han assolit les diferents etapes del projecte.
- **Capítol 4: Arquitectura del sistema.** Tecnologies utilitzades i estructura del programa.
- **Capítol 5: Desenvolupament de l'algoritme.** Explicació detallada del nou algoritme. Etapes que s'han superat per tal d'arribar a la versió final.
- **Capítol 6: Resultats.** Mostra i anàlisi dels resultats obtinguts.
- **Capítol 7: Conclusions.** Conclusions extretes amb la realització del projecte i treball futur.

2 Estudi de l'àrea

2.1 Què és la Intel·ligència Artificial

Per a definir que s'entén per IA en computació, primer serà necessari definir què és la intel·ligència.

La intel·ligència és l'habilitat de percebre informació, retenir-la, transformar-la en coneixements i, finalment, utilitzar-la en diferents entorns adaptant-la a possibles canvis [7].

La IA és una branca de la computació. Les diferents tècniques serveixen per a resoldre tot tipus de problemes, i el seu objectiu és el de crear màquines capaces de realitzar tasques normalment realitzades per persones [8].

En aquest punt, podem distingir dos tipus de IA [9]:

- **IA forta:** és la capacitat d'una màquina de realitzar qualsevol tasca de la mateixa manera que ho faria una persona. La màquina ha de ser capaç de: raonar, resoldre problemes, aprendre, comunicar-se, etc. És una definició teòrica, la seva aplicació pràctica encara no s'ha aconseguit ja que simular la cognició humana és un problema molt difícil de resoldre.

- **IA dèbil:** és el camp on s'està treballant més actualment. En aquest cas, l'objectiu no és actuar obligatòriament igual que una persona, sinó aconseguir realitzar una tasca específica. Alguns exemples:

  - **Watson:** és un sistema de IA capaç de respondre preguntes realitzades en llenguatge natural. Combina les tècniques de cerca de patrons a gran quantitat de texts amb la d'assignació de pesos als patrons trobats.
  - **AlphaGo:** és un sistema de IA capaç de jugar al joc Go. Internament fa ús de xarxes neuronals artificials per a dues tasques ben diferenciades: seleccionar l'estratègia correcta i avaluar la posició actual.
  - **Camí més curt:** utilització de la tècnica d'optimització per eixam de partícules (PSO) per a trobar la ruta més curta entre dos punts.

Un altre punt de vista d'aquest mateix concepte és:

- La IA dèbil utilitza models del domini del problema donats pels enginyers.

- La IA forta genera els seus propis models basats en la informació sense tractar que obté. No necessita de cap instrucció dels programadors.

2.2 Intel·ligència d'eixam

La IE és una branca de la IA que estudia el comportament col·lectiu dels sistemes descentralitzats, autoorganitzats, naturals o artificials [1].

Els sistemes descentralitzats tenen un punt fort, i és que són resistents a fallada. Un agent pot fallar i el sistema segueix funcionant correctament. Això és degut a que cada agent pren les decisions individualment a partir d'informació del seu entorn. Així, per a que el sistema entri en fallida han de fallar un elevat nombre d'agents.

La IE es basa en el funcionament d'alguns sistemes biològics. Aquests sistemes no tenen un ens que doni les ordres, però els seus individus són capaços d'autoorganitzar-se utilitzant només la informació del seu voltant, de l'entorn o d'altres individus, per tal de prendre una decisió sobre l'acció a realitzar i resoldre un problema, com per exemple construir un niu.

Es pot utilitzar per a resoldre problemes de gran complexitat utilitzant regles bàsiques i senzilles, i sense tenir coneixement de tot l'entorn del problema. Problemes que en un principi semblen molt complexos de solucionar poden ser resolts a partir de un algoritme senzill que només solucioni un problema de forma local, i que quan aquesta informació es combina amb la de la resta dels agents, dóna una solució al problema global.

3 Seguiment

A la secció 1.4 s'han definit les diferents etapes i tasques del projecte. En aquesta secció, es farà un seguiment detallat del seu compliment i canvis realitzats.

Com a resum es pot dir que la planificació no ha sigut del tot satisfactòria. S'han assolit tots els objectius principals del projecte, però no s'han aconseguit realitzar les tasques addicionals de **Visualització i aplicació** de l'algoritme.

A continuació, s'amplia en que ha consistit cada tasca, agrupat per fites, i s'especifica el grau de compliment amb la planificació. Les tasques planificades es poden veure a la figura 1.

3.1 Desenvolupament

- Descripció del model natural: document que explica en l'observació i estudi de quina espècie es basa el nostre algoritme implementat.
- Revisió graf i algoritme bàsic: s'han revisat els fitxers de codi ja disponibles i corregit alguns errors, siguin de documentació o codificació. Les classes afectades són les que es troben en els paquets: `graphs` i `swarmintelligence`.
- Definició i implementació dels nous algoritmes a partir del bàsic. Les diferents versions de l'algoritme bàsic no es troben en una etapa funcional al final del temps planificat.

No s'han pogut completar totes les tasques en el temps planificat. Les diferents versions es troben definides però la seva implementació encara no és funcional. Ha sigut suficient agafar deu hores de la tasca de **Visualització i aplicació**, així com els dos dies prevists de contingència, per a obtenir una versió funcional de les noves versions. És suficient aplicar un desplaçament de les tasques posteriors a aquesta etapa amb un valor de vint hores, quatre dies, i reduir en deu hores la tasca de la interfície gràfica.

3.2 Benchmarking

- Estudi i implementació de Dijkstra: s'ha estudiat i implementat una versió de l'algoritme. Ens serà de gran ajuda per a obtenir la millor solució al problema del camí mínim i comparar-la amb l'obtinguda amb els nostres diferents algoritmes. L'algoritme s'ha afegit directament a la mateixa definició de graf no dirigit. El mòdul es troba en el paquet `graphs` amb el nom de `graph_undirected.py`.

- Mètriques: s'han definit un conjunt de mètriques que ens permeten comparar els resultats obtinguts amb les diferents versions de l'algoritme bàsic, així com amb Dijsktra. El mòdul que ho implementa es troba dintre del paquet `utils`, amb el nom de `benchmarking.py`. Les mètriques s'utilitzen intercalades en el mateix codi dels nostres algoritmes per a obtenir els resultats de la seva execució.

Les mètriques que genera l'execució de l'algoritme es troben separades en dos conjunts:

- **Globals**: contindran informació global de l'execució, com ara la suma total de pesos i llargàries de tots els agents per a cada iteració.

- **Locals**: emmagatzemen informació de cada agent per a cada iteració. Per exemple, informació de la millor ruta obtinguda per cada agent a cada iteració.

Al final de cada iteració podem emmagatzemar la informació de les mètriques globals i locals a un fitxer en format .csv per a poder realitzar anàlisis posteriors. La definició dels diferents objectes implementen la funció necessària per a obtenir els diferents atributs en un format adient.

Addicionalment, s'ha definit una classe que ens permet obtenir amb facilitat el temps d'execució d'un tros de codi. A més, implementa diferents possibilitats a l'hora de mostrar els valors. Ho podem trobar al paquet `utils` amb el nom `my_time.py`

Aquestes mètriques es consideren suficients per a poder estudiar l'evolució de cada algoritme i comprovar si per a cada iteració de l'algoritme, s'obtenen millors rutes, i estudiar la tendència global de tots els agents.

3.3 Execució i obtenció de resultats

Les següents tasques no han tingut gaire dificultat, però ens hem trobat que el temps d'execució de l'algoritme és elevat. No ens ha afectat, ja que podem seguir amb les següents tasques amb petites proves de l'execució, que no tenen temps d'execució elevats.

- Implementació de creació automàtica de grafs: s'han implementat dos mètodes de generació de grafs. Un ens permet generar grafs controlant el nombre de nodes, arestes i pesos de les arestes, sense una topologia concreta. El segon mètode ens genera grafs amb cicles, a partir d'un graf línia hi afegeix arestes amb grafs generats amb el mètode general.

- Definició de diferents configuracions: amb la tasca anterior podem generar dos tipus de grafs, com ja hem indicat.
- Càrrega de grafs: mètodes necessaris per a llegir grafs a partir d'un fitxer. S'han obtingut un conjunt de grafs d'exemple de la xarxa.
- Execució i obtenció de resultats: és una tasca automàtica que s'ha anat executant fins al final del projecte, per tal de realitzar diferents proves.

Les tasques s'han realitzat en el temps previst.

3.4 Anàlisi

Aquesta etapa s'ha complicat en excés i ens ha obligat a descartar la realització de la interfície gràfica.

- Estudi dels resultats obtinguts: a causa de la gran quantitat de fitxers resultants, configuracions a testejar, etc., ha sigut necessari crear dos *scripts* en R per a realitzar l'estudi. Aplicant els dos *scripts* als resultats obtinguts obtenim un bon resum de les millors configuracions.
- Conclusions: amb l'estudi dels fitxers generats automàticament podem dir que les diferents versions de l'algoritme milloren els resultats obtinguts per l'algoritme bàsic. Això s'ampliarà en aquest mateix document a la secció 7.

Com ja hem comentat, ha sigut necessari agafar el temps previst per a implementar la interfície gràfica per tal de poder acabar el projecte correctament. Hem hagut de fer canvis a les diferents versions i realitzar gran quantitat de proves per tal d'obtenir bons resultats. A més, els *scripts* d'anàlisi de resultats s'havien planificat incorrectament, ja que s'han necessitat 20 hores addicionals entre la programació i el testeig de diferents execucions.

4 Arquitectura del Sistema

Una part important en tot projecte és la decisió de les tecnologies a utilitzar i la seva arquitectura. Una bona elecció d'ambdós ens permetrà centrar tots els esforços en solucionar el problema presentat i no en resoldre problemes tecnològics o de metodologia.

En aquest capítol mostrarem una visió general de les estructures del projecte: les tecnologies principals utilitzades i el perquè s'han elegit, i l'estructura del programa.

4.1 Tecnologies utilitzades

S'ha de tenir present que el nostre projecte és el desenvolupament informàtic d'un programa, que implementa un nou algoritme basat en IE, i de la posterior anàlisi dels resultats obtinguts. Per tant, les tecnologies a utilitzar es centren únicament en llenguatges de programació.

Etapa de desenvolupament. Per a desenvolupar l'algoritme s'ha elegit el llenguatge Python. Els motius són:

- És de codi obert i s'utilitza en assignatures del Màster.
- Facilitat de programació: el codi és fàcilment llegible i té implementacions en les seves llibreries base de multitud d'estructures, com ara: conjunts, llistes, diccionaris, etc. Moltes d'aquestes ens seran molt útils per a implementar el nostre algoritme i les diferents parts del programa.
- Disponible en múltiples plataformes: el llenguatge no és per definició multiplataforma però s'hi considera, ja que alguns dels seus intèrprets sí ho són. Per tant, si existeix un intèrpret per a la plataforma (Windows, Unix, o altres), podrem executar un mateix script sense haver de fer cap adaptació del codi.

Etapa d'anàlisi. Per a l'etapa d'anàlisi de resultats s'ha utilitzat R. També és de codi obert i s'ha utilitzat a les assignatures d'estadística del Màster. És un llenguatge àmpliament utilitzat per la comunitat educativa per a realitzar anàlisis estadístiques i representar gràfics.

Un dels desaventatges del llenguatge és la seva baixa eficiència en el temps d'execució. Això no ens resulta cap problema. ja que no és una aplicació en temps real, i s'ha descartat realitzar mètriques de temps d'execució.

4.2 Estructura del programa

És fonamental que l'estructura del programa sigui modular, és a dir, permeti afegir mòduls addicionals o noves funcionalitats de forma senzilla sense la necessitat de modificar altres parts del codi. Ens ha de permetre, per exemple, canviar la implementació de graf no dirigit a dirigit sense haver de canviar altres parts del codi.

El nou mòdul només s'haurà d'adaptar al nostre format d'entrada i sortida de dades. En l'exemple anterior, si volem canviar la implementació del graf serà suficient en implementar la interfície definida de graf. Si s'implementen tots els mètodes correctament, el nostre algoritme seguirà funcionant sense necessitat de cap canvi.

L'estructura del programa es divideix en tres grans paquets: `graphs`, `swarmintelligence` i `utils`. S'ha de tenir en compte que només es tractarà la versió definitiva de l'algoritme. No es comentaran els paquets de les versions inicials. A la figura 2 en podem veure l'estructura.

A continuació s'explica detalladament el contingut de cada paquet.

4.2.1 Graphs

Sabem que un graf és un conjunt de nodes, i que aquests s'uneixen mitjançant arestes. En el projecte s'ha implementat la definició de graf no dirigit.

Aquest paquet conté els mòduls bàsics de la definició d'un graf. El mòdul més important és la interfície `graph` que serà la que s'implementarà al mòdul `graph_undirected`. Aquesta conté la descripció bàsica dels mètodes que ha d'implementar qualsevol tipus de graf si es vol que el nostre algoritme pugui funcionar sobre aquesta definició de graf: dirigit, no dirigit, mixte, etc. Qualsevol definició que pugui implementar tots aquests aquests mètodes podrà ser utilitzada per a executar-hi el nostre algoritme.

Una funcionalitat a destacar és la implementació d'un mètode per a generar grafs amb gran quantitat de cicles. La seva descripció és senzilla:

- Generació d'un graf línia de llargària N.
- Generació de M grafs amb qualsevol estructura.
- Cada graf generat s'uneix a un dels nodes del graf línia.

Si executam el nostre algoritme fixant com a nodes origen i destí alguns que formin part del graf línia original tindrem que els agents tindran probabilitats altes d'elegir una ruta sense sortida i hauran de tornar enrere, resultant en un cicle.

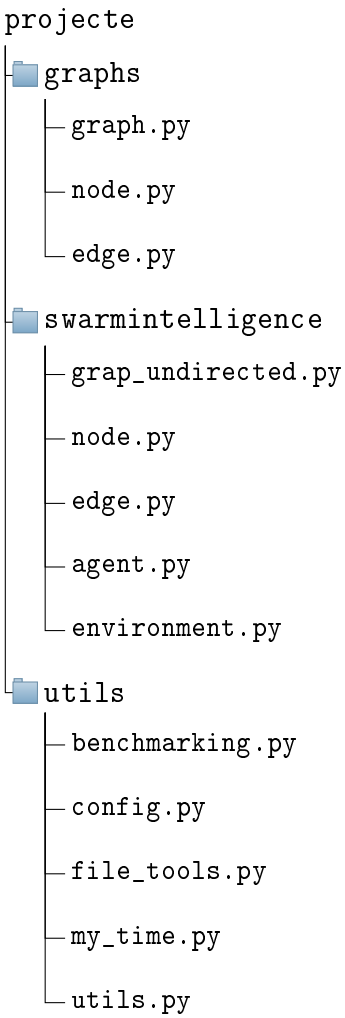


Figura 2: Estructura del projecte.

4.2.2 Swarmintelligence

Conté tots els mòduls i classes necessàries per a definir l'estructura d'un graf, així com la implementació de l'algoritme.

Els mòduls es poden agrupar en dues parts:

- Graf: definició d'un graf els elements que el formen, nodes i arestes. Els mòduls són: `graph_undirected`, `node` i `edge`.
- Algoritme: definició de l'entorn i de l'agent. Els mòduls són: `environment` i `agent`.

Graf. Format pels mòduls que implementen l'estructura de graf utilitzada per l'algoritme. La descripció dels mòduls és la següent:

- **graph_undirected**: implementa la definició de graf no dirigit que permet generar, o carregar, grafs sobre els quals executar el nostre algoritme. Els seus atributs són: `nodes` i `edges`, que contenen el conjunt de nodes i arestes que formen el graf, respectivament. Les funcionalitats més importants són:
  - Crear un nou objecte de tipus graf, ja sigui generat de forma automàtica o llegit des d'un fitxer.
  - Obtenir la llista de nodes i arestes que formen el graf.
  - Obtenir informació important donat un node, com ara, els seus veïns i les arestes que els uneixen.
- **Node**: mòdul descartat. Inicialment no es sabia si necessitaríem que un node tingués més atributs que una id, i per això es definí dita classe. A mesura que el projecte avançà, es decidí descartar dita classe i fou substituïda per objectes de tipus String.
- **Edge**: classe contenidora sense cap funcionalitat. Els seus atributs són: trail, que indica el nivell de feromona, i weight, que indica el pes. No és necessari que es guardi l'origen i destí dels nodes que uneix, ja que aquesta informació l'obtenim de l'atribut `edges` de la classe `graph_undirected`.

Algoritme. No s'entrarà en detall, ja que a la secció 5 s'explica l'algoritme, i els seus dos mòduls implicats, de forma detallada.

El mòdul `environment` s'encarrega de guardar tot tipus d'informació necessària per tal d'analitzar posteriorment el funcionament de l'algoritme. Un

dels fitxers més importants generats i que serà la base per a realitzar una anàlisi posterior és el que guarda la informació de, a cada iteració de l'algoritme, quina és la millor ruta obtinguda.

4.2.3 Utils

Implementa multitud de mòduls amb tasques totalment diferents. En principi les seves funcions són genèriques i no depenen del format de dades dels altres paquets.

Config. Mòdul que s'encarrega d'inicialitzar i gestionar els directoris i fitxers per a guardar els resultats que s'obtenen amb l'execució de l'algoritme. Es recomana només introduir manualment la variable `path_files`. Per a la resta, és millor utilitzar els mètodes que es troben en el mateix mòdul.

Els fitxers més importants són:

- path_file_config: configuracions d'entrada a l'algoritme.
- path_file_dijkstra: valors obtinguts amb l'execució de l'algoritme de Dijkstra com són el pes mínim, l'origen i destí, i la ruta.
- path_file_basic: pes i llargària de la ruta de cada agent per a cada iteració.
- path_file_best_actual: pes i llargària de la millor ruta trobada per un agent a la iteració actual.

Benchmarking. Benchmarking: mòdul que conté un nombre de classes que ens permeten obtenir els resultats de l'execució de l'algoritme i faciliten la tasca d'emmagatzemar-los en fitxers.

File_tools. Proporciona funcionalitats generals pel tractament de fitxers implementant les funcions de *load*/*save*. També facilita la tasca de creació de fitxers i directoris.

My_time. Ens permet obtenir el temps en distints formats. S'utilitza com a suport del mòdul logger. Ens permet emmagatzemar un punt d'inici i final i convertir-lo entre formats.

Utils. Funcionalitats generals compartides per tots els mòduls. En aquest cas només implementa la funció d'obtenir els elements d'una llista de dos en dos.

5 Desenvolupament de l'algoritme

A l'hora d'implementar l'algoritme s'han posat una serie de restriccions:

- L'entorn no pot donar informació als agents. Així, no és possible que un agent conegui quina és la millor ruta trobada.
- Els agents no és comuniquen de forma directa, sinó que ho fan a partir de modificacions sobre l'entorn.

El que és vol aconseguir és simular de la forma més fidedigna el comportament natural, sense perdre l'avantatge d'estar en un entorn artificial.

Per tal d'assolir l'obtenció d'un bon algoritme, aquest s'ha fet de forma incremental. El programa final té tres diferents versions de l'algoritme:

- Versió bàsica: primera versió que ens permet realitzar les primeres proves i fer-ne optimitzacions per tal de millorar els resultats.
- Versió amb pesos: versió que redueix significativament el temps d'execució, així com millora els resultats obtinguts. Això és degut a una modificació en la condició d'aturada de l'agent, que té en compte si la ruta actual té un pes major que la millor ruta trobada per dit agent.
- Versió amb cicles: els agents són capaços de detectar si han passat més d'una vegada per un mateix node. Permet reduir el nivell de feromona d'aquelles arestes que formen el cicle.

Com que la versió final amb cicles ens permet executar les diferents versions segons quins siguin els paràmetres de configuració d'entrada, s'explicarà en detall només aquesta versió.

5.1 Algoritme

El model utilitzat pel nostre algoritme es basa en l'estudi i observació de l'espècie de formiga argentina *Linepithema humile* [4, p.26-29, 32] i [10].

A continuació, es detallarà el comportament d'aquesta espècie i com s'ha adaptat per a crear un nou algoritme que el simuli i doni bons resultats en un entorn artificial.

5.2 Descripció del model natural

El mecanisme que utilitza la formiga argentina és bàsic però s'ha mostrat efectiu. Inicialment, un grup reduït de formigues surten cercant fonts d'aliments. Quan

una formiga en troba una, torna al niu deixant un rastre de feromona. Així, les formigues que es troben a la base comencen a seguir aquest rastre.

Aquesta tècnica funciona pel fet que, com més curt sigui el camí, la formiga tornarà abans deixant el rastre de feromona i la resta de formigues començaran a seguir dita ruta abans, augmentant així el nivell de feromona i fent-la més elegible per a la resta de formigues. A més, donades dues rutes, una més llarga que l'altra, i un mateix nombre de formigues a cada ruta, l'efecte d'evaporació de la feromona serà superior al camí més llarg. Aquesta espècie té gran dificultat en adoptar noves millors rutes sorgides posteriorment a la primera exploració.

Les formigues tindran tendència a seguir aquelles rutes amb un nivell superior de feromona. Així, obtenen una bona aproximació de la ruta més curta entre el niu i la font d'aliments més propera.

Com a curiositat, hi ha espècies amb mecanismes més sofisticats. Per exemple, l'espècie *Lasius niger* és capaç de detectar si va en paral·lel del seu objectiu o niu, per tal de cercar una nova millor ruta.

El nostre algoritme es basarà en la tècnica comentada d'ús de feromones i evaporació d'aquesta. En el nostre cas, sempre tindrem la possibilitat d'afegir nous elements, ja que no tenim cap tipus de limitació. Per exemple, definint una feromona negativa que faci una ruta menys elegible, o directament marcant-la com a no elegible. Això si, es perseguirà mantenir la definició d'intel·ligència eixam i el fet que és un sistema descentralitzat, on tota la informació vindrà donada per l'entorn accessible per l'agent i la seva pròpia informació. Això és, els agents només sabran quins nodes són accessibles a partir de la seva posició actual, i tindran informació de la ruta que han seguit. Els agents no tindran un coneixement global de l'entorn.

Si traduïm el mecanisme natural a un simple algoritme, tenim:

1. Reconeixement de l'entorn per les exploradores cercant possibles fonts d'aliments. Aquestes deixen un rastre de feromona a la tornada al niu si han trobat una font d'aliments.

2. Les recol·lectores segueixen les diferents rutes, sent més elegibles aquelles amb un nivell major de feromona.

3. Decreixement del nivell de feromona amb el pas del temps.

4. Augment del nivell de feromona a la tornada al niu.

L'algoritme repetirà els punts 2 a 4 fins a esgotar totes les fonts d'aliments conegudes, o perdre el rastre d'alguna d'elles per evaporació del nivell de feromona a causa de ser una ruta difícilment elegible. S'ha de tenir en compte que, a la naturalesa, aquestes passes succeeixen de forma simultània.

5.3 Model implementat

El nostre algoritme segueix un plantejament molt similar a l'indicat a la secció 5.2.

Consta de dues parts:

- Entorn, mòdul `environment`: gestiona l'estructura sobre la qual es mouen els agents, en aquest cas un graf. També és l'encarregat de gestionar els agents, i que aquests recorrin el graf.
- Agent, mòdul `agent`: és la unitat que recorre l'entorn, en aquest cas el graf. Obté la ruta que ha realitzat des de l'origen al destí aplicant una sèrie de mètodes per a la selecció d'un node o un altre per a moure's.

Els paràmetres d'entrada utilitzats per l'algoritme són:

- graph: graf sobre el qual executar l'algoritme. Es permet qualsevol configuració de graf, mentre que aquest sigui connex.
- source, dest: nodes origen i destí. Han de ser nodes vàlids del nostre graf.
- max_iter: nombre de vegades que cada agent recorre l'entorn cercant el node destí. Rang: $[0, \infty)$.
- max_steps: nombre màxim de passes que pot realitzar un agent a cada iteració. Rang: $[1, \infty)$.
- n_agents: nombre d'agents que recorreran l'entorn. Rang: $[1, \infty)$.
- min_trail: valor mínim de feromona. Rang: $[0, \infty)$.
- max_trail: valor màxim de feromona. Rang: $(0, \infty)$. Condició: min_trail < max_trail.
- init_trail: valor inicial de feromona a cada aresta. Rang: $[0, \infty)$. Condició: init_trail >= min_trail.
- trail_decay: factor de disminució del valor de feromona que s'aplicarà a cada iteració. Rang: $[0, 1]$.
- q: factor d'increment del nivell de feromona aplicat als nodes que s'han visitat pels diferents agents. Rang: $[0, \infty)$.
- alpha: impòrtancia del valor de feromona a l'hora de calcular la probabilitat d'elegir el node destí. Rang: $[0, \infty)$.

- beta: importància del valor del pes de l'aresta a l'hora de calcular la probabilitat d'elegir el node destí. Rang: $[0, \infty)$.
- use_weight: booleà que ens indica si utilitzar el pes de la millor ruta local obtinguda per cada agent per a la condicio d'aturada. Valors: $[\text{True}, \text{False}]$.
- use_cycle: booleà que ens indica si utilitzar la detecció de cicles per aplicar canvis sobre el nivell de feromona. Valors: $[\text{True}, \text{False}]$
- tr_cycle_decay: factor de disminució de feromona en les arestes dels cicles. Tres possibles rangs:
  - $[0, 1)$: el decreixement de feromona serà proporcional al valor indicat.
  - 1: el nivell de feromona quedarà com abans de la iteració.
  - $(1, \infty)$: el nivell de feromona es reduirà més del que s'ha augmentat a l'etapa d'increment d'aquella iteració.
- as_set: un node pot ser visitat més de dues vegades. En cas que aquest valor sigui vertader, només disminuirem el valor de feromona una vegada. En cas contrari, tantes com (nombre de visites). Valors: $[\text{True}, \text{False}]$.

Els paràmetres més rellevants són: `n_agents`, `alpha`, `beta`, `tr_decay`, `tr_cycle_decay`. I els que ens permeten activar les diferents versions de l'algoritme: `use_weight` i `use_cycle`. A la secció 6 s'entrarà en els detalls.

5.3.1 Implementació bàsica

En la implementació bàsica explicarem el funcionament general de l'entorn, i com decideix l'agent a quin node moure's.

Entorn. L'entorn és l'estructura que s'encarrega de gestionar els agents. L'algoritme segueix les següents etapes:

1. Inicialització del nivell de feromona de cada aresta del graf.
2. Inicialització dels agents. Aquests es situen al node inicial.
3. En aquest punt comença realment l'algoritme. És a dir, el moviment dels agents pel graf. L'entorn controla el nombre de vegades que tots els agents han recorregut el graf. Cada execució es pot dividir en dues parts:
   (a) Recorregut del graf pels agents.
   (b) Fase d'actualització del valor de feromona de les arestes. Consta de les següents fases:

i. Actualització de la millor ruta obtinguda.
ii. Reducció global del nivell de feromona de totes les arestes.
iii. Actualització del nivell de feromona. Cada agent que ha arribat al destí, incrementa el nivell de feromona de les arestes implicades en la seva ruta. En cas que s'apliqui la penalització dels cicles, es fa alhora.
iv. Reinici dels agent. Aquests tornen al node inicial.
v. L'entorn executa una nova iteració de l'algoritme punt 3a.

Agent. Una part fonamental de l'algoritme, i on radica l'obtenció d'uns bons resultats, passa per una bona elecció a cada iteració del pròxim node a visitar.

L'elecció la realitza l'agent i la fa a partir de la informació que obté de l'entorn en el node que es troba. S'utilitzen dos valors: la quantitat de feromona i el pes de l'aresta que uneix els nodes origen i destí.

La fórmula que s'utilitza per a obtenir la probabilitat de dirigir-se a cada un dels nodes veïns **no visitats** és la següent:

$$p_{i,j}^{k} = \frac{(\tau_{i,j}(t))^{\alpha} \cdot (\eta_{i,j})^{\beta}}{\sum (\tau_{i,j}(t))^{\alpha} \cdot (\eta_{i,j})^{\beta}} \tag{1}$$

En cas que ja s'hagin visitat tots els nodes veïns, es selecciona de forma aleatòria un dels nodes, tots ells amb la mateixa probabilitat.

Els elements de la fórmula tenen el següent significat:

- $p_{i,j}^{k}$: probabilitat d'anar d'i a j per l'agent k.
- i, j: nodes origen i destí. Quan es mostren junts, indiquen l'aresta que connecta ambdós nodes.
- k: agent.
- t: iteració de l'algoritme. Cada iteració és un recorregut del graf per tots els agents.
- $\tau_{i,j}$: valor de feromona present a l'aresta.
- $\eta_{i,j}$: inversa del pes de l'aresta, $1/d_{i,j}$.
- α, β: controlen la importància dels valors de feromona i pes, respectivament. Si volem que l'algoritme només tingui en compte la feromona o el pes per a decidir el node objectiu serà suficient posar l'altre valor a 0.

5.3.2 Versió amb pes

Aquesta segona versió, amb un petit canvi aconsegueix reduir significativament el temps d'execució. Així com millorar els resultats obtinguts. Aquest segon punt ho veurem a la secció 6.

El canvi aplicat és que l'agent tindrà dos nous atributs. Un per anar acumulant el pes de la ruta actual i l'altre per a tenir el pes de la millor ruta obtinguda per l'agent fins aquesta iteració. El que sigui la millor ruta de l'agent, i no la global, compleix que els agents no es comuniquen de forma directa, sinó mitjançant l'entorn.

Els canvis al codi de l'agent són mínims. Un primer canvi consisteix en anar acumulant el pes de l'aresta que uneix el node actual amb el següent a visitar. El segon canvi l'aplicam al mètode d'aturada de l'agent, per tal de que s'aturi quan el pes actual sigui major o igual al de la millor ruta local.

L'entorn no es veu afectat per dit canvi.

Amb aquest senzill canvi, l'agent evita realitzar rutes pitjors a la seva millor trobada. Això produeix tres millores:

- Reducció del temps d'execució per a cada agent, ja que redueix el nombre d'iteracions.
- No s'augmenta el nivell de feromona de rutes que no són eficients. Només s'augmenta el nivell de feromona de les rutes obtingudes per agents que han arribat al destí. Per tant, si l'agent s'ha aturat abans d'arribar al destí a causa d'haver superat el seu millor pes local, aquesta ruta no es tindrà en compte.
- No és necessari calcular el pes de la ruta per l'entorn, sinó que s'obté del propi agent.

5.3.3 Versió amb cicles

És la versió completa de l'algoritme, ja que ens permet executar les dues implementacions anteriors a partir dels paràmetres d'entrada de configuració.

Un cicle es format per tots aquells nodes, i arestes, intermedis entre un mateix origen i destí. Si tenim això en compte, sembla necessari ser capaços de detectar-os, ja que l'agent ha realitzat passes innecessàries. Si eliminam aquests nodes intermedis que formen el cicle de la ruta, hauríem pogut arribat al destí amb menys passes. Un cicle només es pot donar quan, donat un node, ja s'han visitat tots els veïns. Quan això succeeix, es selecciona qualsevol node veí amb la mateixa probabilitat, i s'identifica el cicle.

Per tal d'evitar que els agents repeteixin el cicle en iteracions posteriors aplicarem una disminució del nivell de feromona de les arestes contingudes en

el cicle. A partir del paràmetre de configuració `tr_cycle_decay` controlarem la reducció a aplicar. El valor a disminuir es troba directament relacionat amb l'increment de feromona a l'aresta aplicat per l'agent en aquella iteració.

Com a resultat, a la següent iteració de l'algoritme, els diferents cicles trobats hauran tingut una penalització en el seu nivell de feromona i resultaran menys elegibles.

A continuació, s'explicaran els canvis necessaris sobre l'algoritme general, i l'algoritme implementat per a treballar amb els cicles i els nodes que el formen.

L'algoritme de detecció de cicles té dues fases.

La primera fase és molt senzilla i és realitzada per l'agent alhora que recorre l'entorn. Cada agent té un atribut, en aquest cas un conjunt, on posam els nodes visitats, `visited`. Així, a cada passa l'agent comprova si el node seleccionat ja és troba en el conjunt. En cas que el node objectiu ja es trobi al conjunt, afegeix un nou valor a la llista interna que conté els cicles, `cycles`. Aquest valor és una tupla de dos índexs, (i, j), que indiquen l'índex inici i final del cicle amb referència a la llista amb la ruta, `tour`. Obtenir j és molt senzill, ja que és el nombre de passes realitzades per l'agent fins aquest punt. Per a trobar i es recorre la llista `tour` des del final cercant la primera aparició del node en conflicte. Amb aquestes tuples podem identificar posteriorment els nodes que formen cada cicle.

La segona fase es realitza al final de cada iteració de l'algoritme, quan actualitza el nivell de feromona. A partir de les tuples de `cycles` reconstruïm les rutes amb els nodes que la formen. No és tan senzill com agafar cada parell d'índexs i obtenir els nodes continguts entre i i j de `tour`. Si ho fem així, estarem retornant un mateix node més vegades de les visitades. Això és deu a que un cicle, o alguns dels seus nodes, poden estar contingut dintre d'un altre cicle més gran.

Per exemple, si tenim:

- ruta=[0, 1, 2, 1, 0]
- cicles=[(1, 3), (0, 4)]

El primer cicle està inclòs en el segon, per tant, només haurem de contar les arestes del segon cicle. Això ho podem saber de forma senzilla a partir dels índexs. Siguin (i, j) els índexs del primer cicle, i (k, l) els del segon, el primer cicle estarà inclòs, o parcialment inclòs dintre del segon si $k <= j$ o $k <= i$.

Per a retornar correctament els nodes, s'ha de tenir en compte aquest fet i fusionar els subcicles creant un nou cicle i eliminant els cicles afectats.

Per a obtenir els subcicles recorrerem la llista `cycles` començant pel final, si el cicle immediatament anterior s'identifica com a subcicle, parcial o complet,

els fusionam, i seguim recorrent la llista `cycles` de forma inversa. Si no ho és, avançam al cicle anterior i repetim el procés.

A diferència de la modificació aplicada a la versió anterior, aquesta millora si té efectes negatius sobre el temps d'execució a l'hora de realitzar els càlculs dels nodes que formen un cicle, però que es compensa amb la millora de les posteriors rutes obtingudes.

6 Resultats i anàlisi

En un principi, l'objectiu final de l'algoritme és, donat un graf, trobar la ruta de pes mínim entre dos nodes. Això no és del tot cert, o no és una definició completa. La solució trobada es pot considerar com aquella a la que l'algoritme ha convergit. No és suficient trobar la millor ruta si a les iteracions posteriors els diferents agents no l'assoleixen com a la bona. Per això, és important que a cada iteració la tendència sigui d'obtenir millors rutes. No és suficient trobar la millor ruta a la iteració tres, si després l'algoritme convergeix a una ruta de major pes. Aquest fet indicaria que la millor ruta s'ha obtingut de sigut casualitat i no per qualitat de l'algoritme.

El nostre algoritme té gran quantitat de paràmetres d'entrada i no tenim cap eina ni mètode que ens indiqui quin és el millor valor per a cada paràmetre. Direm que una configuració és l'assignació d'un valor a cada paràmetre. Aquesta selecció és fa a partir dels diferents valors possibles. Per exemple, si tenim els paràmetres A i B, amb rangs de valors [0, 1, 2] i [True, False], respectivament, dues possibles configuracions són: (A=1, B=True) i (A=2, B=True).

L'obtenció de resultats s'ha realitzat en dues fases. Ambdues consisteixen en executar l'algoritme implementat, però amb dos objectius diferents:

- Una primera fase consistent en provar gran quantitat de configuracions. Cada configuració s'executa un nombre N de vegades, amb un sol node d'origen i un de destí. Amb el posterior anàlisi s'espera poder obtenir un nombre reduït de possibles bones configuracions.

- La segona fase consisteix en, a partir de les bones configuracions obtingudes amb l'anàlisi dels resultats de la fase anterior, executar les configuracions seleccionades amb un node origen i un nombre elevat de nodes destí. Aquesta segona fase és la que ens permetrà saber si una configuració és bona de forma general.

Encara que segurament s'obtindrien millors configuracions si s'executàs cada configuració amb multitud de nodes (origen, destí), això no és possible per l'elevat temps que requereix. Per això, l'obtenció de resultats s'ha dividit en les dues fases indicades.

Com a resultat de l'anàlisi de la primera fase esperam obtenir una ràtio que ens indiqui com de bona s'espera que sigui una configuració.

Quan executam l'algoritme, per a cada configuració i execució d'aquesta, obtenim un fitxer que conté la millor ruta trobada a cada iteració. Aquesta millor ruta no té perquè ser la millor trobada entre totes les iteracions. Anomenarem aquesta ràtio com `Fase1-ràtio`, i es calcula de la forma següent:

- A partir del fitxer amb les rutes agafam els M darrers pesos.
- Realitzam la mitjana d'aquests, `mitjana-pesos`.
- Obtenim la ràtio d'una execució: $\frac{\text{pes-dijkstra}}{\text{mitjana-pesos}}$.
- Obtenim la mitjana de les ràtios de cada configuració.

Aquest valor serà el que utilitzarem per a agafar les millors configuracions.

6.1 Execució de l'algoritme bàsic

S'ha realitzat una primera execució senzilla per tal de comprovar el funcionament de l'algoritme bàsic, ja que el nostre objectiu no és estudiar l'algoritme bàsic, sinó les nostres modificacions proposades. A més, el fet de no utilitzar-lo en l'execució de les versions ens permet utilitzar un major nombre de configuracions, ja que redueix el nombre de configuracions a testejar a la meitat, sent aquestes, a més, les més lentes de provar.

A la fase 1 s'han executat un total de 192 configuracions, i s'ha necessitat un temps total de 3 hores i 50 minuts. S'ha realitzat el producte cartesià dels valors de cada paràmetre de la taula 3.

Els altres valors d'entrada són:

- graph: mediumEWD.txt.
- source: 201.
- dest: 126.

El pes de la millor ruta amb aquest origen-destí és de 1.06216.

Paràmetres i valors		
n_agents = [62, 124, 186, 248]	max_steps = [50]	max_iter= [50]
min_tr = [0]	max_tr=[1000]	init_tr = [1]
tr_decay = [0.25, 0.5, 0.75]	q = [1]	alpha = [0, 1, 2, 3]
beta = [0, 1, 2, 3]	use_weight = [False]	use_cycle = [False]
tr_cycle_decay = [None]	as_set = [None]	

Taula 3: Valors de configuració de l'execució de l'algoritme bàsic.

A la secció 6.2 s'explica per què s'han fixat alguns paràmetres a un únic valor.

Després d'aplicar l'*script* d'anàlisi a les dades obtingudes la millor configuració trobada té una ràtio de 0.79 i una desviació estàndard de 0.123.

Això vol dir que, per haver obtingut el pes mínim, el nostre algoritme hauria d'haver obtingut una ruta amb un pes un 21% més petit.

Amb l'execució de l'anàlisi també podem veure la tendència de la millor ruta trobada a cada iteració. Per a les millors configuracions ha sigut negativa, fet que és una bona senyal. No ens interessa que l'algoritme sigui capaç de trobar una bona solució si després aquesta no és manté. Podem veure dos exemples a la figura 3.

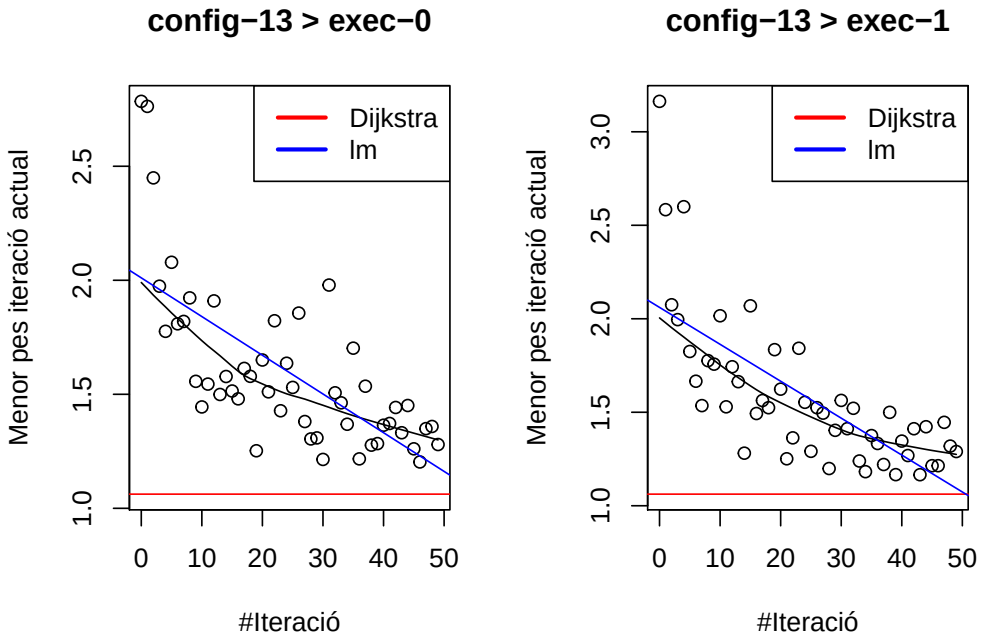


Figura 3: Gràfic amb la tendència de la mitjana de pesos de les millors rutes locals obtingudes per la configuració 13 i les execucions 0 i 1.

No s'ha executat la segona fase, ja que no és el nostre objectiu realitzar un estudi concret de l'algoritme bàsic.

6.2 Execució de l'algoritme: versió final

El resultats d'aquesta execució són els que estudiarem més profundament. El nostre objectiu és obtenir un nombre de bones configuracions per a posteriorment comprovar si aquestes realment ho són executant-les amb un gran nombre de nodes (origen, destí).

6.2.1 Fase 1

No repetirem els comentaris realitzats a la secció 6.1 i ens limitarem a mostrar els resultats obtinguts. Aquesta fase només ens interessa per a obtenir cinc bones configuracions, les quals passarem a la següent etapa d'obtenció de resultats.

S'han executat un total de 576 configuracions, i s'ha necessitat un temps total de 9 hores i 10 minuts. S'ha realitzat el producte cartesià dels valors de cada paràmetre de la taula 3.

Els altres valors d'entrada són:

- graph: mediumEWD.txt.
- source: 201.
- dest: 126.

El pes de la millor ruta amb aquest origen-destí és de 1.06216.

Paràmetres i valors		
n_agents = [83, 166, 249]	max_steps = [50]	max_iter= [50]
min_tr = [1]	max_tr=[20, 1000]	init_tr = [1]
tr_decay = [0.25, 0.5, 0.75]	q = [1]	alpha = [0, 1, 2, 3]
beta = [0, 1, 2, 3]	use_weight = [True]	use_cycle = [False, True]
tr_cycle_decay = [1]	as_set = [False]	

Taula 4: Valors de configuració de l'execució de la versió final de l'algoritme.

El paràmetres fixats a un únic valor són:

- max_steps: és un valor que depèn de la complexitat del graf. Si la ruta a trobar és molt complexa a causa de que el node destí es troba enfora i té un grau baix, aquest valor s'hauria d'augmentar. En el nostre cas, sabem que els grafs utilitzats per a les proves tenen una cota superior de llargària de rutes de 30 i tots els nodes tenen un grau adequat.
- max_iter: s'ha comprovat que el nostre algoritme convergeix a una solució amb menys de 50 iteracions.
- min_trail: valor poc rellevant. El que és cerca posant 1, envers de 0, és que un node sempre sigui elegible per la serva feromona.
- init_trail: el valor inicial es pot iniciar a qualsevol valor, mentre totes les arestes tinguin el mateix.

- q: no té efectes sobre els resultats obtinguts però convé que estigui en la mateixa escala que els pesos de les arestes. S'utilitza a l'hora de calcular l'increment de feromona: $q/$pes-ruta.

- use_weight: no ens interessa executar l'algoritme bàsic.

- tr_cycle_decay: hem considerat que la penalització a aplicar a les arestes que formen part d'un cicle és que el seu nivell de feromona es mantingui en el valor anterior d'abans de ser incrementat per l'agent.

Els paràmetres amb més d'un valor són els que es troben sota estudi:

- n_agents: un nombre molt elevat d'agents provoca que l¡algoritme convergeixi massa aviat a una ruta poc eficient. Un nombre reduït d'agents difícilment convergeix a una ruta. S'ha d'assignar un valor a la variable segons el nombre de nodes del graf.

- max_tr: no és un paràmetre gaire important. Si la diferència amb min_trail és petita, sempre serà possible obtenir noves rutes, però això pot fer que aquestes siguin dolentes.

- tr_decay: controla el nivell de feromona que es disminueix a cada iteració. Un nivell molt elevat provoca que mai es convergeixi a una ruta. Mentre que un nivell massa petit té l'efecte contrari, una vegada s'ha assolit una ruta, és quasi impossible trobar millors solucions.

- alpha i beta: aquests dos paràmetres es presenten junts, ja que no importa el seu valor, sinó la diferència entre ells. Quan major sigui la diferència, major serà el pes a l'hora d'obtenir la probabilitat del paràmetre major. Alpha per la feromona, beta pel pes de la ruta. En cas que algun d'ells sigui 0, o ambdós, no s'utilitzarà el nivell de feromona, o pes, per a obtenir les probabilitats d'anar als distints nodes veïns.

- use_cycle: ens permet utilitzar, o no, la versió que redueix el valor de feromona dels cicles.

El que ens interessa és que per a cada iteració de l'algoritme la millor ruta trobada tingui pendent decreixent, a part del valor de la ràtio, `Fase1-ràtio`. Això és, que la correlació entre la variable iteració i la del pes de la millor ruta trobada en aquella iteració, tingui una correlació negativa. Aquest valor serà el que ens permetrà descartar gran quantitat de configuracions, ja que només ens interessen aquelles que compleixin aquesta premissa.

Els resultats obtinguts per les cinc configuracions seleccionades es poden veure a la taula 5.

Els seus valors de configuració els podem veure a la taula 6. Només es mostren els paràmetres amb més d'un possible valor.

id conf	mitjana pesos	sd-p	mitjana ràtio	sd-r
config-228	1.214	0.010	0.875	0.027
config-11	1.240	0.007	0.858	0.036
config-383	1.356	0.004	0.797	0.116
config-482	1.355	0.004	0.788	0.067
config-269	1.415	0.018	0.772	0.136

Taula 5: Valors obtinguts de les cinc millors configuracions.

id conf	config-228	config-11	config-383	config-482	config-269
n_agents	249	249	249	249	249
max_tr	20	20	20	20	1000
tr_decay	0.25	0.25	0.5	0.75	0.25
alpha	2	2	3	3	2
beta	0	0	0	0	0
use_cycle	true	false	true	true	false

Taula 6: Valors dels paràmetres d'entrada de les cinc millors configuracions.

Amb els resultats de la taula 6 podem treure un parell de conclusions:

- L'algoritme funciona correctament quan té un nombre d'agents i nodes similar.
- Que el valor de max_tr sigui baix ens indica que permet seleccionar nodes amb un baix nivell de feromona per tal de cercar noves rutes.
- No s'utilitza en cap de les millors configuracions el pes de l'aresta per a calcular la probabilitat.
- Sembla que la versió completa amb la penalització als cicles és efectiva.

A la figura 4 podem veure dos exemples de la tendència decreixent entre la iteració i el pes de la ruta.

6.2.2 Fase 2

Els resultats complets d'aquesta segona fase els podem veure a l'apèndix B.

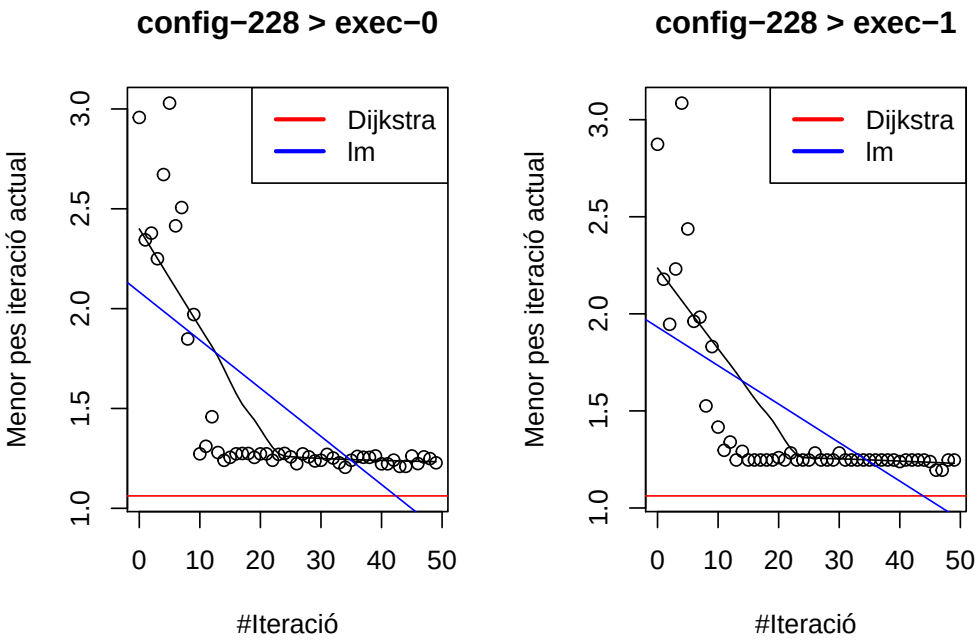


Figura 4: Gràfic amb la tendència de la mitjana de pesos de les millors rutes locals obtingudes per la configuració 228 i les execucions 0 i 1.

S'han provat cinc configuracions amb una sola execució per a cada configuració, entre un node origen i tota la resta de nodes com a destí, que són 249. Això resulta en un total de 1245 execucions de l'algoritme i el temps que ha trigat ha sigut de 2 hores i 23 minuts.

Els resultats obtinguts amb aquest segon anàlisi són molt importants. La primera fase només executa la configuració entre un node origen i un de destí. En aquesta segona fase s'executen les cinc millors configuracions obtingudes a la secció 6.2.1 per a executar-les amb un node origen, i tota la resta de nodes del graf. Així, tindrem una visió molt més global del funcionament del nostre algoritme.

En aquest cas, ens interessa mostrar la ràtio mitjana obtinguda per a cada configuració i tots els destins. Ho podem veure a la taula 7. La columna `r_conver` ens indica el tant per 1 de convergència de l'algoritme. Es considera que l'algoritme ha convergit quan la variació entre les seves N darreres rutes és mínima.

id_config	r_conver	m_ratio	sd_ratio
config-228	1.000	0.935	0.0642
config-11	0.991	0.931	0.0769
config-269	0.995	0.878	0.114
config-383	0.995	0.863	0.127
config-482	1.000	0.824	0.131

Taula 7: Valors resultants de l'estudi de les cinc configuracions.

A partir d'aquests valors podem dir que hem obtingut bons resultats. Una ràtio del 0.935 ens indica que la nostra ruta mínima només ha de disminuir un 0.065 per tal d'assolir el valor mínim ideal.

Una altra estadística a mirar és la distribució de dintre quins rangs cauen les diferents rutes origen-destí. És a dir, volem veure de manera gràfica dintre quins rangs de ràtios ens movem. Podem veure un exemple a la figura 5.

Convé recordar que el nostre algoritme ha d'aconseguir dos objectius:

- Ruta amb un pes mínim, o molt proper a aquest.
- Que la seva tendència de pesos sigui decreixent.

Com podem veure a la figura 5, si només tinguéssim en compte la ruta mínima obtinguda els resultats serien millors, però això seria fals. No ens serveix haver obtingut la ruta mínima si després aquesta no és consolida com a ruta a seguir pels agents a les següents iteracions.

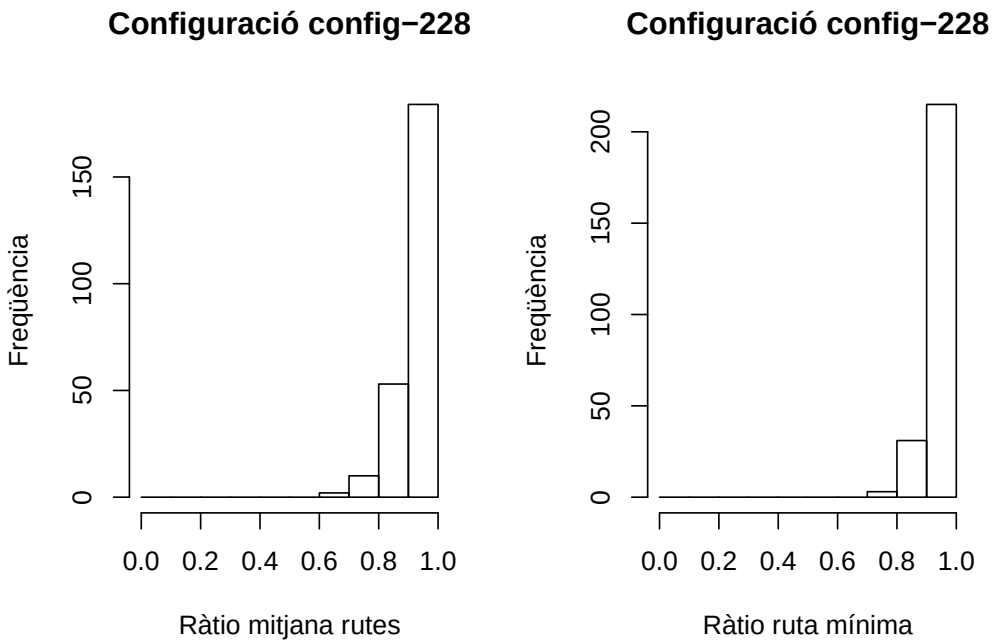


Figura 5: Histograma de la mitjana de pesos mínims de cada ruta.

7 Conclusions

S'han assolit els requeriments inicials del projecte de forma satisfactòria i en podem extreure algunes conclusions.

- Estudi específic del comportament de l'espècie de formiga argentina *Linepithema humile* sobre el qual es basa el nostre algoritme.

- Implementació de diferents versions de l'algoritme base que ens permeten millorar els resultats inicials.

- Els resultats obtinguts són satisfactoris. La millor configuració obté una mitjana de ràtio del 0.93.

- El pes de la millor ruta obtinguda a cada iteració de l'algoritme té tendència decreixent, que és el que cercàvem, però convergeix abans de trobar la millor ruta.

- El valor de feromona té un impacte major en l'obtenció de bones rutes que no el del pes. No només això, per a totes les configuracions sota estudi a la secció 6.2, el nivell del paràmetre β és 0, fet que indica que no s'utilitza el pes a l'hora de calcular la probabilitat d'anar al següent node.

- S'ha aconseguit transformar els coneixements teòrics en la implementació d'un algoritme per a solucionar el problema del camí més curt. A més, l'algoritme és capaç de resoldre de forma satisfactòria el problema del camí més curt.

7.1 Treball futur

- Estudi d'una tècnica per a realitzar proves sobre grafs amb cicles.

- Implementació d'una versió que sigui capaç de detectar si un node és segur que no pot arribar al destí. Per exemple, si un node té grau 1 o tots els seus veïns menys un són no visitable, pot ser marcat com a no visitable.

- Estudi intensiu dels resultats de cada configuració per a detectar els seus punts forts. Amb això és cerca saber si és possible utilitzar més d'un tipus d'agent, amb diferents configuracions, en una mateixa execució i millorar els resultats.

- Eliminar la restricció de no utilitzar informació global de l'entorn per tal d'implementar un augment de feromona de la millor ruta trobada fins aquella iteració. Amb això és reforçarà la millor ruta trobada, però s'ha d'estudiar que això no resulti en una ràpida convergència a aquesta ruta, quan aquesta no és la de pes mínim.

- Realitzar un estudi per cada llargària de ruta. Es possible que algunes configuracions donin bons resultats per a rutes molt properes, i d'altres per a rutes llunyanes.

Glossari

Cami més curt És aquell on la suma dels pesos de les arestes que el forma és mínima.

Cicle Ruta del graf on el node origen i destí és el mateix.

Condició d'aturada És la condició que s'ha de complir per tal que l'agent s'aturi.

Configuració Assignació d'un valor únic per a cada paràmetre d'entrada de l'algoritme, a partir dels possibles valors de cada paràmetre.

Entorn És el medi que recorren els agents, en aquest cas un graf. També es pot referir a la part de l'algoritme que s'encarrega de moure els agents sobre el graf controlant quan s'han d'aturar, actualitzar els valors de feromona a partir de les rutes obtingudes pels agents, etc. És a dir, té una funció de gestió i control.

Feromona Atribut d'una aresta que inidica la desiribilitat d'aquesta de ser seleccionada.

Iteració Té dos possibles usos. A l'entorn, és el nombre de vegades que els agents han recorregut el graf. A l'agent, és el nombre de vegades que aquest s'ha desplaçat.

Node veï Un node vei és aquell al qual es pot arribar amb un sol pas des del node actual. És a dir, el node actual té una aresta sortint cap a aquest.

Pes Atribut d'una aresta que indica el cost de passar per aquesta.

Ruta o camí Llista de nodes recorreguts per un agent.

Acrònims

TFM Treball Final de Màster

TFG Treball Final de Màster

SVM màquina de vector de suport

IA Intel·ligència Artificial

IE Intel·ligència d'Eixam

PSO optimització per eixam de partícules

8 Referències

[1] W. contributors, "Swarm intelligence — Wikipedia, the free encyclopedia," 2018, [Online; accessed 2-June-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Swarm_intelligence&oldid=839454024 1.1, 2.2

[2] ——, "Travelling salesman problem — wikipedia, the free encyclopedia," 2018, [Online; accessed 5-March-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=826478388 1.1.1

[3] ——, "Artificial intelligence — wikipedia, the free encyclopedia," 2018, [Online; accessed 25-March-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=832306017 1.1.2

[4] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. New York, NY, USA: Oxford University Press, Inc., 1999. 1.1.2, 5.1

[5] G. Benison, "The nerdy stuff matters," 2012, [Online; accessed 8-October-2017]. [Online]. Available: https://gcbenison.wordpress.com/2012/03/28/the-nerdy-stuff-matters/ 1.1.2

[6] W. contributors, "Dijkstra's algorithm — wikipedia, the free encyclopedia," 2018, [Online; accessed 22-March-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&oldid=830637516 1.2.1

[7] ——, "Intelligence — Wikipedia, the free encyclopedia," 2018, [Online; accessed 2-June-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Intelligence&oldid=843216868 2.1

[8] B. J. Copeland, "What is artificial intelligence?" 2000, [Online; accessed 2-June-2018]. [Online]. Available: http://www.alanturing.net/turing_archive/pages/referencearticles/whatisai.html 2.1

[9] R. J. Mooney, "Philosophical arguments against 'strong' ai," 2010, [Online; accessed 22-March-2018]. [Online]. Available: http://www.cs.utexas.edu/~mooney/cs343/slide-handouts/philosophy.4.pdf 2.1

[10] G. P. Ripollés, "Hormigas y transferencia de datos en internet," 2016, [Online; accessed 24-April-2018]. [Online]. Available: http://naukas.com/2016/10/18/hormigas-transferencia-datos-internet/ 5.1

[11] L. Gelabert, “Detecció dels colors de pell en imatges rgb utilitzant tècniques d’aprenentatge supervisat,” 2016.

A Manual de l'aplicació

Tot el codi, així com els fitxers resultants obtinguts, es troben en el següent enllaç: arrel del projecte.

El directori arrel conté les carpetes següents:

- Codi: codi del projecte.
- Resultats: directori que conté els resultats obtinguts de l'execució de l'algoritme i de l'anàlisi d'aquests resultats.

Per a executar el programa és suficient amb descarregar el projecte i descomentar la línia de la classe principal segons l'acció que ens interessi realitzar.

A.1 Programari

El projecte s'ha desenvolupat en llenguatge Python. La versió utilitzada ha sigut la 3.6.5.

Addicionalment, s'han necessitat alguns paquets no inclosos en la instal·lació bàsica. Podem llistar els paquets addicionals amb la instrucció `pip list`. A la taula 8 podem veure els diferents paquets i la seva versió.

Nom del paquet	Versió
DateTime	4.2
pip	9.0.3
pytz	2018.4
setuptools	39.0.1
zope.interface	4.4.3

Taula 8: Paquets requerits pel programa.

B Anàlisi específic

Els resultats que es mostren a continuació formen part de la segona fase d'anàlisi automàtic dels fitxers obtinguts amb l'execució de l'algoritme.

S'han provat cinc configuracions amb una sola execució per a cada configuració, entre un node origen i tota la resta de nodes com a destí, que són 249. Això resulta en un total de 1245 execucions de l'algoritme i el temps que ha trigat ha sigut de 2 hores i 23 minuts.

Les configuracions que s'han testejat es poden veure a la taula 4.

TFM - Anàlisi específic de configuracions

Llorenç Gelabert Marí

05 de juny de 2018

Índex

1 Paquets necessaris i paràmetres d'entrada

```
# Paquets necessaris.
packages = c("jsonlite")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(jsonlite)
```

A continuació es mostren els valors dels paràmetres d'entrada

Generals:

- seed = 12345.
- folder_data = ../Results/evaluate_configs_complete/2018-06-02_11-11-21.
- id_algorithm = complete.

Noms de fitxers o carpetes:

- fn_dijkstra = dijkstra.txt.
- fn_info_config = info_configurations.txt.
- fn_global = info.txt.
- fn_best_actual = best_actual.txt.
- dir_exec = exec-0.

Valors:

- v_pmean = 0.1. Tant per 1 d'elements a seleccionar per a realitzar la mitjana del pes de la millor ruta de cada iteració.
- v_pconver: 0.1. Tant per 1 del valor de la mitjana de pesos que utiltizarem per a calcular la maxima sd permesa per a saber si la ruta ha convergit.

2 Inicialització rutes de directoris i fitxers. Informació global

```
# Funcions auxiliars.
round_numeric = function(df, n = 2){
  for (i in 1:ncol(df)) {
    if (class(df[, i]) %in% c("numeric")) {
      df[, i] = round(df[, i], n)
    }
  }
  return(df)
}

# https://stackoverflow.com/a/7963963
substr_right = function(x, n){
  substr(x, (nchar(x) - n + 1), nchar(x))
}
```

```
# Directoris i fitxers importants.
LIST_PATH_RUNS = file.path(params$folder_data, list.files(params$folder_data))
LIST_PATH_DIJKSTRA = file.path(LIST_PATH_RUNS, params$fn_dijkstra)

PATH_CONFIGS = file.path(LIST_PATH_RUNS[1], params$id_algorithm, params$fn_info_config)
```

3 Funcions per a realitzar l'anàlisi

```
NAMES_DIJKSTRA = c("src", "dest", "min_weight")

get_dijkstra = function(list_p){
  df = data.frame(matrix(ncol = length(NAMES_DIJKSTRA), nrow = 0),
                  stringsAsFactors = FALSE)
  for (p in list_p){
    r = fromJSON(txt = p)

    src = r$objective$src
    dest = r$objective$dest
    weight = r$min_weight

    df = rbind(df, data.frame(src, dest, weight, stringsAsFactors = FALSE))
  }

  colnames(df) = NAMES_DIJKSTRA
  return(df)
}
```

```
get_configs = function(path){
  return(fromJSON(txt = path))
}
```

```
NAMES_GLOBAL = c("path_graph", "n_exec")

get_global_info = function(path_dij){
  r = fromJSON(txt = path_dij)
```

```
  path_graph = r$path
  n_exec = length(LIST_PATH_RUNS)

  df = data.frame(path_graph, n_exec, stringsAsFactors = FALSE)
  colnames(df) = NAMES_GLOBAL
  return(df)
}
```

```
NAMES_PATH = c("id_run", "id_config", "path", "src", "dest", "min_weight")
construct_info_best = function(list_path_base){
  # Informacio identificativa de cada execucio de cada configuracio.
  df = data.frame(matrix(ncol = length(NAMES_PATH), nrow = 0), stringsAsFactors = FALSE)

  id_configs = get_configs(PATH_CONFIGS)$id
  for (p in list_path_base){
    dij = get_dijkstra(file.path(p, params$fn_dijkstra))

    id_run = basename(p)
    for (id in id_configs){
      p_aux = file.path(p, params$id_algorithm, id, params$dir_exec,
                        params$fn_best_actual)
      df = rbind(df, data.frame(id_run, id, p_aux, dij$src, dij$dest, dij$min_weight,
                                stringsAsFactors = FALSE))
    }
  }

  colnames(df) = NAMES_PATH
  return(df)
}
```

```
# rended: ratio d'iteracions amb desti trobat. eelems: suficients iters amb desti trobat.
# cor: correlacio entre iteracio i weight.
# m_weight: mitjana del weight dels darrers N elems. sd_weight: desviacio dels mateixos.
# elements. ratio_m: m_weight/dijkstra.
# min_weight: pes minim trobat. ratio_min: min_weight/dijkstra.
NAMES_EXEC = c("rended", "eelems", "cor", "m_weight", "sd_weight", "ratio_m",
               "min_weight", "ratio_min")

anal_exec = function(path, config, dij_weight) {
  rdata = read.csv2(path, header = TRUE, sep = ",", stringsAsFactors = FALSE)
  rdata$weight = as.numeric(rdata$weight)

  rended = paste(nrow(rdata), config$max_iter, sep = "/")

  nelems = floor(config$max_iter * params$v_pmean)
  if (nrow(rdata) >= nelems) {
    subdata = tail(rdata, n = nelems)
    eelems = TRUE
  } else{
    subdata = rdata
    eelems = FALSE
  }

  cv = cor(rdata$iter, rdata$weight)
```

```
  m_weight = mean(subdata$weight)
  sd_weight = sd(subdata$weight)
  ratio_m = dij_weight/m_weight

  min_weight = min(rdata$weight)
  ratio_min = dij_weight/min_weight

  res = data.frame(rended, eelems, cv, m_weight, sd_weight, ratio_m, min_weight,
                   ratio_min, stringsAsFactors = FALSE)
  colnames(res) = NAMES_EXEC

  return(res)
}

NAMES_BEST_ACTUAL = c("id_run", "id_config", "src", "dest", NAMES_EXEC)
get_best_actual = function(df){
  # Obtenim informacio detallada de cada execucio de src->dest per a cada configuracio.
  # df: data frame generat amb construct_info_best.
  configs = get_configs(PATH_CONFIGS)
  df_res = data.frame(matrix(ncol = length(NAMES_PATH), nrow = 0),
                      stringsAsFactors = FALSE)
  for(i in 1:nrow(df)){
    row = df[i, , drop = FALSE]

    id_run = row$id_run
    id_conf = row$id_config
    src = row$src
    dest = row$dest

    c = configs[configs$id == id_conf, , drop = FALSE]

    exec = anal_exec(row$path, c, row$min_weight)

    df_res = rbind(df_res, data.frame(id_run, id_conf, src, dest, exec[1,],
                                      stringsAsFactors = FALSE))
  }

  colnames(df_res) = NAMES_BEST_ACTUAL
  return(df_res)
}
```

```
# id_config: identificador de la configuracio.
# r_conver: ratio de rutes que han covnergit.
NAMES_CONFIG = c("id_config", "r_conver", "m_ratio", "sd_ratio")

get_resume = function(df, list_conf){
  # Agrupam per configuracio i obtenim:
  # - Tant per 1 de rutes (src, dest) on l'algoritme ha convergit.
  # - Mitjana del ratio de la millor ruta.
  # - Desviacio del ratio de la millor ruta.
  df_res = data.frame(matrix(ncol = length(NAMES_CONFIG), nrow = 0),
                      stringsAsFactors = FALSE)
  for (id in list_conf){
```

```
    subset = df[df$id_config == id, , drop = FALSE]
    count = 0
    for (i in 1:nrow(subset)) {
      row = subset[i, , drop = FALSE]
      diff = row$m_weight * params$v_pconver
      if (diff >= row$sd_weight){
        count = count + 1
      }
    }
    rconv = count/nrow(subset)
    mratio = mean(subset$ratio_m)
    sdratio = sd(subset$ratio_m)
    df_res = rbind(df_res, data.frame(id, rconv, mratio, sdratio))
  }
  colnames(df_res) = NAMES_CONFIG
  return(df_res)
}
```

```
NAMES_BASE_HIST = c("id_conf", "tipus")

get_text_hist = function(df, list_idconf, in_seq) {
  df_res = data.frame(matrix(ncol = length(NAMES_BASE_HIST) + length(in_seq) - 1,
                             nrow = 0), stringsAsFactors = FALSE)
  dummy_names = seq(1, length(df_res))
  for (id in list_idconf){
    subset = df[df$id_config == id, , drop = FALSE]
    intervals_mean = cut(subset$ratio_m, breaks = in_seq)

    t_tab = t.data.frame(table(intervals_mean))
    t_prop_tab = data.frame(prop.table(t_tab))

    app = data.frame(id, "mean", t_tab, stringsAsFactors = FALSE)
    colnames(app) = dummy_names
    df_res = rbind(df_res, app)

    app = data.frame(id, "prop_mean", t_prop_tab, stringsAsFactors = FALSE)
    colnames(app) = dummy_names
    df_res = rbind(df_res, app)

    intervals_min = cut(subset$ratio_min, breaks = in_seq)
    t_tab = t.data.frame(table(intervals_min))
    t_prop_tab = data.frame(prop.table(t_tab))

    app = data.frame(id, "min", t_tab, stringsAsFactors = FALSE)
    colnames(app) = dummy_names
    df_res = rbind(df_res, app)

    app = data.frame(id, "prop_min", t_prop_tab, stringsAsFactors = FALSE)
    colnames(app) = dummy_names
    df_res = rbind(df_res, app)
  }

  NAMES_HIST = c(NAMES_BASE_HIST, colnames(t_tab))
  colnames(df_res) = NAMES_HIST
```

```
  return(df_res)
}
```

4 Obtenció dels resultats

```
df_dijkstra = get_dijkstra(LIST_PATH_DIJKSTRA)
head(df_dijkstra)
```

```
##   src dest min_weight
## 1 201  244    0.73463
## 2 201  246    0.84455
## 3 201  239    0.28983
## 4 201  240    0.28619
## 5 201  238    0.51687
## 6 201  245    0.53306
```

```
df_configs = get_configs(PATH_CONFIGS)
head(df_configs)
```

```
##   max_iter max_steps n_agents min_trail max_trail init_trail trail_decay q
## 1       50        50      249         1        20          1        0.25 1
## 2       50        50      249         1        20          1        0.25 1
## 3       50        50      249         1        20          1        0.50 1
## 4       50        50      249         1        20          1        0.75 1
## 5       50        50      249         1      1000          1        0.25 1
##   alpha beta use_weight use_cycle tr_cycle_decay as_set         id
## 1     2    0       TRUE      TRUE              1  FALSE  config-11
## 2     2    0       TRUE     FALSE              1  FALSE config-228
## 3     3    0       TRUE      TRUE              1  FALSE config-269
## 4     3    0       TRUE      TRUE              1  FALSE config-383
## 5     2    0       TRUE     FALSE              1  FALSE config-482
```

```
df_global_info = get_global_info(LIST_PATH_DIJKSTRA[1])
df_global_info
```

```
##                         path_graph n_exec
## 1 .\\exemples grafs\\mediumEWD.txt    249
```

```
df_info_best = construct_info_best(LIST_PATH_RUNS)
df_aux = df_info_best
df_aux$path = paste("...", substr_right(df_aux$path, 35), sep = "")
head(df_aux)
```

```
##                id_run  id_config                                    path
## 1 2018-06-02_22-05-06  config-11 ...te/config-11/exec-0/best_actual.txt
## 2 2018-06-02_22-05-06 config-228 ...e/config-228/exec-0/best_actual.txt
## 3 2018-06-02_22-05-06 config-269 ...e/config-269/exec-0/best_actual.txt
## 4 2018-06-02_22-05-06 config-383 ...e/config-383/exec-0/best_actual.txt
## 5 2018-06-02_22-05-06 config-482 ...e/config-482/exec-0/best_actual.txt
## 6 2018-06-02_22-06-00  config-11 ...te/config-11/exec-0/best_actual.txt
##   src dest min_weight
## 1 201  244    0.73463
## 2 201  244    0.73463
## 3 201  244    0.73463
```

```
## 4 201  244    0.73463
## 5 201  244    0.73463
## 6 201  246    0.84455
```

```
df_best_actual = get_best_actual(df_info_best)
df_aux = round_numeric(df_best_actual[, !(colnames(df_best_actual) %in% c("id_run"))], 3)
head(df_aux)
```

```
##    id_config src dest rended eelems    cor m_weight sd_weight ratio_m
## 1  config-11 201  244  50/50   TRUE -0.449    0.790     0.021   0.930
## 2 config-228 201  244  50/50   TRUE -0.507    0.886     0.021   0.829
## 3 config-269 201  244  50/50   TRUE -0.152    1.006     0.020   0.730
## 4 config-383 201  244  50/50   TRUE -0.456    0.840     0.000   0.875
## 5 config-482 201  244  50/50   TRUE -0.383    0.925     0.000   0.794
## 6  config-11 201  246  50/50   TRUE -0.643    0.887     0.026   0.952
##   min_weight ratio_min
## 1      0.769     0.955
## 2      0.863     0.851
## 3      0.947     0.775
## 4      0.820     0.896
## 5      0.787     0.934
## 6      0.845     1.000
```

```
df_resume = get_resume(df_best_actual, df_configs$id)
df_resume[order(-df_resume$m_ratio), , drop = FALSE]
```

```
##    id_config  r_conver   m_ratio   sd_ratio
## 2 config-228 1.0000000 0.9358975 0.06424865
## 1  config-11 0.9919679 0.9316261 0.07696041
## 3 config-269 0.9959839 0.8782846 0.11467005
## 4 config-383 0.9959839 0.8630772 0.12738177
## 5 config-482 1.0000000 0.8248503 0.13107421
```

```
seq_05 = seq(0, 1, 0.05)
df_hist_text05 = round_numeric(get_text_hist(df_best_actual, df_configs$id, seq_05), 3)
t.data.frame(df_hist_text05)
```

```
##            [,1]        [,2]        [,3]        [,4]        [,5]
## id_conf    "config-11" "config-11" "config-11" "config-11" "config-228"
## tipus      "mean"      "prop_mean" "min"       "prop_min"  "mean"
## (0,0.05]   "0"         "0"         "0"         "0"         "0"
## (0.05,0.1] "0"         "0"         "0"         "0"         "0"
## (0.1,0.15] "0"         "0"         "0"         "0"         "0"
## (0.15,0.2] "0"         "0"         "0"         "0"         "0"
## (0.2,0.25] "0"         "0"         "0"         "0"         "0"
## (0.25,0.3] "0"         "0"         "0"         "0"         "0"
## (0.3,0.35] "0"         "0"         "0"         "0"         "0"
## (0.35,0.4] "0"         "0"         "0"         "0"         "0"
## (0.4,0.45] "1.000"     "0.004"     "0.000"     "0.000"     "0.000"
## (0.45,0.5] "0.000"     "0.000"     "0.000"     "0.000"     "0.000"
## (0.5,0.55] "0.000"     "0.000"     "0.000"     "0.000"     "0.000"
## (0.55,0.6] "0.000"     "0.000"     "1.000"     "0.004"     "0.000"
## (0.6,0.65] " 0.000"    " 0.000"    " 0.000"    " 0.000"    " 0.000"
## (0.65,0.7] " 1.000"    " 0.004"    " 0.000"    " 0.000"    " 2.000"
## (0.7,0.75] " 5.000"    " 0.020"    " 0.000"    " 0.000"    " 0.000"
## (0.75,0.8] "10.000"    " 0.040"    " 2.000"    " 0.008"    "10.000"
```

```
## (0.8,0.85] "16.000"    " 0.064"    "14.000"    " 0.056"    "14.000"
## (0.85,0.9] "31.000"    " 0.124"    "25.000"    " 0.100"    "39.000"
## (0.9,0.95] "53.000"    " 0.213"    "46.000"    " 0.185"    "56.000"
## (0.95,1]   "132.000"   "  0.530"   "161.000"   "  0.647"   "128.000"
##            [,6]         [,7]         [,8]         [,9]
## id_conf    "config-228" "config-228" "config-228" "config-269"
## tipus      "prop_mean"  "min"        "prop_min"   "mean"
## (0,0.05]   "0"          "0"          "0"          "0"
## (0.05,0.1] "0"          "0"          "0"          "0"
## (0.1,0.15] "0"          "0"          "0"          "0"
## (0.15,0.2] "0"          "0"          "0"          "0"
## (0.2,0.25] "0"          "0"          "0"          "0"
## (0.25,0.3] "0"          "0"          "0"          "0"
## (0.3,0.35] "0"          "0"          "0"          "0"
## (0.35,0.4] "0"          "0"          "0"          "0"
## (0.4,0.45] "0.000"      "0.000"      "0.000"      "0.000"
## (0.45,0.5] "0.000"      "0.000"      "0.000"      "0.000"
## (0.5,0.55] "0.000"      "0.000"      "0.000"      "1.000"
## (0.55,0.6] "0.000"      "0.000"      "0.000"      "5.000"
## (0.6,0.65] " 0.000"     " 0.000"     " 0.000"     " 5.000"
## (0.65,0.7] " 0.008"     " 0.000"     " 0.000"     "11.000"
## (0.7,0.75] " 0.000"     " 0.000"     " 0.000"     "12.000"
## (0.75,0.8] " 0.040"     " 3.000"     " 0.012"     "33.000"
## (0.8,0.85] " 0.056"     " 6.000"     " 0.024"     "29.000"
## (0.85,0.9] " 0.157"     "25.000"     " 0.100"     "27.000"
## (0.9,0.95] " 0.225"     "57.000"     " 0.229"     "36.000"
## (0.95,1]   "  0.514"    "158.000"    "  0.635"    " 90.000"
##            [,10]        [,11]        [,12]        [,13]
## id_conf    "config-269" "config-269" "config-269" "config-383"
## tipus      "prop_mean"  "min"        "prop_min"   "mean"
## (0,0.05]   "0"          "0"          "0"          "0"
## (0.05,0.1] "0"          "0"          "0"          "0"
## (0.1,0.15] "0"          "0"          "0"          "0"
## (0.15,0.2] "0"          "0"          "0"          "0"
## (0.2,0.25] "0"          "0"          "0"          "0"
## (0.25,0.3] "0"          "0"          "0"          "0"
## (0.3,0.35] "0"          "0"          "0"          "0"
## (0.35,0.4] "0"          "0"          "0"          "0"
## (0.4,0.45] "0.000"      "0.000"      "0.000"      "0.000"
## (0.45,0.5] "0.000"      "0.000"      "0.000"      "1.000"
## (0.5,0.55] "0.004"      "0.000"      "0.000"      "2.000"
## (0.55,0.6] "0.020"      "1.000"      "0.004"      "6.000"
## (0.6,0.65] " 0.020"     " 1.000"     " 0.004"     "10.000"
## (0.65,0.7] " 0.044"     " 4.000"     " 0.016"     "19.000"
## (0.7,0.75] " 0.048"     " 2.000"     " 0.008"     "16.000"
## (0.75,0.8] " 0.133"     "15.000"     " 0.060"     "13.000"
## (0.8,0.85] " 0.116"     "36.000"     " 0.145"     "32.000"
## (0.85,0.9] " 0.108"     "34.000"     " 0.137"     "37.000"
## (0.9,0.95] " 0.145"     "49.000"     " 0.197"     "28.000"
## (0.95,1]   "  0.361"    "107.000"    "  0.430"    " 85.000"
##            [,14]        [,15]        [,16]        [,17]
## id_conf    "config-383" "config-383" "config-383" "config-482"
## tipus      "prop_mean"  "min"        "prop_min"   "mean"
## (0,0.05]   "0"          "0"          "0"          "0"
```

```
## (0.05,0.1] "0"          "0"          "0"          "0"
## (0.1,0.15] "0"          "0"          "0"          "0"
## (0.15,0.2] "0"          "0"          "0"          "0"
## (0.2,0.25] "0"          "0"          "0"          "0"
## (0.25,0.3] "0"          "0"          "0"          "0"
## (0.3,0.35] "0"          "0"          "0"          "0"
## (0.35,0.4] "0"          "0"          "0"          "0"
## (0.4,0.45] "0.000"      "0.000"      "0.000"      "0.000"
## (0.45,0.5] "0.004"      "0.000"      "0.000"      "1.000"
## (0.5,0.55] "0.008"      "0.000"      "0.000"      "7.000"
## (0.55,0.6] "0.024"      "0.000"      "0.000"      "3.000"
## (0.6,0.65] " 0.040"     " 2.000"     " 0.008"     "17.000"
## (0.65,0.7] " 0.076"     " 2.000"     " 0.008"     "19.000"
## (0.7,0.75] " 0.064"     "14.000"     " 0.056"     "28.000"
## (0.75,0.8] " 0.052"     "16.000"     " 0.064"     "32.000"
## (0.8,0.85] " 0.129"     "31.000"     " 0.124"     "35.000"
## (0.85,0.9] " 0.149"     "37.000"     " 0.149"     "17.000"
## (0.9,0.95] " 0.112"     "43.000"     " 0.173"     "32.000"
## (0.95,1]   "  0.341"    "104.000"    "  0.418"    " 58.000"
##            [,18]        [,19]        [,20]
## id_conf    "config-482" "config-482" "config-482"
## tipus      "prop_mean"  "min"        "prop_min"
## (0,0.05]   "0"          "0"          "0"
## (0.05,0.1] "0"          "0"          "0"
## (0.1,0.15] "0"          "0"          "0"
## (0.15,0.2] "0"          "0"          "0"
## (0.2,0.25] "0"          "0"          "0"
## (0.25,0.3] "0"          "0"          "0"
## (0.3,0.35] "0"          "0"          "0"
## (0.35,0.4] "0"          "0"          "0"
## (0.4,0.45] "0.000"      "0.000"      "0.000"
## (0.45,0.5] "0.004"      "0.000"      "0.000"
## (0.5,0.55] "0.028"      "0.000"      "0.000"
## (0.55,0.6] "0.012"      "0.000"      "0.000"
## (0.6,0.65] " 0.068"     " 1.000"     " 0.004"
## (0.65,0.7] " 0.076"     " 1.000"     " 0.004"
## (0.7,0.75] " 0.112"     " 5.000"     " 0.020"
## (0.75,0.8] " 0.129"     "13.000"     " 0.052"
## (0.8,0.85] " 0.141"     "31.000"     " 0.124"
## (0.85,0.9] " 0.068"     "39.000"     " 0.157"
## (0.9,0.95] " 0.129"     "42.000"     " 0.169"
## (0.95,1]   "  0.233"    "117.000"    "  0.470"
```

```
seq_1 = seq(0, 1, 0.1)
df_hist_text1 = round_numeric(get_text_hist(df_best_actual, df_configs$id, seq_1), 3)
t.data.frame(df_hist_text1)
```

```
##           [,1]        [,2]        [,3]        [,4]        [,5]
## id_conf   "config-11" "config-11" "config-11" "config-11" "config-228"
## tipus     "mean"      "prop_mean" "min"       "prop_min"  "mean"
## (0,0.1]   "0"         "0"         "0"         "0"         "0"
## (0.1,0.2] "0"         "0"         "0"         "0"         "0"
## (0.2,0.3] "0"         "0"         "0"         "0"         "0"
## (0.3,0.4] "0"         "0"         "0"         "0"         "0"
## (0.4,0.5] "1.000"     "0.004"     "0.000"     "0.000"     "0.000"
```

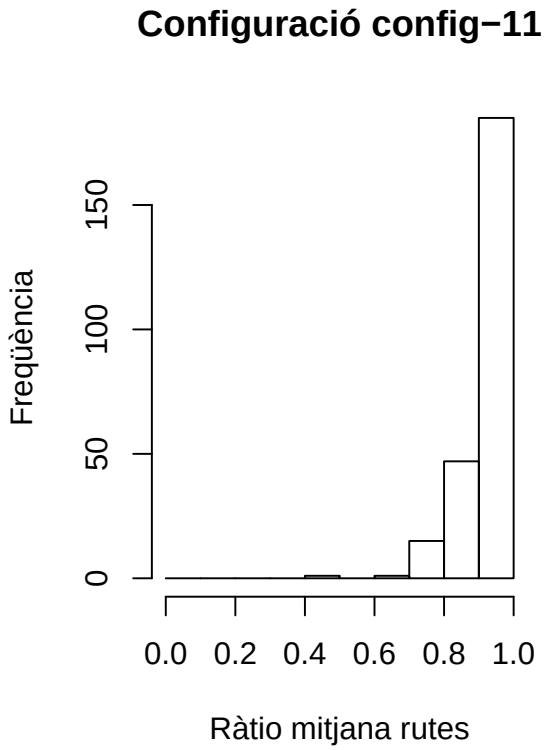
```
## (0.5,0.6] " 0.000"    " 0.000"    " 1.000"    " 0.004"    " 0.000"
## (0.6,0.7] " 1.000"    " 0.004"    " 0.000"    " 0.000"    " 2.000"
## (0.7,0.8] "15.000"    " 0.060"    " 2.000"    " 0.008"    "10.000"
## (0.8,0.9] "47.000"    " 0.189"    "39.000"    " 0.157"    "53.000"
## (0.9,1]   "185.000"   "  0.743"   "207.000"   "  0.831"   "184.000"
##           [,6]         [,7]         [,8]         [,9]         [,10]
## id_conf   "config-228" "config-228" "config-228" "config-269" "config-269"
## tipus     "prop_mean"  "min"        "prop_min"   "mean"       "prop_mean"
## (0,0.1]   "0"          "0"          "0"          "0"          "0"
## (0.1,0.2] "0"          "0"          "0"          "0"          "0"
## (0.2,0.3] "0"          "0"          "0"          "0"          "0"
## (0.3,0.4] "0"          "0"          "0"          "0"          "0"
## (0.4,0.5] "0.000"      "0.000"      "0.000"      "0.000"      "0.000"
## (0.5,0.6] " 0.000"     " 0.000"     " 0.000"     " 6.000"     " 0.024"
## (0.6,0.7] " 0.008"     " 0.000"     " 0.000"     "16.000"     " 0.064"
## (0.7,0.8] " 0.040"     " 3.000"     " 0.012"     "45.000"     " 0.181"
## (0.8,0.9] " 0.213"     "31.000"     " 0.124"     "56.000"     " 0.225"
## (0.9,1]   "  0.739"    "215.000"    "  0.863"    "126.000"    "  0.506"
##           [,11]        [,12]        [,13]        [,14]        [,15]
## id_conf   "config-269" "config-269" "config-383" "config-383" "config-383"
## tipus     "min"        "prop_min"   "mean"       "prop_mean"  "min"
## (0,0.1]   "0"          "0"          "0"          "0"          "0"
## (0.1,0.2] "0"          "0"          "0"          "0"          "0"
## (0.2,0.3] "0"          "0"          "0"          "0"          "0"
## (0.3,0.4] "0"          "0"          "0"          "0"          "0"
## (0.4,0.5] "0.000"      "0.000"      "1.000"      "0.004"      "0.000"
## (0.5,0.6] " 1.000"     " 0.004"     " 8.000"     " 0.032"     " 0.000"
## (0.6,0.7] " 5.000"     " 0.020"     "29.000"     " 0.116"     " 4.000"
## (0.7,0.8] "17.000"     " 0.068"     "29.000"     " 0.116"     "30.000"
## (0.8,0.9] "70.000"     " 0.281"     "69.000"     " 0.277"     "68.000"
## (0.9,1]   "156.000"    "  0.627"    "113.000"    "  0.454"    "147.000"
##           [,16]        [,17]        [,18]        [,19]        [,20]
## id_conf   "config-383" "config-482" "config-482" "config-482" "config-482"
## tipus     "prop_min"   "mean"       "prop_mean"  "min"        "prop_min"
## (0,0.1]   "0"          "0"          "0"          "0"          "0"
## (0.1,0.2] "0"          "0"          "0"          "0"          "0"
## (0.2,0.3] "0"          "0"          "0"          "0"          "0"
## (0.3,0.4] "0"          "0"          "0"          "0"          "0"
## (0.4,0.5] "0.000"      "1.000"      "0.004"      "0.000"      "0.000"
## (0.5,0.6] " 0.000"     "10.000"     " 0.040"     " 0.000"     " 0.000"
## (0.6,0.7] " 0.016"     "36.000"     " 0.145"     " 2.000"     " 0.008"
## (0.7,0.8] " 0.120"     "60.000"     " 0.241"     "18.000"     " 0.072"
## (0.8,0.9] " 0.273"     "52.000"     " 0.209"     "70.000"     " 0.281"
## (0.9,1]   "  0.590"    " 90.000"    "  0.361"    "159.000"    "  0.639"
```

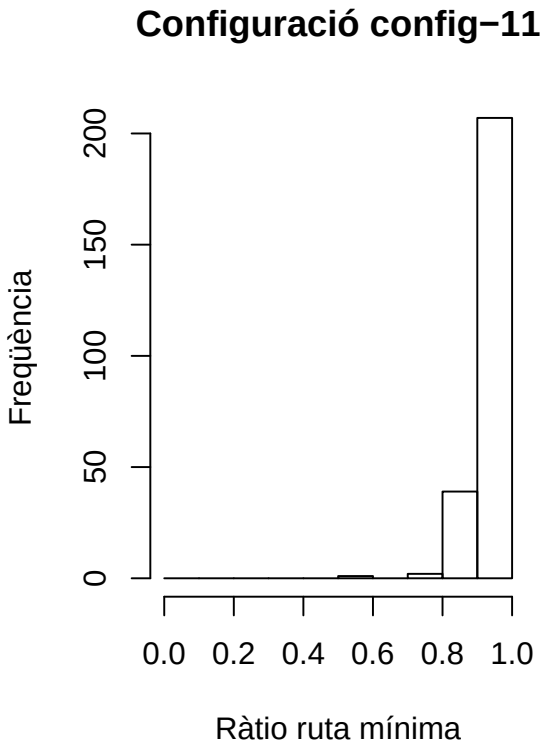
5 Visualització

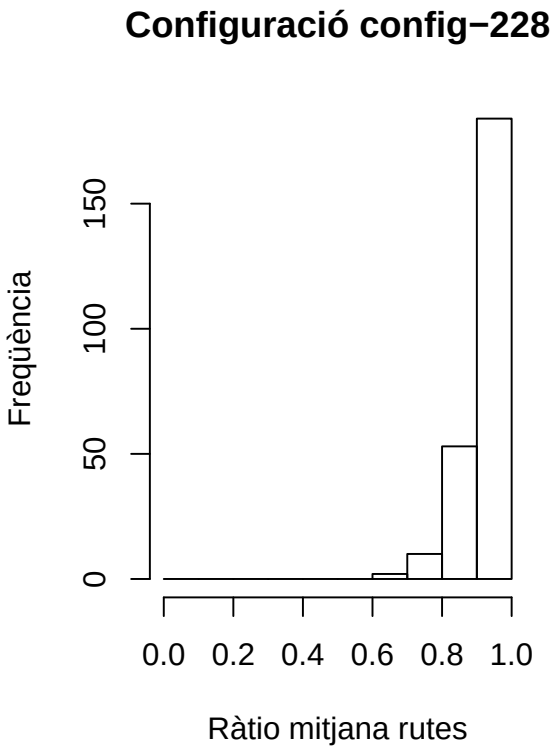
Informació general
Nombre d'execucions amb diferents nodes origen-destí: 249.

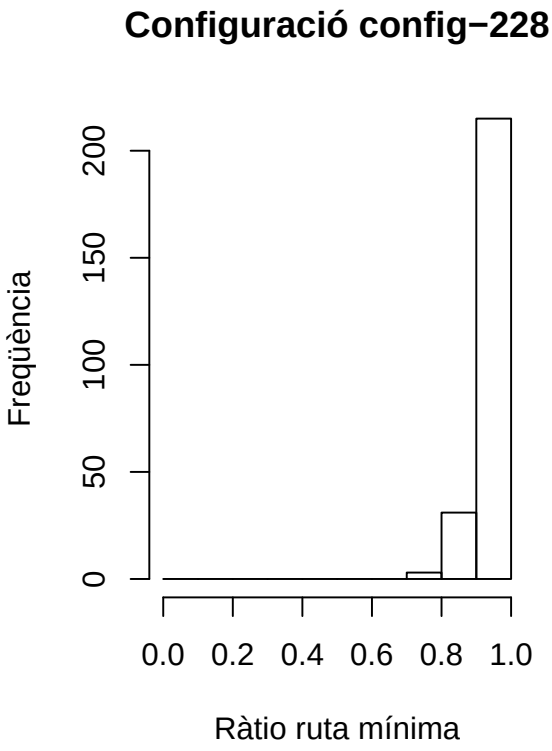
```
seq_cut = seq(0, 1, 0.1)
base_settings = par(mfrow = c(1, 2))
for (id in df_configs$id){
```

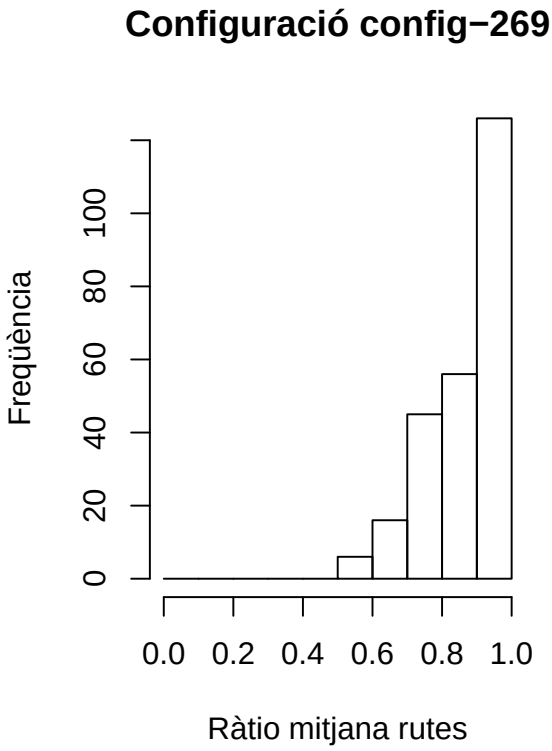
```
  m = paste("Configuració", id, sep = " ")
  subset = df_best_actual[df_best_actual$id_config == id, , drop = FALSE]
  hist(subset$ratio_m, breaks = seq_cut, main = m, xlab = "Ràtio mitjana rutes",
       ylab = "Freqüència")
  hist(subset$ratio_min, breaks = seq_cut, main = m, xlab = "Ràtio ruta mínima",
       ylab = "Freqüència")
}
```

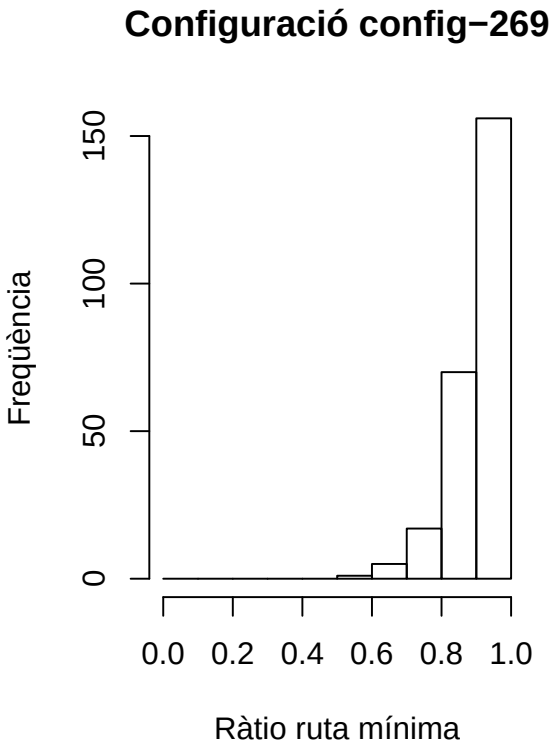


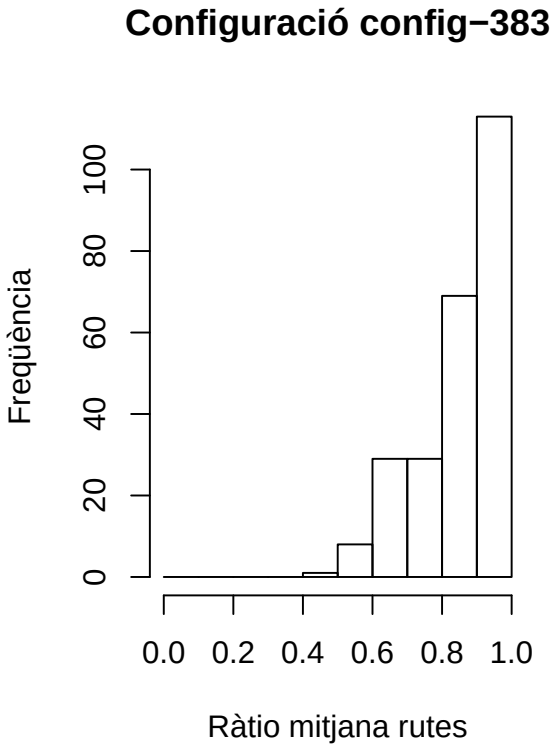


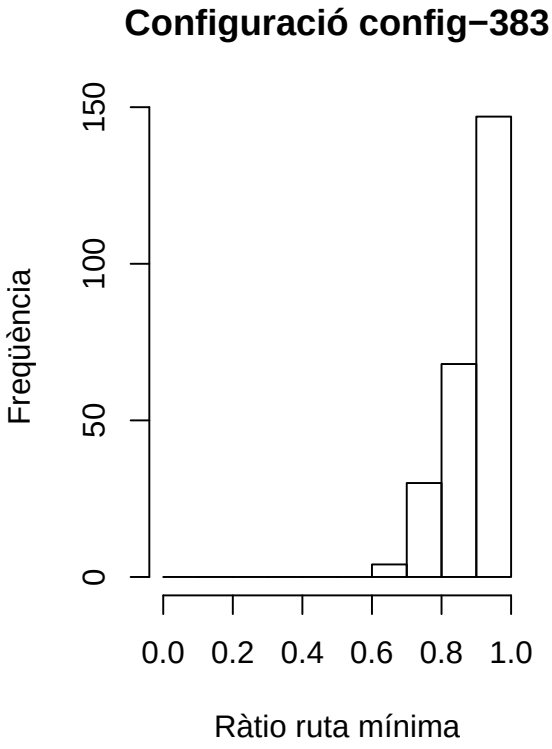

Configuració config−228
Freqüència
0
50
100
150
0.0
0.2
0.4
0.6
0.8
1.0
Ràtio mitjana rutes

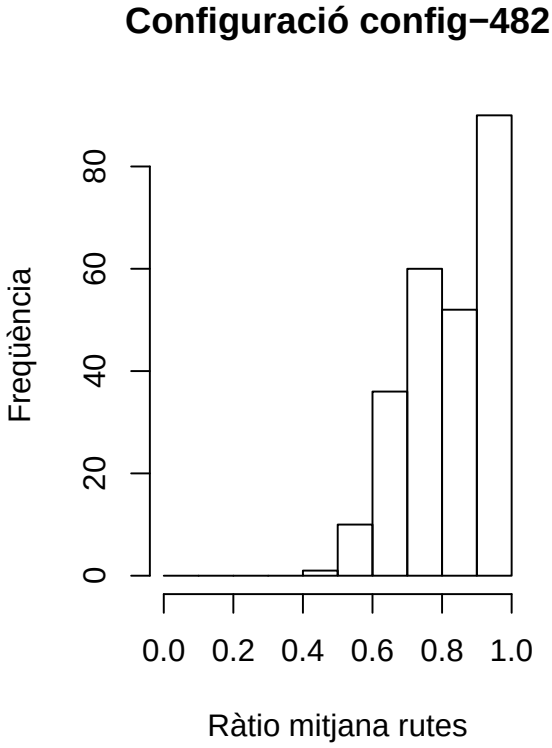

Configuració config−228
Freqüència
0
50
100
150
200
0.0
0.2
0.4
0.6
0.8
1.0
Ràtio ruta mínima

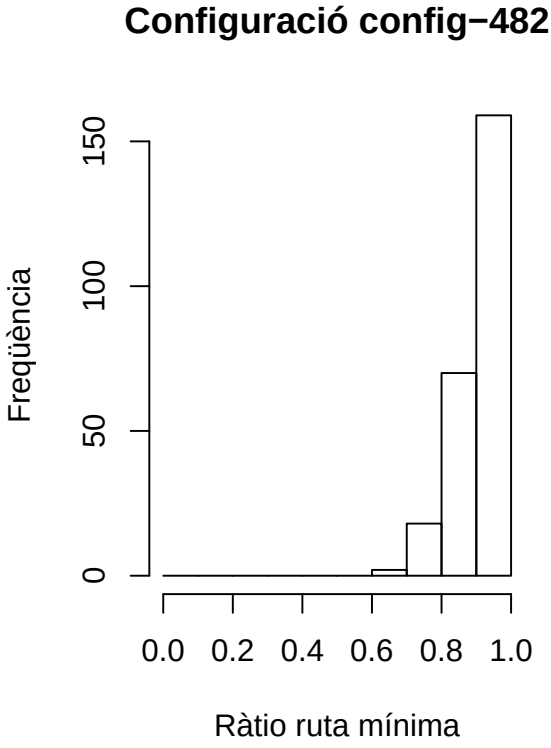

Configuració config−269
Freqüència
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0
Ràtio mitjana rutes


Configuració config−269
Freqüència
0
50
100
150
0.0
0.2
0.4
0.6
0.8
1.0
Ràtio ruta mínima


Configuració config−383
Freqüència
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0
Ràtio mitjana rutes


Configuració config−383
Freqüència
0
50
100
150
0.0
0.2
0.4
0.6
0.8
1.0
Ràtio ruta mínima


Configuració config−482
Freqüència
0
20
40
60
80
0.0
0.2
0.4
0.6
0.8
1.0
Ràtio mitjana rutes


Configuració config−482
Freqüència
0
50
100
150
0.0
0.2
0.4
0.6
0.8
1.0
Ràtio ruta mínima

```r
par(base_settings)
```