Universitat Autònoma de Barcelona

escola d'enginyeria

# (6233-1: IMPLEMENTACIÓ D'UNA SOLUCIÓ APROXIMADA PER AL PROBLEMA DEL VIATJANT GENERALITZAT -TSPg-)

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
José Emilio Sánchez Aparicio
i dirigit per
Joaquim Borges Ayats
Bellaterra, 16 de juny de 2016

El sotasignat, Joaquim Borges Ayats

professor de l'Escola d'Enginyeria de la UAB,

**CERTIFICA**:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en José Emilio Sánchez Aparicio

I per a que consti firma la present.

Signat: Joaquim Borges Ayats

Bellaterra, 16 de juny de 2016

# Taula de continguts

# Capítol 1. Introducció

## Objectius

L'objectiu principal del projecte és resoldre el problema del viatjant (TSP – Travelling Salesman Problem) utilitzant un algorisme d'aproximació basat en aparellaments òptims, tot avaluant-ne la complexitat.

El problema del viatjant es pot plantejar de la següent manera: un viatjant surt d'una població "A" i ha de visitar diverses ciutats per vendre els seus productes. El viatjant té dues alternatives:

- Trobar un recorregut amb origen i final a "A" que visiti un únic cop cada ciutat i tingui cost total mínim (denominat "problema del viatjant" o TSP).

- Trobar un recorregut amb origen i final a "A" que visiti com a mínim un cop cada ciutat i tingui un cost total mínim (denominat "problema del viatjant generalitzat" o TSPg).

El projecte pretén aplicar un algorisme d'aproximació per al cas del TSPg, tot presentant la introducció de les dades (ciutats a visitar) i el resultat (camí resultant) de manera gràfica mitjançant "google maps".

En tractar-se d'un algorisme d'aproximació, el resultat del programa no serà per tant un camí òptim quant a distància a recórrer, però sí que es podrà acotar quina és aquesta distància òptima, mostrant-ne per pantalla les fites.

De manera addicional, es realitzarà també un estudi del cost computacional que tenen els diferents algorismes implementats, identificant quin és el "coll d'ampolla" que limita el temps d'execució i de què en depèn.

## Estat de l'art

### Història del TSP

Diversos problemes relacionats amb el "problema del viatjant (TSP)", com el joc de taula denominat "icosian game" [1], van ser tractats en els anys 1800s pel matemàtic irlandès Sir William Rowan Hamilton [2] i pel matemàtic britànic Thomas Penyngton Kirkman [3]. Pel que sembla, el primer cop que el TSP va ser estudiat per matemàtics va ser durant els 1930's, per Karl Menger [4] en Viena i Harvard. Als 1940's va ser estudiat per diversos estadístics

(Mahalanobis -1940-, Jessen -1942-, Gosh -1948-, Marks -1948-) en relació amb el camp de l'agricultura.

**Complexitat de càlcul**

El TSP és un problema NP-Hard, pel què és probable que en el pitjor cas el temps d'execució per a qualsevol algorisme que resolgui el TSP de manera exacta augmenti de manera exponencial respecte al número d'instàncies o ciutats.

Els programes informàtics capaços d'implementar el TSP han incrementat la seva sofisticació al llarg dels anys el que, acompanyat de la millora de la capacitat de càlcul dels computadors, ha propiciat un augment progressiu de les instàncies no trivials que poden ser solucionades, com es pot veure a la següent taula extreta de [5].

| Year | Research Team | Size of Instance | Name |
|---|---|---|---|
| 1954 | G. Dantzig, R. Fulkerson, and S. Johnson | 49 cities | dantzig42 |
| 1971 | M. Held and R.M. Karp | 64 cities | 64 random points |
| 1975 | P.M. Camerini, L. Fratta, and F. Maffioli | 67 cities | 67 random points |
| 1977 | M. Grötschel | 120 cities | gr120 |
| 1980 | H. Crowder and M.W. Padberg | 318 cities | lin318 |
| 1987 | M. Padberg and G. Rinaldi | 532 cities | att532 |
| 1987 | M. Grötschel and O. Holland | 666 cities | gr666 |
| 1987 | M. Padberg and G. Rinaldi | 2,392 cities | pr2392 |
| 1994 | D. Applegate, R. Bixby, V. Chvátal, and W. Cook | 7,397 cities | pla7397 |
| 1998 | D. Applegate, R. Bixby, V. Chvátal, and W. Cook | 13,509 cities | usa13509 |
| 2001 | D. Applegate, R. Bixby, V. Chvátal, and W. Cook | 15,112 cities | d15112 |
| 2004 | D. Applegate, R. Bixby, V. Chvátal, W. Cook, and K. Helsgaun | 24,978 cities | sw24798 |

*Figura 1. Evolució del nombre d'instàncies no trivials que poden ser resoltes*

El 1990 es va començar a desenvolupar un programa informàtic denominat "Concorde" [6] consistent en més de 130000 línies de codi en C que ha permès resoldre problemes de fins a 85900 ciutats en l'any 2006. Matemàtics i altres professionals treballen actualment en la millora

d'aquest programa.

EL TSP és, per tant, un dels problemes de matemàtica computacional que s'han estudiat més intensament i que encara no té un mètode que el resolgui de manera eficaç en el cas general. De fet, la resolució del TSP en temps polinòmic o demostrar que és impossible suposaria resoldre el problema "P vs NP", que és un dels "problemes del mil·lenni" i que té un premi d'1 milió de dòlars per part del Clay Mathematics Institute [7].

**Algorismes d'aproximació**

Per a obtenir solucions en un temps menor, se solen utilitzar algorismes d'aproximació mitjançant mètodes heurístics. Un algorisme d'aproximació permet trobar una solució propera a l'òptima en un temps raonable. Aquest tipus d'algorismes proporcionen una fita sobre l'error màxim que pot tenir la solució aproximada que retornen, denominada *raó d'aproximació*.

Aquests mètodes, no obstant això, no es poden utilitzar en tots els casos. Per exemple, si el graf no compleix la desigualtat triangular, llavors no hi ha algorismes d'aproximació per al TSP.

Sí que hi ha algorismes d'aproximació per al TSP generalitzat (circuit que passa com a mínim un cop per cada vèrtex, enlloc d'exactament un cop), independentment de si el graf compleix la desigualtat triangular.

Existeixen diversos programes que utilitzen mètodes heurístics per resoldre determinats tipus del TSP, com per exemple el LKH [8].

**Integració amb google maps**

L'API de google maps proveu d'un conjunt de funcions encaminades a facilitar la programació d'algorismes que implementin el TSP. Existeixen alguns programes similars a l'objectiu del projecte, com per exemple [9], que implementa una aproximació del TSP mitjançant un algorisme genètic sobre un mapa de google maps. No obstant això, l'algorisme que es pretén implementar en aquest projecte difereix respecte al tipus de problema que vol resoldre (TSP generalitzat) i en la manera d'enfocar l'algorisme.

També existeix algun precedent d'aquest projecte en el que s'ha fet una implementació de l'algorisme sobre google maps, però es pretén millorar la interfície i refer de nou la programació de l'algorisme.

## Estudi de la viabilitat del projecte

La definició dels objectius del projecte determina quins aspectes s'han d'estudiar per establir si és o no factible. En concret, s'estudiaran la viabilitat tècnica i econòmica, ja que altres aspectes com la viabilitat ambiental, financera o comercial del projecte no tenen sentit en no tractar-se d'un projecte que tingui efectes sobre el medi ambient o un producte que es vulgui explotar comercialment.

Quant a la viabilitat tècnica, cal avaluar dos aspectes bàsics, com són la possibilitat d'utilitzar l'API de google maps per implementar el programa i si és factible que l'algorisme proposat pugui obtenir bons resultats.

En el primer cas sembla clara la viabilitat, ja que existeixen eines similars com [9] en funcionament i, encara que l'algorisme difereixi, s'entén que amb l'ajuda de les funcions específiques de l'API es podrà realitzar la programació de l'algorisme pretès. Respecte a l'algorisme pròpiament dit, la diferent bibliografia existent ens indica que és viable la seva implementació per a resoldre de forma aproximada el TSPg en un temps eficaç.

Finalment, quant a la viabilitat econòmica del projecte, cal dir que l'API de google maps compta amb una llicència gratuïta que, encara que té una limitació respecte al nombre de peticions que se li poden fer al sistema, sembla suficient per a cobrir les necessitats, motiu pel qual la viabilitat econòmica no ha de suposar cap problema per a l'èxit del projecte.

## Planificació temporal del treball

La duració del projecte s'ha d'ajustar a 15 crèdits lectius, és a dir, aproximadament 150 hores. Existeixen dues dates límit imposades pel calendari acadèmic, que són el 14 de desembre de 2015 per a l'entrega de l'informe previ i el 17 de juny del 2016 per a l'entrega de la memòria definitiva del projecte.

En la redacció de l'informe previ, es va proposar una distribució del treball en cinc etapes: l'anàlisi prèvia i estudi teòric dels algorismes, el disseny i programació de la interfície d'entrada de dades, el disseny i la programació de l'algorisme, el disseny i programació de la presentació de les dades de sortida i, finalment, la redacció de la memòria i la presentació del projecte. Es van assignar 30, 25, 50, 35 i 10 hores respectivament a cadascuna de les etapes.

Durant la realització del projecte, però, s'han hagut d'adaptar aquestes previsions. La modificació més important ha estat quant a la programació de les interfícies d'entrada i sortida de dades. Tot

i estar plantejades com a etapes independents, s'ha vist que l'evolució del projecte requeria que es desenvolupessin aquestes interfícies de forma paral·lela a la programació de les diferents parts de l'algorisme, per tal de comprovar-ne el correcte funcionament i veure que els resultats estaven en la línia de l'esperat.

Amb aquesta adaptació, la realitat és que de les 60 hores previstes en aquestes dues etapes, se n'han dedicat força menys, sobre les 30 hores aproximadament. En canvi, en l'etapa de programació de l'algorisme s'han hagut de dedicar unes 10 hores extra per optimitzar l'algorisme d'aparellament. Les 20 hores restants han anat a la redacció de la memòria, que certament tenia assignat una quantitat d'hores molt escassa.

Quant a les dates de finalització de cadascuna de les etapes, ja es van fixar amb força marge respecte les que marca el calendari acadèmic, per tal de poder absorbir les possibles desviacions. Tot i estar fixada la finalització en la planificació el 4 de març de 2016, la realitat ha estat que les tasques d'optimització de l'algorisme han dut aquesta data cap el 15 de maig, en tot cas amb marge suficient per redactar la memòria i enllestir el projecte abans de la data del 17 de juny fixada al calendari per a l'entrega.

En la següent taula es presenten a mode esquemàtic les diferents tasques, amb el nombre d'hores que s'hi ha dedicat a cadascuna i la data de finalització:

| Tasca | Hores | Data finalització |
|---|---|---|
| **1.- Anàlisi prèvia i estudi teòric** | **30** | **10/12/2015** |
| - Estudi de viabilitat i objectius | 3 | 16/10/2015 |
| - Estudi dels fonaments teòrics | 17 | 19/11/2015 |
| - Estat de l'art | 5 | 01/12/2015 |
| - Redacció de l'informe previ | 5 | 10/12/2015 |
| **2.- Implementació i optimització de l'algorisme** | **90** | **15/05/2016** |
| - Estudi i familiarització API de google maps | 10 | 20/12/2015 |
| - Càlcul de la matriu de distàncies | 5 | 15/01/2016 |
| - Algorisme de Kruskal | 10 | 30/01/2016 |
| - Algorisme d'aparellament dels vèrtexs senars | 15 | 28/02/2016 |

| | | |
|---|---|---|
| - Mètode de Hierlholzer | 15 | 30/03/2016 |
| - Estudi del cost computacional | 10 | 15/05/2016 |
| - Optimització de l'algorisme d'aparellament | 10 | 15/05/2016 |
| - Programació interfície d'entrada de dades | 15 | 15/05/2016 |
| **3.- Redacció de la memòria** | **30** | **16/06 /2016** |

## Organització de la memòria

Seguint les indicacions de la normativa sobre el PFC, es planteja una estructura de la memòria amb un primer capítol introductori, on s'estableixen els objectius del projecte, així com la seva viabilitat, a més d'una distribució temporal de les tasques a realitzar i una breu descripció de l'estat de l'art.

Els fonaments teòrics referents a la teoria de grafs i els algorismes aplicats constitueixen la base sobre la què s'ha desenvolupat el projecte, motiu pel qual es dedica el segon capítol a la seva explicació i anàlisi.

Posteriorment, s'expliquen la metodologia i les eines utilitzades en el capítol tercer, l'anàlisi i el disseny del programa en el capítol 4t i la codificació dels diferents algorismes en el capítol 5è.

En el capítol 6è es fa un estudi del cost computacional de les diferents parts de l'algorisme i dels canvis que s'han fet en el codi per optimitzar-lo.

Finalment, en el capítol 7è s'inclouen les conclusions, els objectius aconseguits i les limitacions i possibles ampliacions del projecte.

# Capítol 2. Fonaments teòrics

## Teoria de grafs

En l'àmbit de les matemàtiques i la informàtica la *teoria de grafs* és l'estudi de les estructures matemàtiques utilitzades per modelar relacions entre parelles d'objectes anomenats *vèrtexs* [10].

Un *graf* $G(V, A)$ està format per un conjunt de punts $V = \{v_1, v_2..., v_k...\}$ i per un conjunt de línies $A = \{a_1, a_2..., a_q...\}$, de manera que cada línia uneix dos dels punts. Si les línies estan orientades s'anomenen *arcs* i el graf, *dirigit*; si no ho estan, s'anomenen *arestes* i el graf, *simètric*. Els punts s'anomenen sempre *vèrtexs*. Els conjunts V i A seran sempre finits i, en conseqüència, els grafs resultants també [11].

En el cas dels grafs simètrics, es denomina *grau* d'un vèrtex al nombre d'arestes que hi incideixen. En canvi, si el graf és dirigit, se sol distingir entre el *grau d'entrada* (nombre d'arcs que apunten cap al vèrtex) i el *grau de sortida* (nombre d'arcs que surten del vèrtex). El *lema de l'encaixada de mans* estableix que la suma dels graus d'un graf és el doble del nombre d'arestes. Una implicació d'aquest lema és que el nombre de vèrtexs de grau senar és parell.

Molts problemes de planificació, com els de recollida d'escombraries, repartiment de cartes, inspecció de canonades, sortida de laberints, etc. poden plantejar-se com a problemes d'optimització de recorreguts [12]. Per tant, es poden modelar com un graf on els vèrtexs són els punts per on s'ha de passar i les arestes o arcs els camins per on es pot transitar d'un punt a l'altre.

De manera clàssica, s'estudien dos tipus de problemes d'optimització de recorreguts, els *grafs eulerians* i els *grafs hamiltonians*. En els grafs eulerians, es tracta de solucionar problemes en els què el recorregut passa per totes les arestes o arcs del graf una sola vegada. En els grafs hamiltonians, en canvi, el recorregut passa per tots el vèrtexs del graf una única vegada, començant i acabant en el mateix vèrtex.

## Grafs eulerians

En el segle XVIII va ser Leonhard Euler qui, estudiant el problema de realitzar un recorregut complet pels ponts de Königsberg, va fer una abstracció dels elements que el definien (ponts i ribes), substituint les ribes per vèrtexs i els ponts per arestes. El seu estudi va ser l'origen de la teoria de grafs i va caracteritzar els grafs eulerians.

Euler va adonar-se que el problema es podia resoldre en funció del grau dels vèrtexs, essent el recorregut possible si el nombre de vèrtexs de grau senar era 0 ó 2 [12]. Si era 0, es podia fer un recorregut tancat amb inici i final en el mateix vèrtex (circuit); mentre que si era 2, únicament es podia fer un recorregut amb origen i final en els vèrtexs de grau senar (camí).

El principal teorema que s'extreu del seu treball és [11]: G(V, A) multigraf simètric i connex té circuit (camí) eulerià si i només si el nombre de vèrtexs de grau senar a G és 0 (2).

Cal remarcar com a condicions que el graf ha de ser *connex,* és a dir, que entre cada parella de vèrtexs existeix almenys un camí que els uneix; i que existeix la possibilitat que hi hagin arestes repetides, és a dir, que sigui un *multigraf*.

## Grafs Hamiltonians

L'any 1857, el matemàtic William Rowan Hamilton va presentar i patentar un joc de taula (*icosian game*) en què calia seguir un recorregut tancat sense repetició de vèrtexs, seguint les arestes d'un dodecàedre regular [12]. Aquest va ser l'origen dels grafs hamiltonians, definint-se un graf hamiltonià com aquell que conté un circuit hamiltonià, és a dir, un circuit que passa per cada vèrtex una vegada i només una.

No obstant això, la solució que Hamilton proposà per al seu joc de taula no es pot generalitzar sobre grafs arbitraris. De fet, decidir si un graf qualsevol és hamiltonià és un problema *NP-Complet* segons el teorema de Garey-Johnson [13].

Quant a les definicions dels diferents tipus de problemes en funció de la seva complexitat tenim que:

- Els problemes de tipus P es poden resoldre en un temps polinòmic.
- En els problemes de tipus NP, només es pot assegurar que la validació d'una solució positiva es pot realitzar en temps polinòmic.
- Un problema *H* serà NP-Hard quan qualsevol problema *L* en NP pugui ser reduït en temps polinòmic a *H*.
- Finalment, un problema serà NP-Complet, quan sigui NP i NP-Hard.

No se sap si el conjunt de problemes P és igual a NP, però si es demostrés que un problema NP-Hard és P, tindríem que P=NP.

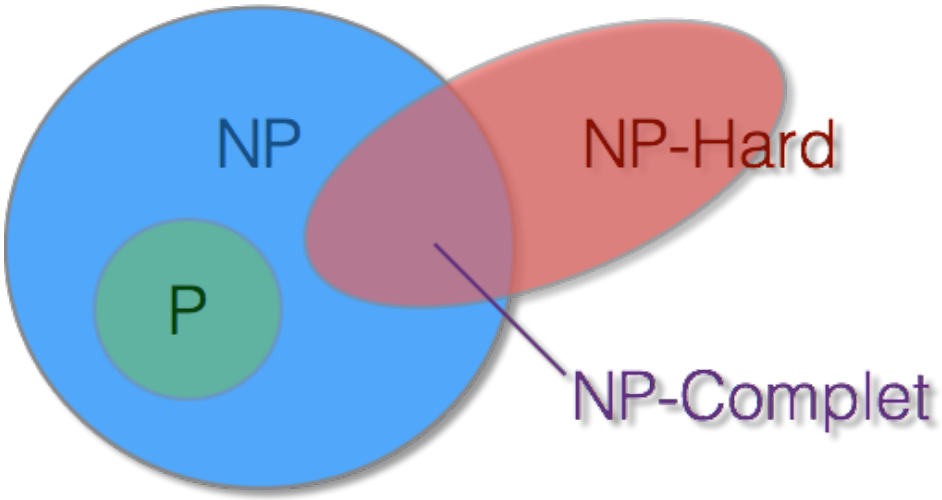



*Figura 2. Diagrama de les famílies de problemes, assumint P≠NP*

Retornant al problema de decidir la hamiltoneïtat d'un graf, no es disposa de condicions necessàries i suficients per poder decidir si un graf qualsevol és hamiltonià, el que és una diferència substancial respecte als grafs eulerians. Sí que existeixen, però, algunes condicions necessàries i d'altres de suficients de hamiltoneïtat per a casos particulars. Una propietat important de cara al projecte és que si un graf simètric és *complet*, amb un número de vèrtexs igual o superior a 3, llavors trivialment té circuit hamiltonià. Que sigui complet significa que cada parella de vèrtexs està connectat per una aresta, com és el cas del graf que s'utilitzarà en el projecte per representar les distàncies que hi ha entre les ciutats que seleccioni l'usuari.

## El problema del viatjant (TSP)

En termes de grafs, el plantejament del problema del viatjant de comerç és el següent: donat un graf simètric (dirigit) G(V, A) amb cost $c_{ij}>0$ associat a cada aresta (arc) $(v_i, v_j)$, es demana trobar un circuit que passi **una vegada i només una** per cada vèrtex i que en minimitzi el cost total [11]. És a dir, cal trobar un circuit hamiltonià de cost mínim.

No obstant això, per a l'elaboració del projecte ens centrarem en l'anomenat TSP generalitzat ($TSP_g$), es demana que el circuit passi com **a mínim** una vegada per cadascun dels vèrtexs. Cal tenir present, doncs, que la solució òptima pel $TSP_g$ no té per què ésser un circuit hamiltonià, ja que el recorregut pot passar més d'un cop per algun dels vèrtexs.

Existeixen també altres generalitzacions del TSP, com per exemple el denominat set-TSP, en el qual l'objectiu és trobar el camí òptim que passi una vegada per cada conjunt de vèrtexs establert. En el cas que cada conjunt de vèrtexs estigués format per un únic vèrtex, ens trobaríem amb el problema del viatjant clàssic.

**Algorismes exactes**

Com ja s'ha comentat a la introducció, la resolució exacta del TSP és un problema NP-Hard, situació que es manté en el cas del TSPg. La solució més directa per resoldre el TSP, que podria ser calcular totes les permutacions possibles de vèrtexs i veure quina és la de menor cost, tindria un temps d'execució d'ordre O(n!), on *n* és el nombre de ciutats a visitar. Aquesta solució és impracticable a partir d'únicament 20 ciutats.

Una de les millors solucions exactes, utilitzant la programació dinàmica de l'algorisme de Held-Karp, resol el problema en $O(n^2 2^n)$ [14]. La millora d'aquests límits de temps és difícil. Per exemple, no ha estat determinat si existeix un algorisme per al TSP que s'executi en un temps d'ordre $O(1.9999^n)$ [15].

**Mètodes heurístics i algorismes d'aproximació**

Els mètodes heurístics utilitzen estratègies que es basen en observacions, de vegades intuïtives, que acostumen a donar bons resultats [12].

En el cas dels algorismes d'aproximació, l'objectiu és donar una solució amb una fita sobre l'error. Per exemple, en el cas de l'algorisme que s'utilitzarà en el projecte, aquest té una raó d'aproximació 1.5, que vol dir que la solució que retorna té, com a molt, un cost igual a 1.5 vegades la solució òptima. Aquest algorisme és el millor dels algorismes d'aproximació que es coneixen per al TSP i per al $TSP_g$ amb una lleugera modificació.

No tots els problemes d'optimització intractables admeten algorismes d'aproximació. En el cas del TSP, està demostrat que no en té quan el graf no compleix la *desigualtat triangular*, és a dir, en algun cas el cost del camí per anar d'un vèrtex *v* a un altre *u* passant per un tercer *s* pot ser inferior al cost d'anar-hi directament de *v* a *u*.

Sí que existiran, encara que el graf no compleixi la desigualtat triangular, algorismes d'aproximació per al $TSP_g$. També cal notar que, com a mínim, una de les solucions del $TSP_g$ serà un circuit hamiltonià si el graf compleix la desigualtat triangular.

## Algorismes utilitzats

Tenint en compte les explicacions dels apartats anteriors, per al projecte s'utilitzaran els següents algorismes extrets de [12] en el cas dels algorismes 1r, 4t i 5è i de [11] l'algorisme 2n:

**Algorisme 1r. Algorisme d'aproximació per al TSP**

El següent algorisme està basat en l'aparellament perfecte de cost mínim. Es requereix un graf que acompleixi la desigualtat triangular i el resultat serà un circuit hamiltonià que proporciona una solució aproximada per al TSP de raó 1.5.

**Algorisme 1r** TSP (G(V,A))

**Requereix:** un graf complet $G$ que verifica la desigualtat triangular

1: Construir un arbre generador de cost mínim, $T$.

2: Sigui $V_s(T)$ el subconjunt de vèrtexs de grau senar de $T$.

3: Sigui $M_s(G)$ el subconjunt d'arestes que permeten connectar de dos en dos els vèrtexs de $V_s(T)$ de manera òptima a $G$.

4: Afegir o duplicar a $T$ les arestes de $M_s(G)$ per obtenir un graf $T_M$.

5: Trobar un circuit eulerià $C_e$ a $T_M$.

6: La solució aproximada pel TSP $H_a$ s'obté prenent els vèrtexs de $G$ segons l'ordre d'aparició a $C_e$.

**Algorisme 2n. Construcció de l'arbre generador de cost mínim**

Per al punt 1 de l'algorisme anterior, s'utilitzarà el denominat algorisme de Kruskal, que obté l'arbre generador simètric de cost total mínim a partir d'un graf G(V, A) simètric amb costos no negatius associats a cada aresta.

L'algorisme de Kruskal es basa simplement en anar seleccionant les arestes de menor cost, saltant-se qualsevol que formi circuit. Aquesta simple tècnica dóna un arbre generador òptim quan hem seleccionat $n$-1 arestes, on $n$ és el nombre de vèrtexs.

**Algorisme 2n** Kruskal (G(V,A))

**Requereix:** un graf simètric $G$

1: Començar amb un graf $G_0(V_0=V, A_0=\varnothing)$.

$k \leftarrow 0$.

2: Ordenar les arestes de $G$ de forma creixent en funció del seu cost.

3: Començant pel cap de la llista ordenada, prendre la primera aresta $a^*$ no usada i que no forma circuit a $G_k$.

$k \leftarrow k + 1$

$G_k \leftarrow G_{k-1} \cup \{ a^* \}$

4: Si $k < n - 1$ tornar a 3.

Si no, acabar.

**Algorisme 3r. Aparellament òptim dels vèrtexs de grau senar**

Per al punt 3 de l'algorisme 1r, l'estratègia utilitzada ha estat calcular de forma exhaustiva tots els possibles aparellaments dels vèrtexs de grau senar per a, posteriorment, seleccionar el que té un cost menor i, per tant, és l'òptim.

Per calcular els diferents aparellaments s'utilitza el següent algorisme, que extreu de forma ordenada totes les permutacions de vèrtexs. Tenint en compte que el nombre total de permutacions és (I-1)!!, essent $I$ el nombre de vèrtexs de grau senar, l'algorisme va creant la matriu de possibles aparellaments per columnes. En cada columna es guarda un vèrtex i cada fila de la matriu seria un aparellament. A la Fig. 3 es pot veure un exemple de com seria l'ordre d'aparellaments obtinguts en els cas de 6 vèrtexs.

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 2 | 4 | 3 | 5 |
| 0 | 1 | 2 | 5 | 3 | 4 |
| 0 | 2 | 1 | 3 | 4 | 5 |
| 0 | 2 | 1 | 4 | 3 | 5 |
| 0 | 2 | 1 | 5 | 3 | 4 |
| 0 | 3 | 1 | 2 | 4 | 5 |
| 0 | 3 | 1 | 4 | 2 | 5 |
| 0 | 3 | 1 | 5 | 2 | 4 |
| 0 | 4 | 1 | 2 | 3 | 5 |
| 0 | 4 | 1 | 3 | 2 | 5 |
| 0 | 4 | 1 | 5 | 2 | 3 |
| 0 | 5 | 1 | 2 | 3 | 4 |
| 0 | 5 | 1 | 3 | 2 | 4 |
| 0 | 5 | 1 | 4 | 2 | 3 |

*Figura 3. Exemple d'execució per a 6 vèrtexs*

**Algorisme 3r** Aparellaments dels vèrtexs de grau senar

**Requereix:** un graf *G*

1: Es crea una matriu permValides de tamany (I * J), on "I" és el nombre de vèrtexs senars i "J" el nombre d'aparellaments, que seria (I-1)!!

2: S'inicialitza numFilesASaltar <- nombre d'aparellaments vàlids

3: Per a cada columna "i":

4: Si la columna és parella:

5: Per a cada fila "j":

6: permValides(i,j) <- vèrtex d'índex menor que no s'hagi col·locat abans a cap columna de la fila "j"

7: Si la columna és senar:

8: Mentre no s'hagin recorregut totes les "J" files:

9: vertexsAcolocar <- vèrtexs que no hagin aparegut abans a la fila "j"

10: nVertexsAcolocar <- nombre de vèrtexs que no hagin aparegut abans a la fila "j"

11: Per a cada element "k" de "vertexsAcolocar":

12: Repetir (numFilesASaltar/nVertexsAcolocar) vegades:

13: permValides (i,j) <- element "k" de "vertexsAColocar"

14: j++;

15: numFilesASaltar <- numFilesASaltar/(nombredeVertexsSenars – i)

**Algorisme 4t. Algorisme d'obtenció d'un circuit eulerià**

Per al punt 5 de l'algorisme 1r, utilitzarem el conegut algorisme de Hierholzer, que permet obtenir un circuit eulerià $C_e$ a partir d'un graf simètric *G*. S'utilitzaran les següents funcions:

*PrimerVertex*($G_{aux}$) retorna un vèrtex de grau parell

*VertexAmbArestes*($G_{aux}$, $C_e$) retorna el 1r vèrtex de *C* que conté arestes a *G*

*Circuit(G, v)* retorna un circuit construït en el graf *G* a partir del vèrtex *v*. Cal anar prenent arestes fins que no es pugui continuar.

*Concatenar*($C_e$, *C, v)* retorna el circuit que s'obté de substituir una aparició de *v* en el circuit $C_e$ per tot el circuit *C*.

*G* – *C* elimina de *G* els vèrtexs que formen part del circuit *C*.

**Algorisme 4t** Hielhorzer(G(V,A))

**Requereix:** un graf eulerià simètric $G$ (opcionalment un vèrtex d'inici $s$).

**Retorna:** un circuit eulerià $C_e$.

1: $G_{aux}$(V,A) ← G(V,A)

2: $v$ ← $PrimerVertex(G_{aux})$

3: $C_e$ ← $\{v\}$

4: **while** V ≠ ∅ **do**

5: $v$ ← $VertexAmbArestes(G_{aux}, C_e)$

6: $C$ ← $Circuit(G, v)$

7: $C_e$ ← $Concatenar(C_e, C, v)$

8: $G_{aux}$ ← $G_{aux} - C$

9: **end while**

10: **return** $C_e$

**Algorisme 5è. Algorisme d'aproximació per al TSP generalitzat**

El següent algorisme està basat en l'algorisme 1r. Es tracta d'assegurar-se que el graf és complet i compleix la desigualtat triangular. Per fer-ho, substituirem el cost de l'aresta entre dos vèrtexs pel cost del camí de cost mínim entre aquells vèrtexs. Posteriorment aplicarem l'algorisme del TSP i finalment tornarem a substituir les arestes que calgui pels circuits corresponents.

**Algorisme 5è** TSPgeneralitzat (G(V,A))

**Requereix:** un graf $G$

1: Construir el graf distància $D$ de $G$, que és el graf complet que compleix la desigualtat triangular, tal que que cada aresta té cost igual al cost del camí de cost mínim a $G$ dels dos vèrtexs que relaciona.

2: *TSP(D)*

3: Transformar el circuit hamiltonià obtingut a D en un recorregut per al graf G substituint les arestes que calgui per camins.

# Capítol 3. Metodologia i eines utilitzades

## Eines utilitzades

### API de Google Maps

L'API de Google Maps és un conjunt de funcions i rutines que permeten interactuar amb els serveis de mapes en línia que ofereix Google. També permet modificacions sobre l'estructura estàndard del mapa i afegir-hi funcionalitat. Per aquest motiu s'utilitza com a base per a la implementació dels diferents algorismes del projecte, ja que permet visualitzar i fer més amigable tant la introducció de dades com el resultat.

La codificació de les funcions està realitzada en JavaScript i la versió amb la què es va començar l'elaboració del projecte és la 3.22. Posteriorment s'han llençat les versions 3.23 i 3.24, que únicament proporcionen modificacions menors de funcionalitat i algunes correccions de bugs.

Un avantatge de l'API és que està disponible per a múltiples plataformes, com Android, IOS i navegadors web. Tot i això, el projecte s'ha realitzat en aquesta darrera plataforma, encara que amb modificacions no gaire complexes es podrien realitzar versions per a les altres dues.

Addicionalment, Google ofereix múltiples serveis web que proporcionen dades geogràfiques com geocodificació, indicacions, elevació, lloc i informació de zones horàries. En el desenvolupament del projecte s'utilitzen els següents:

- "Google Places API Web Service" per implementar l'auto-emplenat del camp d'adreça quan l'usuari vulgui marcar un punt al mapa a partir de la seva denominació.
- "Google Maps Geocoding API" per traduir les adreces a coordenades geogràfiques.
- "Google Maps Distance Matrix API" per calcular la matriu de distàncies entre les diferents ubicacions que marqui l'usuari.

Quant a la política de preus, Google ofereix una llicència gratuïta que cobreix la utilització de la gran majoria de funcions i la publicació del web elaborat, encara que amb limitacions en el nombre de consultes geogràfiques per dia als seus serveis. Per a la realització del projecte s'utilitzarà aquest tipus de llicència, encara que s'ha de ser conscient que si es volgués utilitzar el programa per al càlcul d'un nombre elevat d'instàncies, caldria adquirir una llicència "estàndard" o "Premium", on el cost varia en funció del nombre de peticions de dades que s'enviïn als servidors de Google.

### JavaScript

El JavaScript és un llenguatge de programació interpretat, que es defineix com a orientat a

objectes, basat en prototips, imperatiu, dèbilment tipat i imperatiu [16]. No s'ha de confondre amb el llenguatge Java ja que, tot i adoptar els noms i convencions d'aquest, la seva semàntica i propòsits són diferents. La sintaxi del JavaScript és similar al C, i es considera un dialecte de l'estàndard ECMAScript, del què es va publicar la versió 6 en juny de 2015.

Tot i que tradicionalment ha estat utilitzat en pàgines web HTML per realitzar operacions únicament en el costat de l'aplicació client, actualment és àmpliament utilitzat per enviar i rebre informació del servidor. JavaScript s'interpreta en l'agent d'usuari al mateix temps que les sentències es van descarregant de forma conjunta amb el codi HTML.

Com ja s'ha dit, l'API de Google maps està basada en codi JavaScript i, per tant, és l'alternativa clara en la que es desenvoluparà el codi dels algorismes que composen el projecte.

**HTML**

L'HTML és un llenguatge de marcat que ha esdevingut un estàndard per a l'elaboració de pàgines web. S'utilitza per a definir una estructura bàsica i establir el contingut d'un web. Està basat en elements que tenen un atribut i un contingut, i que s'utilitzen bàsicament amb tres finalitats:

- Estructural, que defineix el propòsit del text i la seva jerarquia.
- D'aparença, sense tindre en compte la funció del text sinó únicament com s'ha de mostrar.
- Hipertextual, que s'utilitza per enllaçar parts del documents amb altres documents (per exemple imatges, altres webs, etc.) o altres parts del mateix document.

En el projecte s'utilitzarà el llenguatge HTML (versió 5) per construir l'estructura bàsica del web que es carregarà amb el contenidor del mapa, així com els diversos botons que implementaran la funcionalitat. També servirà per mostrar algunes de les dades resultants de l'aplicació de l'algorisme i s'integrarà amb el codi JavaScript que conté els algorismes pròpiament dits.

**Adobe Dreamweaver**

Adobe Dreamweaver és un programari propietari que serveix per a la creació de pàgines web, i que pertany a la suite d'Adobe. Facilita la tasca d'edició del codi i s'ha utilitzat en el projecte perquè permet una previsualització del contingut que ha estat útil en determinats moments, així com també proporciona eines per a la depuració del codi que han permès eliminar errors i corregir-los de manera més eficient.

## Metodologia

Per dur a terme aquest projecte es planteja un model basat en el desenvolupament en cascada.

Aquest tipus d'enfoc té com a objectiu ordenar rigorosament les etapes del procés per al desenvolupament del software, de tal manera que s'ha de finalitzar una etapa abans de començar la següent. És un dels primers models de desenvolupament de programari que es van definir i encara és dels més utilitzats avui en dia.

Existeixen diverses variants en la forma de dividir en fases la implementació. Per aquest projecte s'utilitzarà la següent divisió:

1. Anàlisi de requeriments
2. Disseny del sistema i del programa
3. Codificació
4. Proves
5. Verificació

Sobre els avantatges que presenta aquest model s'han de destacar que és un model fàcil d'implementar i entendre, que està orientat a produir una bona documentació i que promou una metodologia de treball efectiva: analitzar abans de dissenyar, dissenyar abans de codificar...

Respecte als desavantatges, la principal és la seva rigidesa, sobre tot quan es tracta de projectes que no estan clarament definits des de l'inici i el seu desenvolupament no segueix una seqüència lineal. Les crítiques més habituals sobre aquest model estan encaminades en aquest sentit, ja que en molts projectes es realitzen canvis en paral·lel al desenvolupament, inclús en fases avançades del mateix, i això suposa que s'hagi tornar a començar des de la fase inicial de disseny amb el consegüent augment de temps i costos. També, qualsevol error en la fase de disseny que es detecti en la fase de proves condueix necessàriament al re-disseny i nova programació del codi afectat, produint-se novament un increment del temps i el cost.

En el cas d'aquest projecte, sembla que pot ser un model suficientment eficient, ja que els objectius del mateix estan establerts des del principi, inclús el fet d'utilitzar les llibreries de Google maps gairebé predetermina les eines amb les quals s'ha de treballar, com s'ha vist en l'apartat anterior. El principal risc resideix en detectar algun error en la fase de proves després de realitzar tot el treball de disseny i codificació. Per minimitzar el treball addicional que suposaria, es realitzarà un cicle complet (disseny, codificació i proves) per cadascuna de les etapes o algorismes a implementar.

# Capítol 4. Anàlisi i disseny

## Anàlisi de requeriments

### Requeriments funcionals

Els requeriments funcionals són els que s'encarreguen de definir què ha de fer el programari. Defineixen l'abast del sistema quant a les accions que ha de realitzar i respecte a la transferència de dades entre les diferents funcions del mateix. En el cas d'aquest projecte, els requisits funcionals són els següents:

- L'usuari podrà introduir les posicions que determinen els vèrtexs del graf mitjançant la interacció gràfica amb un mapa de google maps. Podrà fer-ho bé navegant pel mapa i clicant a la posició escollida o bé utilitzant la barra de cerca per buscar la ubicació al mapa.
- L'usuari podrà moure de lloc o esborrar qualsevol dels punts introduïts, o esborrar-los tots i tornar començar si així ho desitja. En els cas d'esborrar-los tots, el programa guardarà les ubicacions en aquell moment per si hagués estat un error i l'usuari els volgués recuperar.
- El primer dels punts que s'hagi marcat al mapa serà el vèrtex d'inici en el camí resultant de l'algorisme. Si s'hagués esborrat el primer vèrtex per part de l'usuari, el punt inicial seria el segon vèrtex introduït, i així successivament.

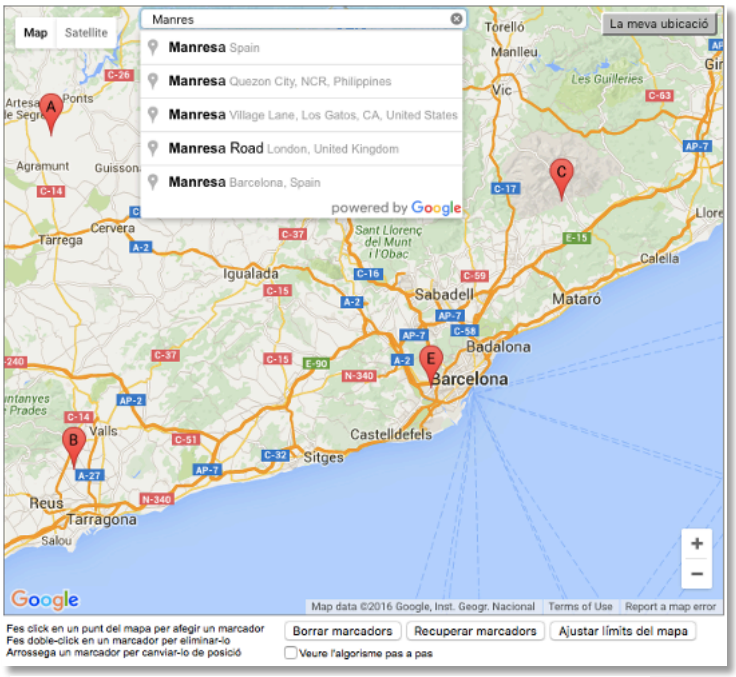



*Figura 4. Introducció i edició dels marcadors*

- Es podran configurar els següents paràmetres de l'algorisme: si es desitja executar l'algorisme pas a pas o obtenir el resultat directament i se es volen utilitzar distàncies en línia recta o les rutes reals més curtes entre els vèrtexs. En aquest darrer cas, es podrà configurar el mode de viatge (caminant, en bicicleta o en cotxe), si es volen evitar autopistes i autovies, i el mode de càlcul (distància o temps de viatge).

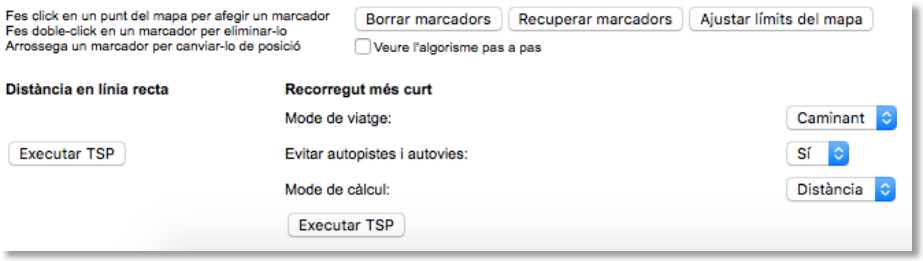



*Figura 5. Configuració dels paràmetres de l'algorisme*

- El programa presentarà el circuit resultant de forma gràfica sobre el mapa de google maps. A més, indicarà el cost del circuit final obtingut i les cotes sobre el resultat òptim.

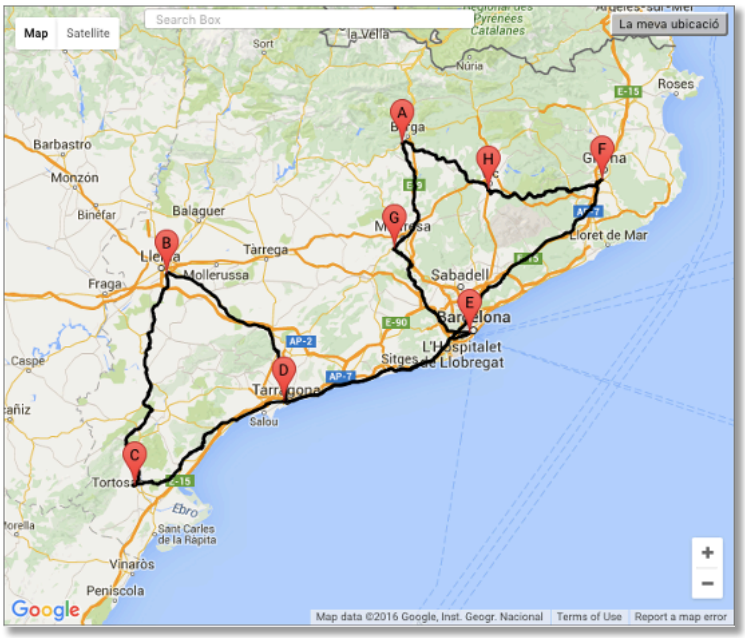



*Figura 6. Representació gràfica del circuit resultant*

- En el cas de seleccionar l'execució pas a pas, el resultat de les diferents etapes s'aniran representant també de forma gràfica sobre el mapa, quan sigui possible, o per pantalla.
- Per realitzar l'estudi del cost computacional, el programa indicarà quant ha trigat l'algorisme en trobar el resultat.

**Requeriments no funcionals**

Els requeriments no funcionals són aquells que defineixen com ha de ser el programari quant a aparença, sensació i manteniment. També poden ser restriccions o atributs dels serveis i funcions que el sistema ofereix, com la fiabilitat, el temps de resposta, la capacitat d'emmagatzemament, ajust a estàndards, etc. En el cas d'aquest projecte, els requisits no funcionals són els següents:

- Tant la introducció de les dades com la visualització del resultat es realitzarà en una mateixa pantalla del navegador web.
- Es dividirà la plana web en tres panells o seccions. El panell principal serà on se situarà el mapa de google maps on l'usuari introduirà les dades i es mostrarà el circuit resultant. El segon panell estarà destinat a la configuració dels paràmetres de l'algorisme. Finalment, el tercer panell serà el destinat a mostrar les dades que el programa retorni després d'aplicar l'algorisme, com el cost del circuit resultant i les seves cotes, o els resultats intermedis en el cas que l'usuari vulgui aplicar l'algorisme pas a pas.
- El codi de l'aplicació haurà de complir l'estàndard HTML, de forma que l'aplicació es pugui utilitzar en qualsevol navegador web.

A continuació es mostra, a tall d'exemple, una execució de l'algorisme, on es pot veure la distribució en 3 panells de la pantalla.

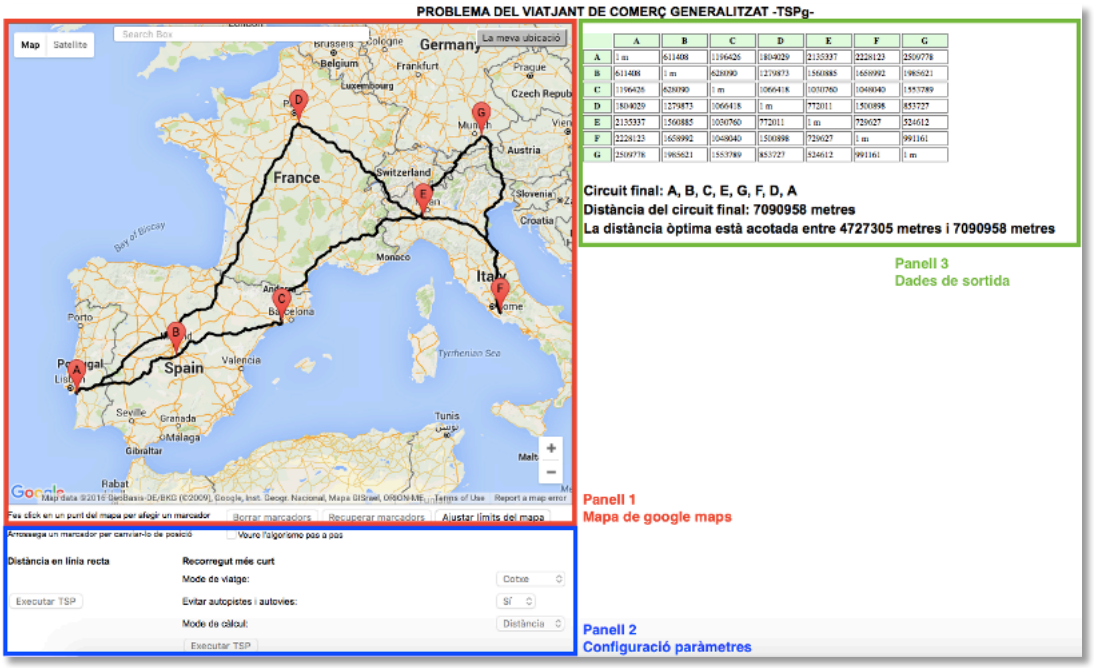



*Figura 7. Divisió en panells de la pantalla*

## Disseny del sistema i del programa

Un cop definits els requeriments al punt anterior, s'observa que existeixen dos blocs de funcionalitats que ha de cobrir el sistema. D'una banda, les referents a la interacció entre l'usuari i el mapa, així com la representació de les dades de sortida. D'altra banda, les referents a l'execució de l'algorisme, incloent-hi l'execució pas a pas i el càlcul del temps d'execució.

El disseny intern de l'aplicació intentarà respectar aquesta divisió lògica, pel què es planteja una elaboració basada en mòduls, amb la intenció que cadascun d'ells s'encarregui d'una part independent de l'algorisme o del control de l'entrada i sortida de dades.

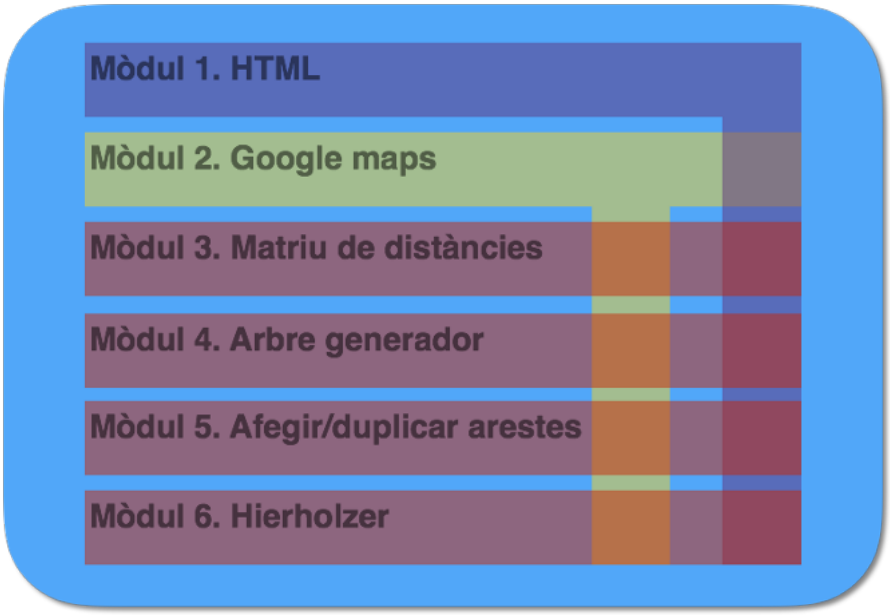



*Figura 8. Esquema de distribució en mòduls del programa*

Un primer mòdul seria el codi HTML encarregat de dibuixar la carcassa de la plana web, amb el seu corresponent codi CSS per manegar l'estil de les diferents parts. No obstant això, hi hauran parts de la web que es modificaran de forma dinàmica mitjançant codi javascript en paral·lel a l'execució de l'algorisme, com per exemple la informació mostrada en el panell 3 (taula de distàncies i resultats dels algorismes) o l'activació/desactivació dels botons del panell 2 en funció de les opcions triades per l'usuari i del punt d'execució en què es trobi l'algorisme.

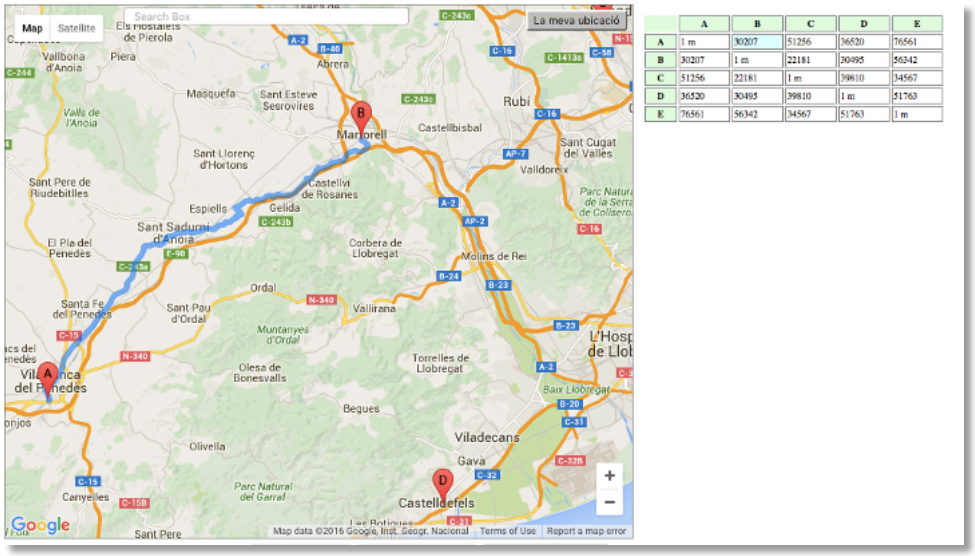
S'ha considerat que la millor forma d'inserir aquestes modificacions transversals és integrant-les en el codi de les funcions que executen aquella part de l'algorisme. Per exemple, la funció que s'encarrega de calcular la matriu de distàncies també la mostrarà per pantalla, o les funcions que comencen a executar cada part de l'algorisme, activaran el botó corresponent a la següent part

en el cas d'una execució de l'algorisme pas a pas.

El segon mòdul tindrà les funcions encarregades de controlar el mapa de google maps. La seva funció principal serà *initializeMap()*, que realitzarà la inicialització del mapa i la connexió amb les llibreries de l'API de google maps. A més, estarà a l'espera de la introducció dels marcadors per part de l'usuari. Aquest mòdul inclou també les funcions necessàries per gestionar els marcadors (introduir-los, esborrar-ne un o tots, restaurar-los en cas d'esborrat accidental i bloquejar-los quan l'usuari ordena el començament de l'algorisme). Finalment, altres funcions per dibuixar línies o rutes, esborrar-les, ajustar els límits del mapa o esborrar-ne tot el contingut, que seran utilitzades de forma transversal per la resta de funcions, també s'inclouen en aquest mòdul.

A partir d'aquí, la funcionalitat d'execució de l'algorisme es dividirà en 4 mòduls independents, que s'executaran de forma seqüencial a partir de l'ordre de l'usuari, detenint-se el programa al final de cada mòdul si es vol una execució pas a pas, o arribant al final de forma automàtica si es vol que l'algorisme s'executi tot de cop.

El primer d'aquests mòduls serà l'encarregat de construir la matriu amb les distàncies (o duració del viatge) entre cada parell de punts, que serà el graf d'entrada a l'algorisme 1r indicat al capítol 2n d'aquesta memòria. Per a fer-ho, utilitzarà les funcions de l'API de google maps necessàries. També mostrarà per pantalla, de forma gràfica en una taula al panell 2, el resultat del càlcul efectuat, permetent a l'usuari visualitzar al mapa qualsevol ruta o línia tot clicant a la casella corresponent.




| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 1 m | 30207 | 51256 | 36520 | 76561 |
| B | 30207 | 1 m | 22181 | 30495 | 56342 |
| C | 51256 | 22181 | 1 m | 39810 | 34567 |
| D | 36520 | 30495 | 39810 | 1 m | 51763 |
| E | 76561 | 56342 | 34567 | 51763 | 1 m |

*Figura 9. Matriu de distàncies amb la ruta entre A i B seleccionada*

El segon mòdul implementarà l'algorisme de Kruskal per tal de construir l'arbre generador de cost mínim (punt 1 de l'algorisme 1r). Es mostrarà de forma gràfica al mapa (amb línies o rutes de color vermell) aquest arbre en el cas que l'usuari hagués seleccionat l'execució pas a pas.

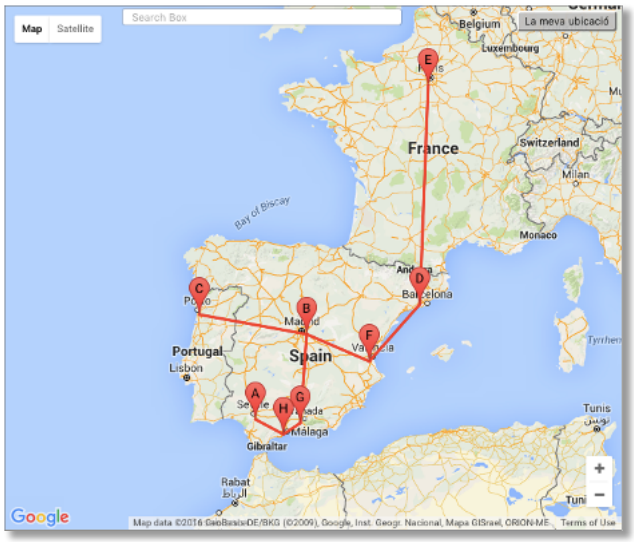



*Figura 10. Exemple d'arbre generador de cost mínim en una execució pas a pas*

El tercer mòdul afegirà o duplicarà les arestes necessàries per tal que no hi hagi vèrtexs de grau senar, després de fer l'aparellament de cost mínim (punts 2, 3 i 4 de l'algorisme 1r). També mostrarà les arestes afegides (de color lila) o duplicades (de color verd) en el mapa si l'usuari ha seleccionat l'execució pas a pas.

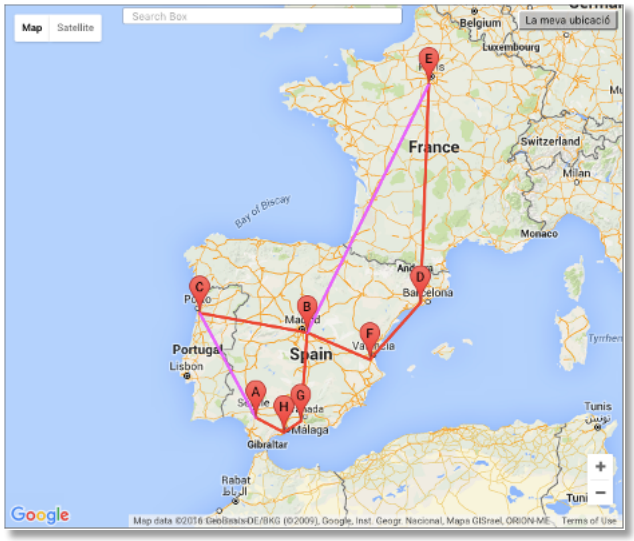
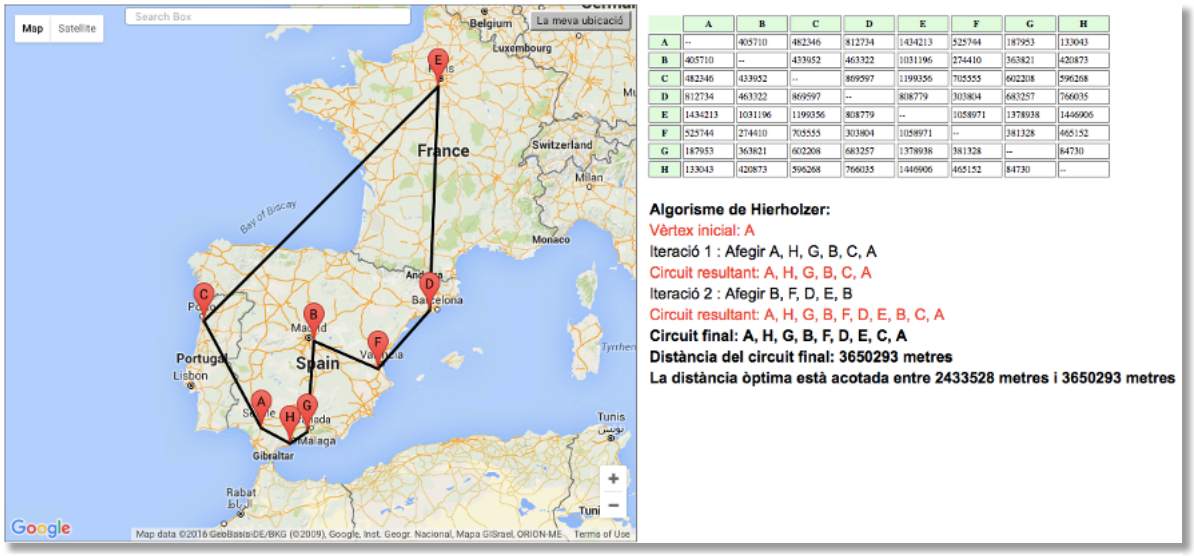



*Figura 11. Arestes afegides a l'arbre en color lila*

Finalment, el quart d'aquests mòduls "algorísmics" implementarà l'algorisme de Hierholzer (algorisme 4t) per tal d'obtenir un circuit eulerià i tractarà el resultat per arribar al circuit resultant final (punt 6 de l'algorisme 1r i punt 3 de l'algorisme 5è). Mostrarà per pantalla l'execució detallada de l'algorisme de Hierholzer si es vol una execució pas a pas, o directament el resultat final en cas contrari. En tot cas, al mapa quedarà dibuixat de color negre el circuit resultat final amb línies o rutes segons s'hagi seleccionat l'opció de línies rectes o rutes en la configuració dels paràmetres.




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | -- | 405710 | 482346 | 812734 | 1434213 | 525744 | 187953 | 133043 |
| B | 405710 | -- | 433952 | 463322 | 1031196 | 274410 | 363821 | 420873 |
| C | 482346 | 433952 | -- | 869597 | 1199356 | 705555 | 602208 | 596268 |
| D | 812734 | 463322 | 869597 | -- | 808779 | 303804 | 683257 | 766035 |
| E | 1434213 | 1031196 | 1199356 | 808779 | -- | 1058971 | 1378938 | 1446906 |
| F | 525744 | 274410 | 705555 | 303804 | 1058971 | -- | 381328 | 465152 |
| G | 187953 | 363821 | 602208 | 683257 | 1378938 | 381328 | -- | 84730 |
| H | 133043 | 420873 | 596268 | 766035 | 1446906 | 465152 | 84730 | -- |

**Algorisme de Hierholzer:**
Vèrtex inicial: A
Iteració 1 : Afegir A, H, G, B, C, A
Circuit resultant: A, H, G, B, C, A
Iteració 2 : Afegir B, F, D, E, B
Circuit resultant: A, H, G, B, F, D, E, B, C, A
**Circuit final: A, H, G, B, F, D, E, C, A**
**Distància del circuit final: 3650293 metres**
**La distància òptima està acotada entre 2433528 metres i 3650293 metres**

*Figura 12. Resultat final en una execució pas a pas*

# Capítol 5. Codificació

En aquest capítol es profunditzarà en la codificació dels diferents mòduls que s'han introduït anteriorment. En cadascun dels mòduls s'explicarà la particularitat del codi i les funcions i procediments més representatius, així com els problemes principals que s'han trobat i hagut de solucionar.

## Mòdul 1. HTML

El codi HTML que genera la carcassa de la pàgina web consta d'una part del codi estàtica, on es declaren els diferents elements fixes, com són: el títol, el contenidor del mapa de google maps (incloent-hi el searchbox per realitzar la cerca d'ubicacions), el text explicatiu de les accions que pot realitzar l'usuari i els elements necessaris (option button i checkbox) per realitzar les configuracions dels paràmetres. En aquesta part també s'implementen els botons per llançar les diferents funcions d'entrada als mòduls que implementen els algorismes. Aquestes funcions es llencen quan ocorre el "click" al botó corresponent, utilitzant listeners.

```
1640  // Create listeners
1641  $(document).ready(function() {
1642      $('#clear-map').click(clearMap);
1643      $('#restore-markers').click(restoreMarkers);
1644      $('#fit-map').click(fitMap);
1645      $('#matrix-calc').click(getDurations);
1646      $('#dist-calc').click(getDistances);
1647      $('#kruskal-calc2').click(calcKruskal2);
1648      $('#kruskal-calc').click(calcKruskal);
1649      $('#arestes-calc').click(calcArestes);
1650      $('#arestes-calc2').click(calcArestes2);
1651      $('#hierholzer-calc').click(calcHierholzer);
1652      $('#hierholzer-calc2').click(calcHierholzer2);
1653      $('#TSP-calc').click(calcTSP);
1654      $('#TSP-calc2').click(calcTSP2);
1655      });
```

Borrar marcadors
Recuperar marcadors
Ajustar límits del mapa
Matriu de camins
Matriu de distàncies línia recta
Kruskal
Kruskal
Afegir arestes
Afegir arestes
Hierholzer
Hierholzer
Executar TSP
Executar TSP

*Figura 13. Funcions que es llencen clicant als diferents botons i correspondència amb els botons que es mostren a pantalla*

La part més complexa d'aquest mòdul, però, és la modificació dinàmica dels elements de la pàgina web en paral·lel a l'execució de l'algorisme. En particular, es realitzen modificacions en tres àrees:

- Activació/desactivació dels botons: es parteix d'una configuració per defecte amb el botó de checkbox "execució pas a pas" desactivat. Això implica que únicament estan visibles els botons "TSP-calc" i "TSP-calc2" que s'encarreguen d'executar tot l'algorisme complet automàticament (per a distàncies en línia recta i rutes, respectivament) i que romandran desactivats fins que l'usuari hagi introduït un mínim de 2 marcadors. El flux de botons que es fan visibles i es desactiven/activen en funció de les accions de l'usuari

es controla en les diverses funcions de la resta de mòduls i segueix el patró que s'indica al diagrama de la Fig. 14.

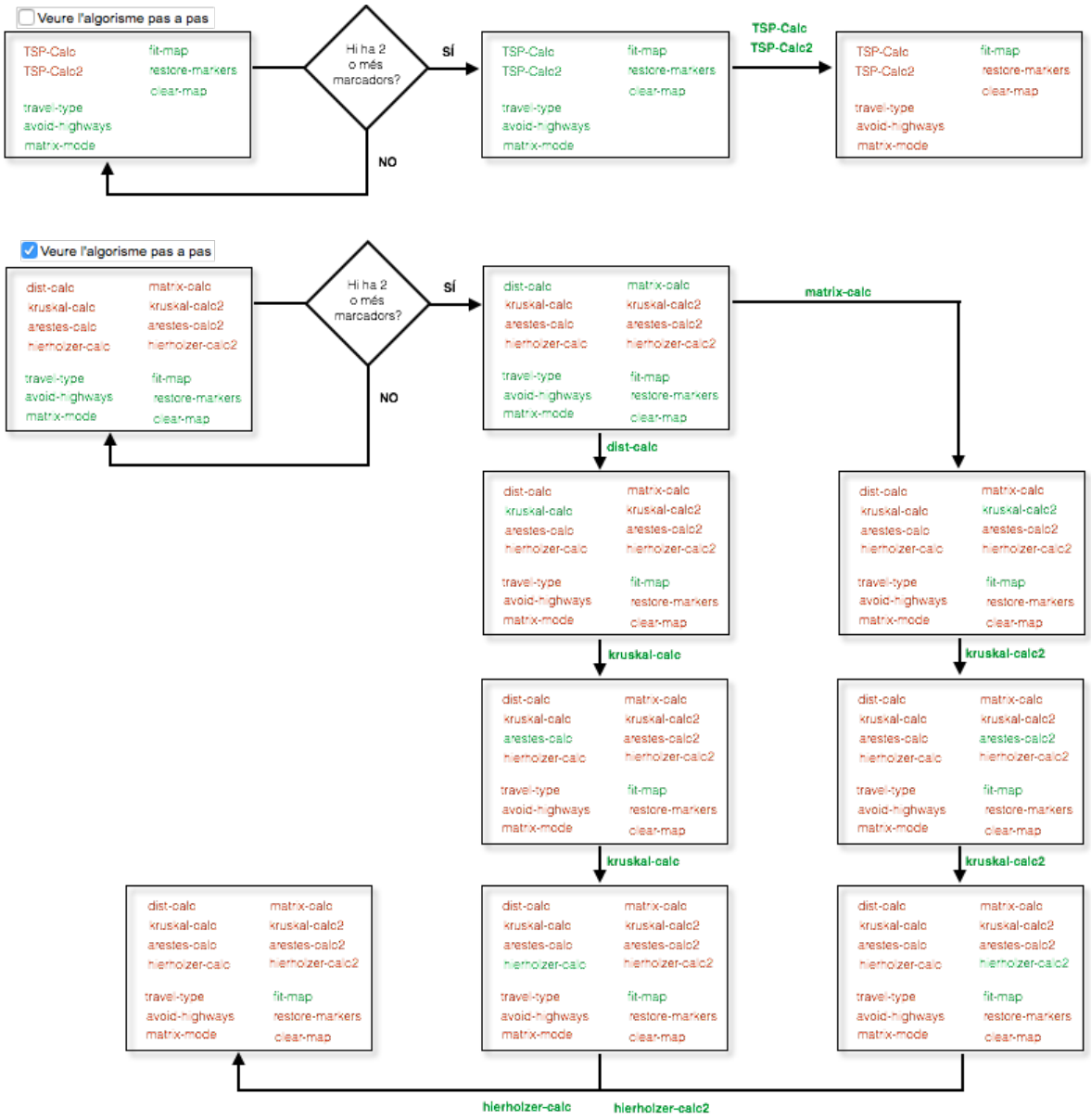



*Figura 14. Diagrama de flux d'activació/desactivació dels botons*

- Taula de distàncies/duracions: quan s'executa el càlcul de la matriu de distàncies o duració (en funció dels paràmetres que seleccioni l'usuari), es mostrarà per pantalla una taula amb les dades obtingudes. Dintre dels procediments "getDistances" o "getDurations" es crearà un element "table" amb el nombre de cel·les adient que s'emplenarà amb el resultat obtingut del càlcul de distàncies/duracions. Quan es

produeix "click" en una cel·la es dispara una funció que remarca de forma gràfica al mapa la línia/ruta entre els marcadors corresponents.

- Dades de sortida: quan s'executen els procediments "calcHierlholzer" o "calcHierlholzer2", es mostrarà per pantalla el circuit resultant de l'algorisme (apart de remarcar-lo en el mapa). També es mostrarà el cost del circuit i les fites respecte al cost òptim. Per fer-ho, es crearà un altre element "table" en aquestes funcions, amb les fileres necessàries per mostrar la informació.

## Mòdul 2. Google maps

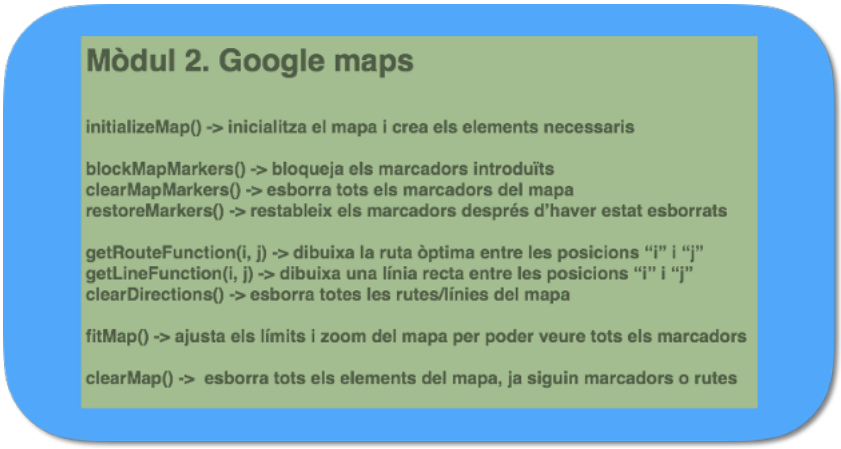



*Figura 15. Procediments del mòdul 2*

En aquest mòdul s'integren els diferents procediments que maneguen la interacció amb el mapa, utilitzant l'API de google maps. El procediment principal d'aquest mòdul és "initializeMap", que realitza les següents tasques:

- Fixa els paràmetres inicials del mapa (ubicació, zoom i tipus de mapa).
- Associa un element google.maps.Map al contenidor del mapa.
- Associa un element google.maps.places.Searchbox al camp de cerca per a què l'usuari pugui utilitzar la funció de cercar ubicacions. Crea els listeners per a què en cas de canvi de l'usuari en l'adreça a trobar, s'actualitzi el centre del mapa.
- Crea un servei de tipus google.maps.DirectionsService en una variable global, que serà l'encarregat de calcular les rutes.
- Crea un servei de tipus google.maps.DirectionsRenderer en una variable global, que serà l'encarregat de dibuixar les rutes sobre el mapa.
- Inicialitza el botó "MyLocation" i crea el listener per a què en el moment de clicar al botó, el mapa es centri en la posició corresponent a la ubicació de l'usuari.

- Crea els listeners per als marcadors. En cas de click sobre el mapa, se'n crea un de nou. En cas de doble click sobre un marcador, s'esborra. Si s'arrossega un marcador, es canvia de posició (es controla la nova posició mitjançant l'esdeveniment "dragend").

La gestió dels marcadors s'ha realitzat per mitjà d'una matriu global anomenada "markers", de tal forma que totes les funcions puguin accedir-ne. Per a cada marcador s'emmagatzema la seva posició en coordenades i el seu nom. En el cas de la inicialització del mapa es poden crear, esborrar i modificar elements; mentre que en la resta de funcions el que es farà és consultar-ne les posicions per a realitzar els diferents càlculs.

A més del procediment d'inicialització, en aquest mòdul s'integren un grup de procediments encaminats a la gestió dels marcadors (esborrar-los, recuperar-los o bloquejar-los) i un altre grup per gestionar les línies i rutes sobre el mapa (dibuixar-les i esborrar-les).

## Mòdul 3. Matriu de distàncies

Aquest mòdul el componen els dos procediments que s'utilitzen per calcular la matriu de costos entre tots els parells de marcadors. S'utilitzarà un o l'altre tenint en compte si l'usuari ha seleccionat que l'algorisme s'apliqui sobre la distància recta entre cada parell d'ubicacions (seria el procediment "getDistances") o sobre la distància/durada de la ruta real més òptima en funció dels paràmetres escollits (en aquest cas s'utilitzaria el procediment "getDurations").

Aquesta estructura de dividir en dos les funcions encarregades del càlcul s'ha utilitzat en els 4 mòduls encarregats d'executar l'algorisme, ja que hi ha vàries diferències en la programació del codi en cada cas, i per claredat s'ha preferit separar-ho que no integrar-ho tot junt dintre de la mateixa funció i internament en cadascuna programar els dos casos.

El resultat tant de "getDistances" com de "getDurations" serà una matriu de 2 dimensions de mida $n$ x $n$ (on $n$ és el nombre de marcadors introduïts per l'usuari). Serà declarada com a variable global i representarà el graf a partir del qual s'aplicaran els posteriors algorismes. En cada posició de la matriu s'emmagatzemarà el cost corresponent (en el cas de la diagonal principal, es posarà el valor "infinit").

L'obtenció dels costos quan es tracta de distàncies en línia recta es realitza mitjançant la funció "computeDistanceBetween" de la llibreria de funcions geomètriques de l'API. En canvi, per al càlcul de rutes exactes s'utilitza el servei "DistanceMatrixService" que retorna directament la matriu de rutes en funció dels paràmetres que se li passin.

En el cas de les rutes, s'ha de tindre en compte que el cost (ja sigui en distància o en temps) no té per què ser el mateix per anar de *a* a *b* que si es fa el recorregut en sentit invers. Per exemple, si s'escull la ruta en cotxe pot ser que en el camí d'anada es vagi per carrers diferents que en el de tornada. En el cas d'escollir el càlcul de la ruta a peu (opció seleccionada per defecte), les diferències són molt menors o inexistent. No obstant això, com és un requisit en l'algorisme de Hierholzer que el graf sigui simètric, en la matriu es posarà el cost mínim d'entre els dos quan fossin diferents.

## Mòdul 4. Arbre generador de cost mínim

Per obtenir l'arbre generador de cost mínim s'utilitza el denominat algorisme de Kruskal (algorisme 2n), que s'implementarà en la funció anomenada "kruskal" (vegeu Fig. 16). Com a entrada se li proporcionen totes les arestes del graf amb el seu cost corresponent i com a sortida la funció donarà les arestes que componen l'arbre generador de cost mínim.

Dintre de l'algorisme de Kruskal, una de les passes és ordenar les arestes segons el cost. Per fer-ho, s'utilitza la funció "orderEdges" que retorna el conjunt d'arestes que se li hagi passat ordenades mitjançant l'algorisme típic d'ordenació "insertion sort".

Finalment, per dibuixar aquestes arestes sobre el mapa (cosa que es farà únicament si l'usuari ha seleccionat l'execució pas a pas), s'utilitzen dos procediments separats depenent de si es tracta de dibuixar línies rectes ("calcKruskal") o rutes ("calcKruskal2").

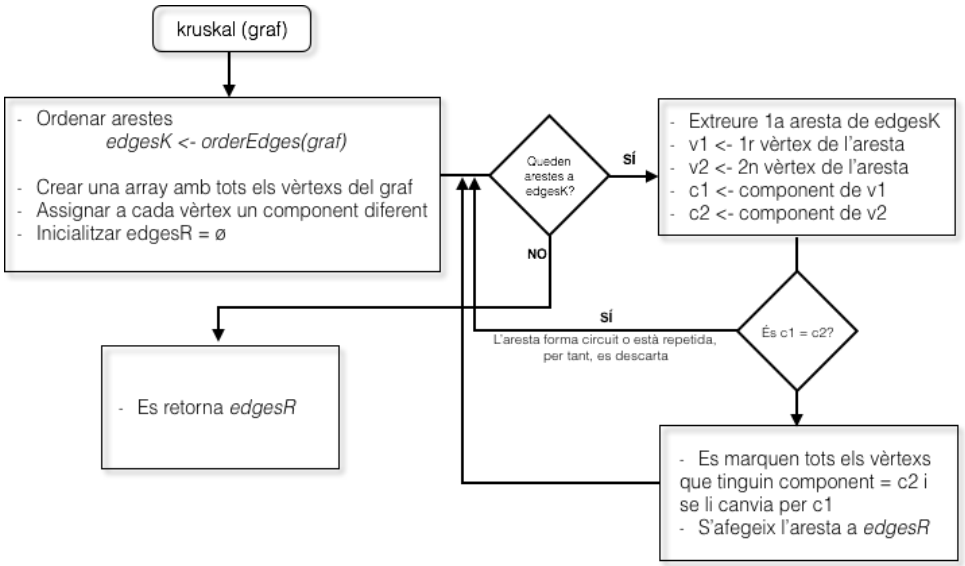



*Figura 16. Representació esquemàtica de l'algorisme de Kruskal*

## Mòdul 5. Afegir/duplicar arestes

Com s'ha comentat, la tasca principal d'aquest mòdul serà trobar les arestes que s'han d'afegir o repetir en el graf (que serà l'arbre obtingut com a resultat del mòdul 4) per tal que aquest no tingui cap vèrtex de grau senar. De manera addicional, en el cas que l'usuari hagués seleccionat l'execució pas a pas, aquest mòdul s'encarregarà de dibuixar-les sobre el mapa utilitzant procediments diferents si es tracta de línies rectes o rutes.

El primer pas per trobar les arestes a afegir és extreure del graf els vèrtexs de grau senar. Per fer-ho, s'ha creat la funció "vertexsSenars" que té com a paràmetre d'entrada el graf i retorna una matriu amb els vèrtexs de grau senar. L'algorisme recorre totes les arestes del graf i registra en una matriu els vèrtexs que van apareixent associats a un comptador amb el nombre de vegades que surt cadascun (si és un vèrtex que no estava a la matriu, l'introdueix amb el comptador a 1, si ja hi estava, n'incrementa el comptador en 1). Després de recórrer tot el graf, s'eliminen de la matriu tots els vèrtexs que apareixen un nombre parell de vegades i se'n retorna la matriu resultant.

A partir d'aquí, es tracta d'aparellar els vèrtexs resultants de forma que el cost total (suma de distàncies o de temps de les arestes resultants) sigui el mínim possible. L'enfoc utilitzat es basa en calcular de forma exhaustiva tots els aparellaments possibles i després seleccionar-ne el de cost mínim. Aquesta ha resultat una tasca complexa de programar, sobre tot per trobar una forma el suficientment eficient per obtenir totes les possibles opcions sense calcular-ne de repetides o innecessàries. S'expliquen de forma més detallada el cost computacional i les passes que s'han fet per optimitzar el rendiment en el capítol 6.

Finalment, la funció implementada s'anomena "aparellamentOptim" i retorna les arestes a afegir després de rebre com a paràmetre el vector de vèrtexs de grau senar. Per calcular el cost una vegada obtinguts tots els possibles aparellaments, simplement s'ha fet un bucle que els recorre i emmagatzema en una variable temporal el de menor cost, que serà el que es retorni com a resultat de la funció després de finalitzar el bucle. La part més complexa és obtenir tots els aparellaments, per a la que s'ha utilitzat l'algorisme 3r explicat en els fonaments teòrics.

## Mòdul 6. Algorisme de Hierholzer

L'algorisme de Hielholzer descrit en els fonaments teòrics (algorisme 4t), es basa en realitzar iteracions mentre no s'hagin visitat totes les arestes del graf. En la primera iteració es parteix d'un vèrtex qualsevol i es construeix un cicle. Si no es compleix la condició d'haver visitat totes

les arestes del graf, es continua amb noves iteracions, creant un altre cicle a partir d'un vèrtex qualsevol, amb les condicions de què ha de pertànyer al cicle que hem construït i ha de tindre únicament arestes pendents de visitar al graf. Es concatena el nou cicle amb el global, substituint cada aparició del vèrtex al cicle global pel nou cicle que em trobat.

Per a la programació del codi, s'han creat les següents funcions que serveixen per resoldre algunes parts de l'algorisme:

- "VertexsAmbArestes(G, C)": se li passen com a paràmetres les arestes del graf que encara no s'han visitat i el circuit que s'està construint. La funció retorna el primer vèrtex pertanyent al circuit i que també té alguna aresta al graf. Per fer-ho, es planteja un doble bucle, que recorre tots els vèrtexs del circuit *C*. Per cadascun del vèrtexs, recorre totes les arestes del graf *G* i, si troba alguna que contingui el vèrtex, retorna aquest vèrtex, aturant l'execució de la funció en aquell moment.
- "concatenar(Ce, C, v)": se li passen com a paràmetres el circuit global, el circuit que es vol concatenar i el vèrtex a substituir. La funció recorre mitjançant un bucle tots els vèrtexs de *Ce*, i substitueix el vèrtex de *Ce* pel circuit *C* quan el vèrtex en qüestió coincideix amb *v*.

Una altra part que ha tingut certa complexitat de programació és trobar un circuit que comenci en un vèrtex donat. Per resoldre-ho, s'ha utilitzat la següent estratègia:

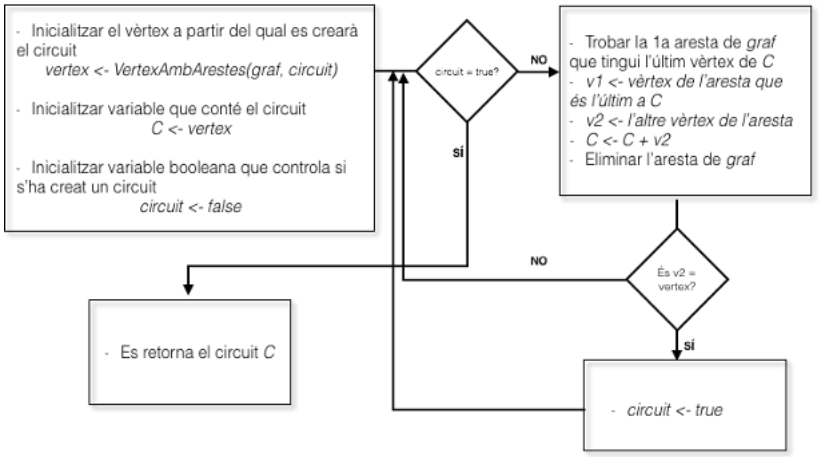



*Figura 17. Creació d'un circuit a partir d'un vèrtex*

# Capítol 6. Cost computacional i optimització dels algorismes

Aquest capítol té com a objectiu l'estudi del cost computacional de les diferents parts de l'algorisme, per identificar si es produeix algun "coll d'ampolla" que limiti l'eficiència del programa i analitzar les possibles opcions per optimitzar-lo en la mesura del possible.

## Algorisme de Kruskal

En l'algorisme de Kruskal, existeixen dos parts que poden limitar el temps d'execució, que són l'ordenació de les arestes per ordre creixent de cost (pas 2) i la comprovació de què no s'ha format un circuit en afegir una aresta a l'arbre (pas 3). Aquest darrer pas es pot realitzar de forma senzilla en un temps d'execució d'ordre lineal respecte al nombre d'arestes (es tracta de realitzar una comparació), pel què no el considerem com un element limitador del rendiment.

Respecte a l'ordenació de les arestes, en el programa s'ha utilitzat un algorisme típic d'ordenació per inserció (insertion sort) que resulta relativament senzill d'implementar de forma iterativa. En general, en la programació d'aquest projecte, es prefereixen els algorismes iteratius respecte als recursius, ja que cada navegador estableix limitacions diferents en quant al nombre de nivells de recursivitat que admet i, per tant, un mateix algorisme podria tenir funcionaments diferents segons el navegador on s'executi (per exemple, donar error amb un nombre determinat d'instàncies que un altre navegador sí que admetria).

L'algorisme *insertion sort* té un temps d'execució d'ordre $O(n^2)$, el que fa que sigui impracticable per a ordenar vectors de grans dimensions, però és més ràpid que d'altres per a casos de vectors petits. En aquest projecte, el que triga en generar-se l'arbre és acceptable per als casos habituals i, en cap cas, és el limitador del temps d'execució total. Per exemple, en una matriu de 150 vèrtexs, el temps de generació de l'arbre és gairebé instantani (menor d'un segon).

## Algorisme de Hierholzer

L'algorisme de Hierholzer, que s'utilitza per trobar el circuit eulerià, és força eficient. De fet, es pot obtenir un temps d'execució lineal respecte al nombre d'arestes (quadràtic respecte el nombre de vèrtexs en el pitjor cas) si es fa una implementació que utilitzi una estructura de dades similar a una doble llista enllaçada per mantenir el vector d'arestes que incideixen a cada vèrtexs i no han estat utilitzades, la llista de vèrtexs en el circuit actual que tenen arestes no utilitzades i el circuit mateix.

Utilitzant aquesta estructura de dades, les operacions individuals de l'algorisme (trobar arestes no utilitzades a partir d'un vèrtex, trobar un nou vèrtex d'inici per al circuit i connectar dos circuits que comparteixen un vèrtex) es poden fer en temps constant cadascuna [17].

En les proves realitzades tampoc ha resultat ser un factor limitador del temps total d'execució. Per exemple, en el cas de 150 vèrtexs, l'execució ha estat pràcticament instantània.

## Aparellament òptim dels vèrtexs de grau senar

L'estratègia utilitzada per calcular quin és l'aparellament òptim passa per calcular de manera exhaustiva tots els possibles aparellaments i, posteriorment, seleccionar el que tingui un cost total menor. Aquest darrer punt no suposa cap problema respecte al cost computacional, ja que es pot efectuar de manera paral·lela a la generació dels diferents aparellaments, simplement comparant si el nou calculat té un cost menor al de cost mínim fins al moment.

Se sap que el nombre total d'aparellaments possibles és (n-1)!!, on *n* és el nombre de vèrtexs de grau senar. Donat que aquest càlcul sí que suposa una limitació en el rendiment de l'aplicació, l'objectiu és obtenir un procediment que calculi els aparellaments com a màxim en un temps d'aquest ordre.

**1r intent. Càlcul de totes les combinacions d'arestes**

En un primer intent per calcular els aparellaments, es va realitzar un procediment recursiu que calculava totes les possibles combinacions sense repetició sobre el total d'arestes possibles, agafades de *n/2* en *n/2*, on *n* és el nombre de vèrtexs de grau senar.

Per exemple, en el cas de 6 vèrtexs, se sap que hi ha 5+4+3+2+1=15 arestes possibles, que són: $(v_1, v_2)$, $(v_1, v_3)$, $(v_1, v_4)$, $(v_1, v_5)$, $(v_1, v_6)$, $(v_2, v_3)$, $(v_2, v_4)$, $(v_2, v_5)$, $(v_2, v_6)$, $(v_3, v_4)$, $(v_3, v_5)$, $(v_3, v_6)$, $(v_4, v_5)$, $(v_4, v_6)$ i $(v_5, v_6)$. A partir d'aquestes 15 arestes, es calculen totes les combinacions sense repetició de 3 arestes i es pren la de menor cost.

El primer problema que va sorgir amb l'aplicació d'aquest procediment va ser la recursivitat. Com ja s'ha comentat, els navegadors estableixen limitacions sobre els nivells de recursivitat que admeten i, en el cas d'aquest algorisme, en el cas del navegador Chrome donava error a partir de 8 vèrtexs de grau senar, impedint l'execució de l'algorisme.

La solució va ser refer l'algorisme traient la recursivitat i transformant-lo en iteratiu. A partir d'aquí, es va fer un càlcul del temps que trigava el procediment en obtenir l'aparellament de cost

òptim. Els resultats es poden veure en la següent taula:

| Nombre de vèrtexs de grau senar | Temps |
|---|---|
| 6 vèrtexs | 1 ms |
| 8 vèrtexs | 15 ms |
| 10 vèrtexs | 436 ms |
| 12 vèrtexs | 30 s |
| 14 vèrtexs | 65 min |

**2n intent. Càlcul de totes les permutacions de vèrtexs**

Per tractar d'optimitzar el temps d'execució, es va realitzar un segon procediment enfocat a calcular totes les permutacions dels vèrtexs i, agafant els vèrtexs de 2 en 2, obtenir les arestes que formen els aparellaments. Estava clar des d'un inici que aquest mètode proporcionaria elements redundants: per exemple, en el cas de 4 vèrtexs les permutacions ($v_1$, $v_2$, $v_3$, $v_4$) i ($v_1$, $v_2$, $v_4$, $v_3$) són diferents però representen les mateixes arestes. Per tant, es feia necessari anar descartant durant l'execució de l'algorisme les permutacions que representaven un aparellament ja calculat anteriorment.

El principal avantatge que semblava que podia proporcionar aquest procediment és que les permutacions s'anaven generant de forma ordenada i previsiblement no caldria calcular-les totes, detenint l'execució de l'algorisme quan ja s'haguessin obtingut (n-1)!! aparellaments diferents.

En la següent taula es presenten els temps d'execució obtinguts amb aquest segon procediment. Per als casos de pocs vèrtexs el temps és una mica millor que amb l'anterior procediment, però es dispara en el cas de 14 vèrtexs. Donat que la darrera permutació que proporciona un aparellament diferent no apareix fins molt tard, el nombre de càlculs a realitzar és major que en el mètode utilitzat per obtenir les combinacions d'arestes.

| Nombre de vèrtexs de grau senar | Temps |
|---|---|
| 6 vèrtexs | 1 ms |
| 8 vèrtexs | 7 ms |
| 10 vèrtexs | 231 ms |
| 12 vèrtexs | 32 s |
| 14 vèrtexs | 128 min |

**3r intent. Càlcul de les permutacions que representen un aparellament diferent**

Tot i que el procediment de l'apartat segon suposa un temps d'execució més elevat a partir dels 12 vèrtexs, sí que semblava que el càlcul de les permutacions podia ser més eficient que el de les combinacions si s'aconseguien evitar els elements redundants.

El tercer procediment, que és el que finalment s'ha implementat en el projecte, obté els (n-1)!! aparellaments possibles sense haver de calcular les n! permutacions i descartar les redundants. L'algorisme està explicat en els fonaments teòrics (algorisme 3r) i obté uns temps d'execució força millors que els dos mètodes anteriors, permetent arribar a calcular l'aparellament de 18 vèrtexs en un temps inferior al que els altres procediments trigaven en fer-ho per 14 vèrtexs:

| **Nombre de vèrtexs de grau senar** | **Temps** |
|---|---|
| 6 vèrtexs | 1 ms |
| 8 vèrtexs | 2 ms |
| 10 vèrtexs | 9 ms |
| 12 vèrtexs | 54 ms |
| 14 vèrtexs | 399 ms |
| 16 vèrtexs | 7,4 s |
| 18 vèrtexs | 4 min |

# Capítol 7. Conclusions

En aquest capítol es valorarà si els objectius que s'havien proposat a l'inici del projecte s'han aconseguit, així com les possibles limitacions i opcions de millora que s'han identificat.

## Objectius aconseguits

Els dos objectius principals del projecte, que eren implementar un algorisme d'aproximació per resoldre el $TSP_g$ i avaluar-ne la complexitat, es poden donar per satisfets.

La utilització de les llibreries de google maps ha permès que la interacció amb l'usuari es realitzés de forma intuïtiva i gràfica, així com la visualització del circuit resultant, que també s'ha aconseguit plasmar de manera visual en el mapa.

Per altra banda, la utilització de la llibreria ha facilitat el càlcul del graf de distàncies. S'han programat unes opcions de configuració que permeten a l'usuari triar entre el càlcul de distàncies en línia recta o de rutes reals (en aquesta segona opció, també es pot configurar el mode en el que es vol fer el viatge –caminant, bici o cotxe- i si el cost es vol obtenir en temps o en distància).

De manera addicional, s'ha implementat un mode "pas a pas", que permet observar el resultat intermedi que van proporcionant els diferents algorismes i il·lustren els diferents passos que es donen per arribar al resultat final.

Finalment, s'ha realitzat un estudi del temps d'execució de les diferents parts de l'algorisme, identificant la més costosa i fent una optimització del codi.

## Limitacions

Respecte a la utilització del programa, la principal limitació ve donada per la pròpia API de google maps, ja que el nombre de peticions en el càlcul de distàncies està limitat per a la llicència gratuïta. Si bé en el cas del càlcul de distàncies en línia recta aquesta limitació no suposa un greu problema (permet fer un màxim de 2500 peticions diàries); en el cas del càlcul de rutes reals, que es fa mitjançant el servei "Distance Matrix", la limitació de la llicència gratuïta és més estricta, ja que existeix un màxim en la mida de la matriu de 100 elements per a cada consulta.

Quant a temps d'execució, la principal limitació és en funció del nombre de vèrtexs de grau senar que tingui l'arbre generador òptim. A partir de 20 vèrtexs de grau senar, l'algorisme trigarà més

d'una hora en donar un resultat. S'ha de dir que el nombre de vèrtexs senars de l'arbre generador no guarda correlació amb el nombre de vèrtexs introduït per l'usuari, sinó amb la seva distribució, com es pot veure en els dos exemples de la Fig. 18., on es mostren els aparellaments de 2 i 14 vèrtexs de grau senar respectivament, quan l'usuari ha introduït 16 vèrtexs en ambdós casos.

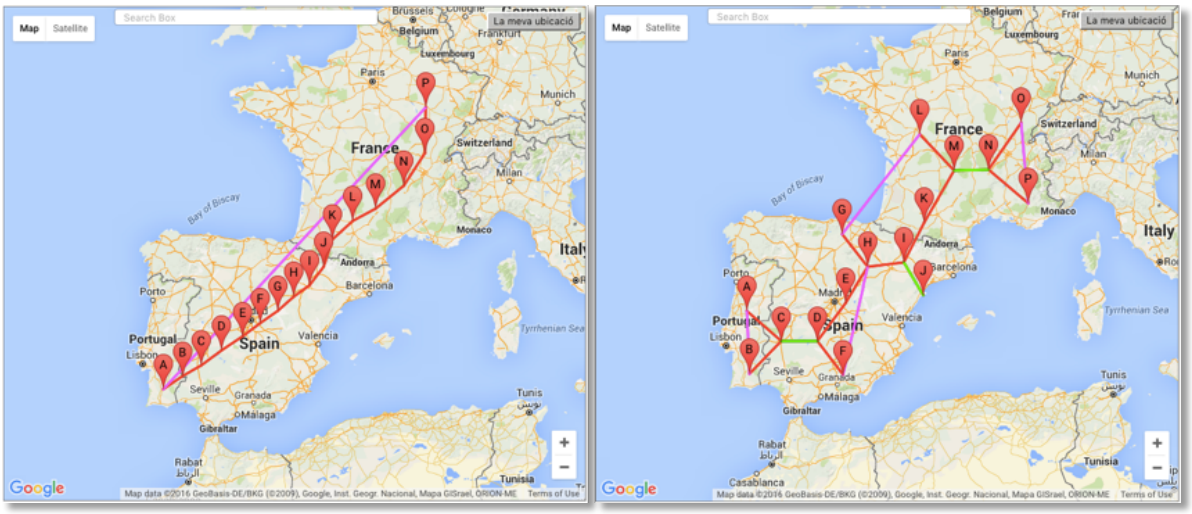



*Figura 18. Comparativa d'aparellaments amb 16 vèrtexs introduïts per l'usuari*

## Opcions de millora

Enllaçant amb l'apartat anterior, una possible línia de millora del projecte seria millorar el temps d'execució de l'algorisme que s'encarrega de trobar l'aparellament òptim. Les possibles opcions serien trobar una manera de calcular-lo sense haver de trobar totes les opcions de manera exhaustiva o bé millorar l'eficiència de l'algorisme que calcula totes les possibilitats i elegeix la de menor cost.

Una altra part de l'algorisme susceptible d'optimització seria el d'ordenació de les arestes en funció del cost. Com ja s'ha comentat, s'ha realitzat una implementació basada en el "insertion sort", però es podria estudiar si d'altres algorismes d'ordenació poden ser més eficients per a nombres elevats d'instàncies.

Finalment, quant a la limitació de la pròpia API de google maps si es vol utilitzar la llicència gratuïta, es podria estudiar si es pot salvar la limitació del servei "Distance Matrix" utilitzant algun altre servei de la pròpia API.

## Referències bibliogràfiques


[1] *Icosian Game.*

http://www.daviddarling.info/encyclopedia/I/Icosian_Game.html

[2] *Biografia de Sir William Rowan Hamilton.*

http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/Hamilton.html

[3] *Biografia de Thomas Penyngton Kirkman.*

http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/Kirkman.html

[4] *Biografia de Karl Menger.*

http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/Menger.html

[5] *The Traveling Salesman Problem.*

http://www.math.uwaterloo.ca/tsp/index.html

[6] *Concorde TSP Solver.*

http://www.math.uwaterloo.ca/tsp/concorde.html

[7] *Millenium problems. P vs NP Problem.*

http://www.claymath.org/millennium-problems/p-vs-np-problem

[8] *LKH.*

http://www.akira.ruc.dk/~keld/research/LKH/

[9] *Solving the Traveling Salesman Problem Using Google Maps and Genetic Algorithms.*

http://www.theprojectspot.com/tutorial-post/solving-traveling-salesman-problem-using-google-maps-and-genetic-algorithms/9

[10] *Graph theory.* Wikipedia.

https://en.wikipedia.org/wiki/Graph_theory

[11] *Grafs: fonaments i algorismes.* Josep M. Basart i Muñoz, 2a Edició, Barcelona: Servei de publicacions UAB.

[12] *Capítol 5: Grafs eulerians i grafs hamiltonians.* Matemàtica discreta curs 2014-2015. Enginyeria informàtica. Escola d'enginyeria. UAB.

[13] *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Michael Garey & David S. Johnson. W.H. Freeman and Company. 1979.

[14] *Dynamic Programming Treatment of the Travelling Salesman Problem*. Bellman, R., J. Assoc. Comput. Mach., 1962.

[15] *Exact Algorithms for NP-Hard Problems. A Survey*. Woeginger, G.J., Combinatorial Optimization – Eureka, You Shrink! Lecture notes in computer science, vol. 2570, Springer, 2003.

[16] *JavaScript*. Wikipedia.

https://es.wikipedia.org/wiki/JavaScript

[17] *X.1Algorithms for Eulerian Trails*. Fleischner, Herbert, Eulerian Graphs and Related Topics:

Part 1, Volume 2, Annals of Discrete Mathematics, 1991

## Resum

Aquest projecte implementa, en una aplicació que s'executa en un navegador web, un algorisme per resoldre de forma aproximada el problema del viatjant generalitzat. S'utilitzen els serveis de l'API de google maps per facilitar el càlcul de les distàncies i rutes, així com s'empren les capacitats gràfiques que proporciona per visualitzar en un mapa tant les dades d'entrada com el circuit resultant. Addicionalment, es realitza un estudi de la complexitat de l'algorisme i del seu temps d'execució.

## Resumen

Este proyecto implementa, en una aplicación que se ejecuta en un navegador web, un algoritmo para resolver de forma aproximada el problema del viajante generalizado. Se utilizan los servicios de la API de google maps para facilitar el cálculo de las distancias y rutas, así como las capacidades gráficas que proporciona para visualizar en un mapa tanto los datos de entrada como el circuito resultante. Adicionalmente, se realiza un estudio de la complejidad del algoritmo y de su tiempo de ejecución.

## Abstract

This project implements, in a web-navigator based application, an approximation algorithm for the generalized Travelling Salesman Problem. Google maps API services are used to facilitate the calculation of the distances and routes, as well as the graphic capacities of the API are used to show in a map the input data and the resulting circuit. In addition, a study of the algorithm complexity and execution time is performed.