PROJECTE DE RECERCA BÀSICA
PAC3 — Tercera Prova d'avaluació continuada

Gener 2013.

Marcos Fernádez Callejo.

Format: Article Científic.
Revistes a on es podria publicar: Applied Software Computing, Journal of the Operational Reasearch Society.

Biblioteca de l'heurística constructiva Clarke and Wright combinada amb simulació Monte Carlo per resoldre el problema d'adreçament de vehicles.

Marcos Fernández Callejo.

Resum El problema d'adreçament de vehicles (VRP) tractar de construir una col·lecció de rutes per a una sèrie de vehicles que subministren bens a un conjunt de clients. És un problema ben conegut d'optimització combinatòria que necessita de mecanismes heurístics per resoldre instàncies amb centenars de clients i diferents característiques. En aquesta recerca es presenta una biblioteca de programari lliure basada en l'heurística constructiva d'en Clarke & Wright combinada amb simulació Monte Carlo que permet, de manera simple, generar solucions al problema d'adreçament de vehicles. Aquesta metodologia construeix solucions competents en poc temps i sense necessitat d'ajustar o configurar paràmetres previs. La biblioteca pot integrar-se fàcilment en projectes que resolguin variants del VRP amb canvis mínims sobre el codi. Aquest es troba ben documentat i accessible de manera lliure a través de la xarxa. Els resultats computacionals indiquen que es resolen instàncies VRP en temps i costos significativament inferiors a altres propostes de la mateixa família.

Paraules clau Adreçament de vehicles Heurística Simulació Monte Carlo Optimització Biblioteca

1. Introducció

El transport per carretera és el mecanisme logístic que més es fa servir al món. En les darreres dècades els recursos petrolífers han patit una pujada pronunciada dels seus preus, això ha implicat un encariment proporcional del cost associat al transport per carretera. Per altre banda, aquesta estratègia de distribució de bens, implica un greu impacte mediambiental degut a les emissions de gasos contaminants que s'ha vist agreujada pel ràpid creixement del model econòmic actual. Altres costos derivats són la congestió de trànsit al voltant de les grans ciutats i la seguretat a les carreteres. L'eficiència és imprescindible per reduir l'impacte d'aquests costos. Es necessari desenvolupar models i metodologies pràctiques que ajudin a dissenyar estratègies que maximitzin el rendiment del transport per carretera.

Aquest article presenta una biblioteca de programari lliure per resoldre el problema d'adreçament de vehicles (VRP). Pensada per que sigui el cor resolutiu per altres variants del VRP. Aquest problema tracta de construir una sèrie de rutes amb el menor cost possible per a una flota de vehicles que ha de satisfer les demandes d'un conjunt de clients. L'adreçament de vehicles és un problema ben conegut de tipus NP-Complexe, la categoria més difícil segons la teoria de complexitat computacional. Tot i que aquest problema ha sigut estudiat durant dècades encara atrau l'atenció de molts investigadors degut a les múltiples variants que se'n poden trobar. Això es tradueix en el desenvolupament de nous algoritmes, mètodes d'optimització i heurístiques que resolen problemes de combinatòria. Aquestes propostes han evolucionat des de models purs d'optimització, capacitats per resoldre instàncies de mida petita o mitjana, fins l'ús d'heurístiques i metaheurístiques que poden proporcionar propostes gairebé òptimes per problemes de mida mitjana o gran amb limitacions complexes (Cordeau et al, 2004).

Existeixen nombrosos paquets de programari per generar solucions a problemes d'optimització combinatòria. Molts d'ells són projectes de desenvolupament tancat però existeix un ventall de propostes de codi obert. Aquí es citaran tres de les més conegudes. La primera és *Choco* (Jussien et al, 2008) una biblioteca Java per resoldre problemes de satisfacció de restriccions (CSP) i programació restrictiva (CP). *Choco* permet modelar un problema a partir de la definició d'un conjunt ampli de variables i restriccions per posteriorment aplicar un algorisme de resolució. La

segona és *JGAP* (Meffert, Klaus et al. 2010) una biblioteca Java basada en algoritmes genètics com a metaheurística per resoldre instàncies VRP, aquesta metodologia és una de les que genera millors solucions però amb la necessitat d'afinar paràmetres específics. Finalment, *VRPH* (Chris Groër et al 2010) una biblioteca desenvolupada en C++ basada en heurístiques de cerca local per resoldre el problema d'adreçament de vehicles, la seva filosofia es basa en construir una solució inicial viable per posteriorment aplicar-hi heurístiques de cerca local que permetin millorar-la.

En aquest article es presenta una biblioteca Java basada en l'heurística constructiva d'en Clarke & Wrigth (Clarke and Wright,1964) combinada amb simulació Monte Carlo. Aquesta metodologia va ser desenvolupada per (Juan, A, et al. 2010) i té com a principals característiques la seva simplicitat i efectivitat. Permet obtenir solucions sense haver de configurar complexos paràmetres en un temps d'execució molt ràpid. Tot i que la seva qualitat no és tan acurada com altres estratègies més complexes, la seva senzillesa i agilitat la permet adaptar-se a multitud d'escenaris. Aquesta biblioteca pot solucionar diferents variants del problema de l'adreçament de vehicles desenvolupant mòduls perifèrics que la facin servir com a nucli de resolució.

La resta de l'article s'estructura de la següent manera: En primer lloc, és fa un revisió de la literatura existent relacionada amb l'adreçament de vehicles. En segon lloc, es descriu l'algorisme que forma part del nucli de la biblioteca, a continuació s'exposen una sèrie de modificacions algorítmiques i tècniques que permeten augmentar el rendiment. En tercer lloc, es presenten el resultats obtinguts sobre un conjunts d'instàncies estàndard per finalment discutir-los, fer una conclusió de la recerca i proposar línies futures.

2. Revisió de la literatura.

El problema de l'adreçament de vehicles, The Vehicle Routing Problem (VRP), consisteix en el disseny òptim d'un conjunt de rutes que parteixen i retornen a un node central visitant una sèrie de clients distribuïts al llarg d'un espai geogràfic. Aquest disseny esta subjecte a una sèrie de restriccions que dependran de la casuística de cada problemàtica. En general trobarem limitacions com, la capacitat dels vehicles, la longitud de les rutes, temps d'entrega, relacions de precedència entre clients, etc.

Aquest és un problema al que s'ha d'enfrontar diàriament milers de distribuïdors arreu del món i té un impacte econòmic i ecològic molt significatiu. Podem trobar exemples d'aquest problema entre distribuïdors de diaris, aliments o begudes com la llet. El transport per carretera es veu afectat diàriament per una sèrie de circumstàncies més enllà de recórrer una distància quilomètrica, com són les congestions de trànsit, la pujada del preu del petroli o les conseqüències mediambientals. Hi ha una gran necessitat de construir mecanismes eficients que ajudin a escollir les millors alternatives en aquest camp.

El problema de l'adreçament de vehicles ha sigut explorat mitjançant diferents metodologies al llarg de les darreres dècades. Aquesta metodologies han evolucionat des de les primeres propostes basades en mètodes d'optimització purs, com la programació lineal, fins l'ús d'heurístiques i metaheurístiques capacitades per donar solucions properes a l'òptim per problemes de grans dimensions.

A (Laporte G., 2007) es descriu les diferents tècniques aplicades a l'adreçament de vehicles. Es comença fent una descripció de les metodologies exactes com la programació lineal, programació dinàmica o algoritmes de Branch and Bound. Les metodologies exactes tenen la capacitat de resoldre problemes de mida petita (al voltant dels 100 nodes) però pateixen problemes a l'hora de resoldre instàncies de mida superior i a més tenen uns requeriments computacionals molt elevats. Per altre banda són mecanismes massa especialitzats que fan difícil la seva abstracció i

adaptació a diferents realitats.

A partir de les limitacions dels algoritmes exactes l'esforç en recerca es va orientar cap a les heurístiques. Aquestes tenen la capacitat de ser més flexibles i per tant es poden adaptar de manera simple a les diferents variants del problema d'adreçament de vehicles. Les heurístiques clàssiques es poden dividir en heurístiques constructives i heurístiques de millora. L'adjectiu de clàssiques prové del fet que aquestes tècniques no cerquen millors combinacions en un espai de solucions on no es milloren resultats parcials o que les propostes no siguin funcionals. Com veurem més endavant, les tècniques metaheurístiques sí que cerquen aquest espai de solucions obtenint millors resultats.

L'heurística constructiva més popular és l'algoritme d'estalvi desenvolupat per (Clarke and Wright, 1964). El seu funcionament es basa en el concepte d'estalvi, aquest es calcula com el guany en cost per visitar una sèrie de clients amb un vehicle envers l'opció de dedicar un vehicle per client. L'heurística construeix en primer lloc tantes rutes com clients o instàncies tingui el problema, cadascuna d'aquestes rutes comença al punt central, visita el client i torna. A continuació es construeix una llista d'arestes, cada aresta connecta dos clients i té un valor d'estalvi associat que és igual a la suma d'anar del client A al punt central i del punt central al client B menys el cost d'anar del client A al client B. La llista d'arestes s'ordena de major a menor valor d'estalvi.

Cada iteració del algorisme consisteix en fusionar una ruta que acaba al client A amb un altre ruta que comença al client B eliminant el camí de retorn de A al punt central i del punt central a B. Cada iteració seleccionarà l'aresta de major estalvi associat que no hagi sigut vinculada amb cap altra ruta. Per fusionar les rutes s'haurà de complir que els nodes que composen les arestes es trobin als extrems de la ruta, directament connectats al punt central, i que la demanda total dels clients de la nova ruta creada no superi la capacitat de càrrega del vehicle. L'heurística de Clarke and Wright proporciona bons resultat per problemes amb instàncies petites o mitjanes, però necessita mecanismes de millora per manegar problemes de molta grandària.

Una segona heurística clàssica constructiva és l'heurística de pètals posada en pràctica per (Gillett i Miller, 1974). Aquesta es basa en el fet que moltes aplicacions resolen el problema d'adreçament de vehicles amb un conjunt de rutes que tenen una estructura de pètals formant una flor. L'algorisme s'inicia construint una gràfica que representa la distribució geogràfica del clients. El punt de distribució es col·loca al centre de la gràfica, d'aquí parteix un vector paral·lel al eix X. Aquest vector comença a rotar en sentit antihorari afegint en una ruta els nodes amb els que va intersecant. Quant la demanda total dels clients sigui superior a la càrrega del vehicle es tanca la ruta, i es crea una nova mentre el vector continua la seva rotació. La limitació principal d'aquesta heurística es que no genera rutes que es puguin interseccionar, per tant es perd l'opció d'explorar altres combinacions. Per alguns autors pot ser un bon mecanisme de partida.

Existeixen dues categories d'heurístiques clàssiques de millora. La primera és la *intra-route heuristics*. Aquest mecanisme treballa per separat cada ruta generada fent petites modificacions en l'ordre de visita del clients. Aquest intercanvis poden rebaixar els costos totals de cada viatge i per tant de la solució completa. La segona és la *inter-route heuristics*, on la filosofia es troba en l'intercanvi de clients entre les diferents rutes propostes amb l'objectiu de cercar noves combinacions que impliquin un guany en el cost total de la solució.

Les heurístiques clàssiques es fan servir com a punt de partida per generar una solució inicial a ser millorada per metodologies metaheurístiques. Com s'ha comentat anteriorment, aquestes estratègies exploren camins que en un principi no semblen donar bons resultats o que fins hi tot son irrealitzables però que poden acabar derivant un solució viable amb un cost total inferior. Existeix un ampli ventall d'algoritmes metaheurístics que es poden classificar com metodologies

de cerca local, cerca de poblacions i mecanismes d'aprenentatge.

Els mecanismes de cerca local tracten de fer petites modificacions a les rutes configurades mitjançant operacions d'intercanvi i desplaçament de clients o conjunts de clients. Els més populars són cerca Tabu descrit per (Taillard E, 1993), la cerca de veïnatge variable desenvolupat per (Mladenović i Hansen,1997) o l'algorisme record-to-record travel proposat per (Li et al. 2005).

La cerca de poblacions es basa en els principis fonamentals biològics que es donen a nivell genètic quant es produeixen cèl·lules sexuals (esperma i òvul). A partir de la combinació de cromosomes d'un individu es construïxen noves cèl·lules mitjançant un mecanisme de replicació que pateix una sèrie de modificacions, gairebé aleatòries, denominades mutacions i recombinacions. Una mutació es un intercanvi d'una base determinada d'un cromosoma, la recombinació és dona quant un bloc de bases d'un cromosoma es talla i s'adapta en una nova localització. Aquesta filosofia es pot aplicar al problema d'adreçament de vehicles on cada ruta proposta representaria a un cromosoma. Aquestes rutes es replicarien generant noves propostes que implicarien mutacions (canvis en l'ordre de visita dins d'una mateixa ruta) i recombinacions (intercanvis de clients entre diferents rutes). L'objectiu final és molt semblant al mecanismes de cerca local, provocar intercanvis i desplaçaments. Els algoritmes genètics són el mecanisme més habitual en cerca de poblacions.

Els mecanismes d'aprenentatge inclouen xarxes neuronals i l'algorisme de la colònia de formigues. En el cas de les xarxes neuronals la filosofia també s'adapta de la biologia mitjançant l'estudi de les les xarxes bioelèctriques del cervell formades per neurones i les sinapsis d'aquestes. El problema de l'adreçament de vehicles es formula a traves d'una xarxa de clients, (neurones), que intercanvien informació sobre els seus costos i demandades (sinapsis). D'acord amb (Laporte G., 2007) aquesta modelització del VRP no va donar bons resultats en el seus inicis i roman abandonada.

L'algorisme de la colònia de formigues pretén formular la problemàtica del VRP imitant el comportament de les formigues, que són capaces de trobar el camí més curt entre una font d'aliments i el seu niu de manera col·lectiva. Les formigues intercanvien informació deixant feromones (rastre del seu olor) al llarg dels camins explorats. Aquesta informació compartida permet establir la ruta més curta per a tota la colònia. En el model del VRP aquesta feromona es representa mitjançant un mecanisme de memòria cau on s'emmagatzemen els vèrtexs que apareixen amb més freqüència a les millors solucions. L'algoritme pot recordar els millors vèrtex i per tant hi ha més probabilitats que els inclogui a la solució final.

El ventall de metaheurístiques és molt ampli i complicat de classificar ja que existeixen propostes que tracten d'adaptar les propietats de les diferents formulacions. Es creen per tant, solucions hibrides que s'inicien amb una heurística clàssica que es va millorant aplicant diferents estratègies metaheurístiques fins a construir una solució quasi-òptima.

El cor de la biblioteca implementada es basa en el desenvolupament del mecanisme hibrid definit per (Juan, A, et al. 2010) on es construeix una solució inicial mitjançant una heurística constructiva i s'aplica un mecanisme de cerca local per refinar la solució. La senzillesa d'aquesta estratègia fa que sigui adaptable a multitud d'escenaris, una característica essencial per a una biblioteca que pretén ser una eina estàndard.

3. Algorisme hibrid Clarke and Wright amb simulació Monte Carlo

A (Juan, A, et al. 2010) es defineix l'estratègia de combinació de l'heurística clàssica constructiva d'en Clarke and Wright amb la simulació Monte Carlo (MCS). La simulació de Monte

Carlo és un conjunt de tècniques que fa servir números aleatoris i distribucions estadístiques per resoldre problemes estocàstics o deterministes. MCS és profitosa per generar solucions numèriques per a problemes que no poden ser resols de manera eficient mitjançant aproximacions analítiques (Guimarans, D. 2011).

Com s'ha descrit a la secció 2, l'heurística d'en Clarke and Wright comença creant una solució trivial on cada client es servit per un vehicle dedicat. A continuació s'executa un procés iteratiu on es combinen algunes de les rutes de la solució trivial amb l'objectiu de reduir el seu cost total de manera que un únic vehicle serveixi els clients d'una ruta fusionada. El criteri per ajuntar rutes segueix el principi d'estalvi. Aquest es defineix com un valor que s'assigna a l'arc que connecta un parell de nodes o clients. El valor es calcula com la reducció en el cost total degut al fet de distribuir bens al dos clients amb un sòl vehicle en comptes de fer-ho amb dos dedicats.

L'algorisme d'en Clarke and Wright construeix un llista d'arcs que connecten dos clients ordenada pel seu valor d'estalvi. A cada iteració es selecciona l'arc amb el major estalvi assignat d'acord amb el següents criteris:

1. Els nodes que defineixen l'arc han de ser adjacents al centre distribuïdor (node central).
2. La fusió dels nodes es viable.

El comportament de l'algorisme es pot modificar amb l'objectiu de crear noves solucions diferents. Fent una cerca sobre l'espai de solucions possibles es pot trobar una amb un cost menor que el proporcionat per Clarke and Wright. Mitjançant la metodologia MCS es pot variar lleugerament l'ordre de la llista d'arcs amb valor d'estalvi mantenint la seva filosofia però construint una proposta diferent.

El re-ordenament de la llista d'arcs s'implementa gràcies a un probabilitat de selecció aleatòria que segueix una distribució geomètrica per respectar el principi d'estalvi. (Els arcs amb valors més elevats han de tenir més probabilitats de ser escollits). Així doncs, un cop s'ha ordenat la llista d'arcs es crea per cadascú una nova posició generant un nombre aleatori sota una distribució geomètrica. A partir de la nova llista d'arcs es continua el procés iteratiu de fusió de rutes.

3.1 Pseudo-codi

La figura 1 mostra el cos principal de l'heurística d'en Clark and Wright. Per començar, es construeix la solució trivial (més costosa) i es genera la llista ordenada d'arcs en funció del seu valor d'estalvi (línies 1 i 2). A continuació, s'itera sobre cada arc de la llista, en primer lloc s'extreuen els nodes dels extrems de l'arc (línies 4 i 5), en segon lloc es cerquen les rutes associades a cada node i candidates a fusionar-se (línies 6 i 7) i finalment es fusionen si compleixen les limitacions o criteris de combinació.

Per ajuntar les rutes en una nova, primerament s'ha de recuperar els arcs de cada ruta que estan directament connectats al node central (distribuïdor) i eliminar-los de les rutes (línies 9 – 12). Tot seguit, es coordinen ambdues rutes al sentit d'orientació de l'arc per finalment concatenar-les (línies 13 – 15). En acabat, s'actualitza la solució eliminant les dues rutes fusionades i afegint la nova. (línies 16 -18).

```
procediment construirSolucio ( nodes, limitacions )
1. solucio = construirSolucioTrivial (nodes);
2. llistaArcs = generarLlistaArcsOrdenda(nodes); //Ordenat per valor d'estalvi
3. per cada arc de llistaArcs →
4.    nodeInici = retornarNodeInici(arc);
5.    nodeFinal = retornarNodeFinal(arc);
6.    rutaNodeInici = retornarRuta(nodeInici);
```

```
7.   rutaNodeFinal = retornarRuta(nodeFinal);
8.   si es satisfan les condicions de fusió →
9.      arcExteriorInici = retornarArcExterior(rutaNodeInici);
10.     arcExteriorFinal = retornarArcExterior(rutaNodeFinal);
11.     rutaNodeInici = eliminarArcRuta(arcExteriorInici,rutaNodeInici);
12.     rutaNodeFinal = eliminarArcRuta(arcExteriorFinal,rutaNodeFinal);
13.     rutaNodeInici = coordinarRutaSentitArc(rutaNodeInici,arc);
14.     rutaNodeFinal = coordinarRutaSentitArc(rutaNodeFinal,arc);
15.     novaRuta = concatenarRutes(rutaNodeInici,arc,rutaNodeFinal);
16.     solucio = eliminarRuta(rutaNodeInici);
17.     solucio = eliminarRuta(rutaNodeFinal);
18.     solucio = afegirRuta(novaRuta);
19.  final si
20. final per cada
21. retornar solucio;
final
```

Figura 1. Heurística constructiva Clarke & Wright

La figura 2 mostra la metodologia de Monte Carlo per generar posicions amb caràcter aleatori amb l'objectiu de generar un nou ordenament de la llista d'arcs del problema. Aquest procediment retorna un vector on cada posició indica l'index de l'arc que s'ha de recuperar de la llista. Afegint la crida a aquest mètode després de generar la llista d'arcs ordenada segons el criteri d'estalvi (línia 2 figura 1) i iterant sobre aquest vector s'aconsegueix que l'heurística d'en Clarke and Wright generi noves solucions.

Per tornar a assignar un ordre a la llista d'arcs es treballa sobre un vector que emmagatzema l'ordre original i un altre vector que guarda el nou ordre aleatori. En primer terme es fa un recorregut equivalent al nombre d'arcs, per cada iteració es recupera una nova posició mitjançant la generació d'un numero aleatori sota una distribució geomètrica (línia 3) i s'assigna al vector de nou ordenament (línia 4). Tot seguit s'elimina la nova posició recuperada de la localització que ocupava a l'ordenació original endarrerint una unitat les posicions posteriors (línies 5 -12). Cal notar que aquest procediment torna un nou ordenament invertit degut a que primer es calculen les posicions aleatòries pels arcs amb menys valor d'estalvi i es situen a les primeres localitzacions del nou vector d'ordenament.

```
procediment reordenacioAleatoria(llistaArcs)
1. numeroArcs = #elements(llistaArcs); //Torna el total d'elements a la llista
2. mentre indexA < numeroArcs →
3.  novaPosicio = retornarPosicioAleatoria(numeroArcs – indexA, distribucio);
4.  ordenamentAleatori [indexA] = ordenamentOriginal [novaPosicio];
5.  desde indexB = novaPosicio fins a indexB < numeroArcs – indexA – 1 →
6.   ordenamentOriginal [indexB] = ordenamentOriginal[indexB + 1];
7.   indexB = indexB + 1;
8.  final desde
9.  indexA = indexA + 1;
10.final mentres
11.retornar ordenamentAleatori;
final
```

Figura 2. Re-ordenament d'acord amb generació de nombres aleatoris sota distribució geomètrica

A la figura 3 es mostra el codi de l'heurística d'en Clarke and Wright modificat per que treballi amb la metodologia de Monte Carlo. El codi és pràcticament idèntic a la figura 2 però amb dos modificacions cabdals, la primera es la crida al procediment que reordena aleatòriament la llista d'arcs (línia 3) i la segona és la manera de recórrer el nou vector d'ordenació (línia 4) que es fa des de l'element final fins l'inicial.

```
procediment construirSolucioAleatori ( nodes, limitacions )
1. solucio = construirSolucioTrivial(nodes);
2. llistaArcs = generarLlistaArcsOrdenda(nodes); //Ordenat per valor d'estalvi
3. ordenamentAleatori = reordenacioAleatoria(llistaArcs)
4. desde arc=ordenamentAleatori[final] fins a arc=ordenamentAleatori[inici] ➔
5.   nodeInici = retornarNodeInici(arc);
6.   nodeFinal = retornarNodeFinal(arc);
7.   rutaNodeInici = retornarRuta(nodeInici);
8.   rutaNodeFinal = retornarRuta(nodeFinal);
9.   si es satisfan les condicions de fusió ➔
10.     arcExteriorInici = retornarArcExterior(rutaNodeInici);
11.     arcExteriorFinal = retornarArcExterior(rutaNodeFinal);
12.     rutaNodeInici = eliminarArcRuta(arcExteriorInici,rutaNodeInici);
13.     rutaNodeFinal = eliminarArcRuta(arcExteriorFinal,rutaNodeFinal);
14.     rutaNodeInici = coordinarRutaSentitArc(rutaNodeInici,arc);
15.     rutaNodeFinal = coordinarRutaSentitArc(rutaNodeFinal,arc);
16.     novaRuta = concatenarRutes(rutaNodeInici,arc,rutaNodeFinal);
17.     solucio = eliminarRuta(rutaNodeInici);
18.     solucio = eliminarRuta(rutaNodeFinal);
19.     solucio = afegirRuta(novaRuta);
20.  final si
21. final desde
22. retornar solucio;
final
```

Figura 3. Heurística constructiva Clarke & Wright combinada amb simulació Monte Carlo

La figura 4 mostra el mètode principal, primer es recuperen els nodes (centre de distribució i clients) i les restriccions del problema (línies 1 i 2). Després es genera una solució bàsica Clarke and Wright (línia 3). Tot seguit es van construint noves solucions mitjançant el mètode Clarke and Wright amb estratègia MCS mentre no es compleix el criteri d'aturada (normalment temps). Es retorna la solució amb el cost menor de totes les construïdes.

```
procediment principal()
1. nodes = retornarNodesProblema();
2. limitacions = retornarLimitacionsProblema();
3. solucioFinal = construirSolucio( nodes, limitacions );
4. mentre no es satisfà el criteri d'aturada ➔
5.  solucioActual = construirSolucioAleatori ( nodes, limitacions );
6.  si retornarCost(solucioFinal) > retornarCost(solucioActual); llavors
7.     solucioFinal = solucioActual;
8.  final si
9. final mentres
10.retornar solucioFinal
final
```

Figura 4. Mètode principal de cerca de solucions

4. Millores per potenciar el rendiment

La biblioteca que es presenta en aquest article presenta un conjunt de millores algorítmiques i tècniques que fan augmentar la quantitat de solucions generades per unitat temporal en comparació a la proposta definida per (Juan, A, et al. 2010). Aquesta millora suposa un creixement en les probabilitats de trobar solucions d'alta qualitat ja que quant més propostes es construeixin més fàcil serà cercar combinacions de rutes que redueixin el cost total del problema.

4.1 Coordinació de sentit de les rutes i arcs a convergir

Una de les operatives que tenen més cost computacional són els bucles que implementen una sèrie d'operacions sobre un conjunt d'elements d'un vector. Normalment aquests mecanismes

tenen un comportament temporal que depèn directament del nombre d'elements, en conseqüència, a mesura que creix el total de membres del vector cau el rendiment temporal del bucle.

Com s'ha vist a la secció 3, a l'hora de fusionar rutes gràcies a un arc connector escollit pel seu valor d'estalvi es coordinen el sentit de les rutes a fusionar a l'orientació que tingui l'arc. Aquesta estratègia es força costosa, en el millor dels casos no hi haurà cap operació de gir però per la resta pot succeir que s'hagi de girar tots els arcs d'una de les rutes o en el pitjor dels casos girar tots els arcs de les dues rutes a concatenar.

El mètode de convergència de rutes i arc connector implementat a la biblioteca gira sempre els elements més petits adaptant-se al sentit de l'element més gran. El casos possibles són els descrits a la figura 5.

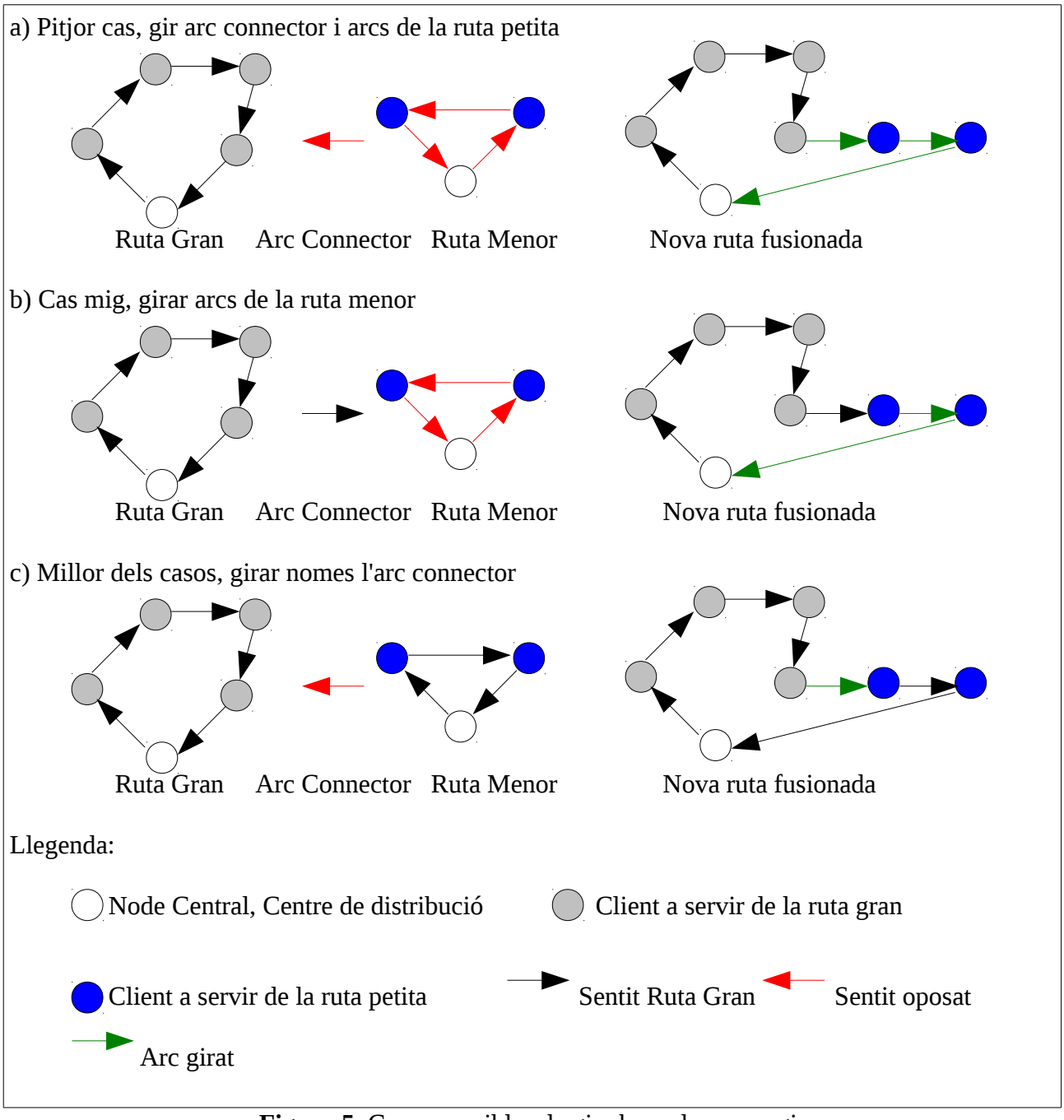


Figura 5. Casos possibles de gir abans de convergir

Com es pot veure a la Figura 5, s'evita haver de girar tots els arcs de les dues rutes i el pitjor cas (gir de l'arc i la ruta petita) equival al millor cas de gir del codi descrit a la secció 3 (Gir d'una de les rutes).

4.1.2 Pseudo-codi

La figura 6 mostra el mecanisme de coordinament de sentit en la fusió de rutes. En primer lloc es defineix la ruta gran i petita segons les grandàries de les rutes associades a cada node extrem de l'arc connector (línies 1-7). En segon lloc es recupera el sentit de la ruta major i es giren l'arc connector i la ruta menor si s'escau (línies 8-11). En acabat es concatenen els tres elements que formen la nova ruta combinada.

```
procediment fusioRutes(rutaNodeInici,arc,rutaNodeFinal)
1.si (numeroArcs(rutaNodeInici) >= numeroArcs(rutaNodeFinal)); llavors
2.   rutaGran = rutaNodeInici;
3.   rutaPetita = rutaNodeFinal;
4.sino
5.   rutaGran = rutaNodeFinal;
6.   rutaPetita = rutaNodeInici;
7.final si
8.sentitRuta = retornarSentit(rutaGran);
9.si (sentitRuta <> retornarSentit(arc)); llavors
10.  girar(arc);
11.final si
12.si sentitRuta <> retornarSentit(rutaPetita)); llavors
10.  girar(rutaPetita);
11.final si
12.rutaFusionada = concatenarRutes(rutaGran,arc,rutaPetita);
13.retornar rutaFusionada;
```

Figura 6. Combinació de rutes optimitzada

4.2 Ús de la biblioteca de contenidors de Java per millorar el rendiment del reordenament aleatori de la llista d'arcs

Una de les operatives que més temps computacional requereix és l'establiment d'un nou odre aleatori de la llista d'arcs construïda segons el criteri d'estalvi. Com s'ha descrit a la figura 2, aquest procediment requereix per a cadascun dels arcs la generació d'un número aleatori seguint una distribució geomètrica que proporcioni un nou posicionament a la llista. Aquesta operativa s'ha d'implementar controlant que no es generi més d'un posicionament per a cada arc ni que s'assigni la mateixa localització a més d'un arc. Aquestes dues restriccions es controlen mitjançant dos bucles niats, el mestre processa cada nou posicionament i l'esclau elimina l'antiga localització mitjançant desplaçament d'elements.

El cost computacional del bucle esclau és de l'ordre O(d), sent d el nombre d'elements a desplaçar. Donat que això succeïx per cada arc de la llista tenim que la complexitat del bucle niat és O(n*O(d)) sent n el total d'arcs.

Existeixen dos contenidors de la biblioteca estàndard de Java amb capacitat de representar llistes, *ArrayList* i *LinkedList*. La diferència principal es troba en el mecanisme d'accés al elements, la classe contenidora *ArrayList* fa servir un taula d'accés aleatori mentre que *LinkedList* implementa un estructura de llaços dobles entre els seus elements. En el primer cas, el temps per fer una operativa sobre un element és independent de la seva posició en contraposició al sistema de llaços dobles. Ambdós mecanismes implementen estructures internes de suport que permeten realitzar l'operació d'eliminació d'elements en un ordre de complexitat temporal proper a O(1). Però aquest rendiment s'aconsegueix quant el context en el qual treballen el contenidors és positiu.

El contenidor *ArrayList* és més eficaç que *LinkedList* quan la quantitat d'elements no és molt elevada i els accessos als mateixos no segueixen un patró definit. Sota aquestes circumstàncies l'estructura d'accés aleatori té un rendiment molt elevat. El contenidor *LinkedList* ofereix un comportament superior a *ArrayList* quant s'ha de manegar un nombre d'elements elevats i l'ordre d'accés segueix una tendència ordenada. En aquest context, el desplaçament entre els nodes cap a endavant o enrere és molt més ràpid que cercar nodes mitjançant un mecanisme d'accés aleatori.

La construcció d'un nou ordenament de la llista d'arcs es basa en la generació aleatòria sota una distribució geomètrica. D'aquesta manera s'aconsegueix que les noves localitzacions dels arcs a la llista no difereixin gaire del seu ordre inicial. Aquest escenari s'adapta al millor context de la llista doblement enllaçada ja que hi ha una tendència d'accés ordenada sobre llistes amb un nombre elevat d'elements, la intenció és resoldre instàncies VRP de gran mida.

Si el criteri d'ordenació es relaxés, fent que la generació aleatòria de noves posicions seguís una distribució uniforme, amb instàncies VRP de mida reduïda, llavors l' *ArrayList* seria més eficaç que el *LinkedList*.

A la taula A.1 és mostren els resultats d'un test comparatiu entre l'ús de la classe *ArrayList* i *LinkedList* sobre un conjunt de llistes de llargada mitjana (50 nodes) i gran (100 i 500 nodes). Els resultats expressen un comportament millor del mecanisme doblement enllaçat, especialment per problemes de mida gran i generació aleatòria de nombres sota distribució altament geomètrica. Aquesta comparativa s'ha implementat sobre un maquinari Intel i5, 3.10 Ghz i 8Gb amb un sistema operatiu Ms. Windows 7.

	Numero Nodes	50		100		500	
	Mida Llista Arcs	1.225		4.950		124.750	
	Crides remove	10.000		10.000		100	
	Beta	*ArrayList*	*LinkedList*	*ArrayList*	*LinkedList*	*ArrayList*	*LinkedList*
Comportament gairebé uniforme	0,01	**3s**	**3s**	**32s**	**17s**	**190s**	**5s**
Comportament geomètric baix	0,10	**3s**	**2s**	**32s**	**8s**	**190s**	**3s**
Comportament geomètric mig	0,20	**3s**	**1s**	**32s**	**7s**	**190s**	**3s**
Comportament altament geomètric	0,30	**3s**	**1s**	**32s**	**7s**	**190s**	**3s**

Taula A1. Rendiment en segons dels contenidor Java *ArrayList* i *LinkedList*.

En conclusió, l'escenari que presenta el mecanisme híbrid d'en Clarke and Wrigth amb simulació Monte Carlo s'adapta al millor dels cassos del contenidor *LinkedList.* D'aquesta manera es pot aconseguir un rendiment proper a O(*n**O(*1*)) a l'hora de reordenar la llista d'arcs segons el criteri d'estalvi.

4.2.1 Pseudo-codi

La figura 7 mostra el nou mètode de ordenament aleatori de la llista d'arcs fent servir la funció *remove* de la classe *LinkedList* de la biblioteca de Java. Això suposa trencar el bucle niat i fer la crida *remove* de cada arc que ja s'hagi assignat al nou ordenament (línia 5).

```
procediment reordenacioAleatoria(llistaArcs)
1. numeroArcs = #elements(llistaArcs); //Torna el total d'elements a la llista
2. ArrayList javaOrdamentOriginal = ordenamentOriginal;
2. mentre indexA < numeroArcs →
3.  novaPosicio = retornarPosicioAleatoria(numeroArcs – indexA, distribucio);
4.  ordenamentAleatori [indexA] = javaOrdamentOriginal [novaPosicio];
5.  javaOrdamentOriginal.remove(novaPosicio)
6.  indexA = indexA + 1;
7.final mentres
8.retornar ordenamentAleatori;
final
```

Figura 7. Reordenament de la llista d'arcs sense bucle niat

4.2.2 Codi biblioteca

La figura 7.1 mostra la implementació a la biblioteca del pseudo-codi del mètode descrit a la figura 7. L'objecte *LinkedList ordreOriginal* s'inicialitza amb les posicions originals (línies 9-12). A continuació es fa un recorregut sobre tots els arcs calculant un nou posicionament aleatori, assignant-lo al nou vector d'ordenament i eliminant l'arc reordenat de l'objecte *LinkedList ordreOriginal* (línies 16 -21). Finalment, es retorna el nou vector d'ordenament (línia 23).

```
01    public int[] calcPositionsArrayFast()
02    {
03        //Array a retornar amb el nou ordenament
04        int[] nouOrdre = new int[numeroArcs];
05
06        // Llista LinkedList amb l'ordenament original
07        LinkedList<Integer> ordreOriginal = new LinkedList<Integer>();
08
09        for (int i = 0; i < numeroArcs; i++)
10        {
11            ordreOriginal.add(i);
12        }
13
14        int pos = 0;
15        // Assignació noves posicions aleatòries
16        for (int i = 0; i < numeroArcs; i++)
17        {
18            pos = getRandomPosition( numeroArcs – i, distribucioGeometrica);
19            nouOrdre[i] = ordreOriginal.get(pos);
20            ordreOriginal.remove(pos);
21        }
22
23        return nouOrdre;
24    }
```

Figura 7.1 Implementació reordenament a la biblioteca.

4.3 Recuperació immediata de la ruta a la que pertany un node

A la secció 3 es descriu l'algorisme d'en Clarke and Wright on un dels passos és la recuperació de les dues rutes a les que pertanyen els nodes extrems de cada arc processat. La biblioteca que es presenta en aquest article realitza aquesta operativa aprofitant les característiques de la programació orientada a objectes. Cada node esta representat mitjançant un objecte de la

classe que defineix las característiques i comportaments d'un node (demanda, localització geogràfica, adjacència al centre distribuïdor o identificador). Incorporant una referència a l'objecte ruta a la qual pertany el node s'aconsegueix un guany en rendiment substancial.

Les línies 6 i 7 de la figura 1 fan referència a un mètode *retornarRuta(node)* que cerca la ruta a la qual està associat un node donat. Si s'implementa una cerca sobre totes les rutes l'ordre de complexitat temporal dependrà del nombre total de rutes, aquesta complexitat es pot reduir significativament si es fa servir un mecanisme de taula associativa amb clau única. La millor estratègia és recuperar la ruta directament a partir d'un membre de la classe node que emmagatzemi la referència. D'aquesta manera, s'estalvia fer cap procediment de cerca i es millora significativament el rendiment de l'algorisme.

5. Resultats computacionals

En aquest article es presenta els resultats computacionals produïts per la biblioteca sobre un conjunt d'instàncies de referència obtinguts de la literatura. Aquestes van ser definides per (Golden, et al. 1998) i són accessibles de manera lliure des del portal web http://code.google.com/p/metavrp**.**

El resultats descrits en aquesta secció han de poder ser reproduïts per un usuari amb valors molt similars. Cal destacar que la versió de la maquina virtual de Java, el sistema operatiu i les característiques de maquinari que es fa servir poden provocar un cert biaix. Totes les dades generades s'han obtingut executant el codi sobre la maquina virtual de Java 1.6.0_26, un sistema operatiu GNU/Linux Ubuntu 10.04 (Kernel 2.6.32-45) amb arquitectura de 32 bits i un microprocessador Intel Core i3 2,27 Ghz amb 3.2 GB de memòria principal. Com que la biblioteca no és multi-fil només s'ha fet servir un nucli en aquests experiments.

Amb l'objectiu de verificar el rendiment i l'eficiència de la biblioteca s'ha comparat els resultants generats amb la proposta de (Juan, A, et al. 2010) que defineix l'algorisme híbrid de Clarke and Wright amb simulació Montecarlo.

La taula 1 esta formada pels següents camps: (1) Número instància del conjunt (Golden, et al. 1998), (2) Número de nodes, (3) Capacitat dels vehicles, (4) Màxima distància permesa per configurar una ruta, (5) Solució base heurística Clarke and Wright CWS, (6) Millor cost de la solució generada per (Juan, A, et al. 2010) [1], (7) Millor cost de la solució generada per la biblioteca [2], (8) Diferència entre els costos (7) – (6).

Problema	Nodes	V. Capacitat	Max.Dist.	CWS	Millor C. (1)	Millor C. (2)	Diferència C.
1	240	550	650	5956,5044	5926,302	5770,488	-155,8139
2	320	700	900	9242,382	9017,858	8696,971	-320,887
3	400	900	1200	12282,903	12282,903	12081,685	-201,218
4	480	1000	1600	16206,975	16048,574	15762,281	-286,293
5	200	900	1800	7378,1494	7082,1533	6953,9067	-128,2466
6	280	900	1500	9358,891	9260,555	9131,598	-128,957
7	360	900	1300	11606,437	11606,437	11358,179	-248,258
8	440	900	1200	13178,818	12893,731	12698,487	-195,244
9	255	1000	999999	733,70984	710,8656	687,31177	-23,55382
10	323	1000	999999	911,42725	911,42725	895,5121	-15,91514
11	399	1000	999999	1143,6272	1143,6272	1125,2661	-18,36109
12	483	1000	999999	1388,4095	1358,9908	1341,3116	-17,6792

13	252	1000	999999	952,74005	945,8203	934,36017	-11,46012
14	320	1000	999999	1221,6871	1210,8391	1198,6964	-12,14269
15	396	1000	999999	1512,6553	1505,4159	1490,0023	-15,41359
16	480	1000	999999	1774,6835	1774,6835	1774,6835	0
17	240	200	999999	771,7049	771,7049	762,79926	-8,905635
18	300	200	999999	1069,286	1069,286	1057,0035	-12,2825
19	360	200	999999	1465,9994	1465,9994	1460,4857	-5,513697
20	420	200	999999	1963,4697	1958,0153	1951,6254	-6,38993
Mitja					4947.25	4856,63	-90,62

Taula 1. Comparació dels millors costos.

La Taula 2 mesura el temps necessari per resoldre l'heurística constructiva Clarke and Wright (CWS). El valors són els següents: (1) Número instància del conjunt (Golden, et al. 1998), (2) Número de nodes, (3) Temps en segons per resoldre CWS per (Juan, A, et al. 2010) [1], (4) Temps en segons per resoldre CWS per la biblioteca i (5) percentatge de millora de la biblioteca.

Problema	Nodes	T.CWS (1)	T.CWS (2)	% Millora T.
1	240	0,216046664	0,048259307	77.6
2	320	0,027985013	0,002673735	90.4
3	400	0,025084714	0,004030839	83.9
4	480	0,039508591	0,005880205	85.1
5	200	0,003593075	0,00084762	76.4
6	280	0,009361234	0,001597891	82.9
7	360	0,016175219	0,003353114	79.2
8	440	0,030359071	0,004777423	84.2
9	255	0,004807084	0,001315148	72.6
10	323	0,011202694	0,0023086	79.3
11	399	0,016402139	0,008207881	49.9
12	483	0,027026566	0,006151934	77.2
13	252	0,00631928	0,001087616	82.7
14	320	0,010952927	0,002040776	81.3
15	396	0,022976137	0,003912408	82.9
16	480	0,039595745	0,006048295	84.7
17	240	0,004278703	0,000099598	97.6
18	300	0,015995174	0,00214505	86.5
19	360	0,014407364	0,003231104	77.5
20	420	0,023828577	0,004423151	81.4
Mitja		0,02829529	0,00561958	80.13

Taula 2. Rendiment temporal.

La Taula 3 mostra una comparativa del numero solucions generades per cada instància en 10 segons. Els camps de la taula descriuen: (1) Número instància del conjunt (Golden, et al. 1998), (2) Número de nodes, (3) Propostes de solució generades per (Juan, A, et al. 2010) [1], (4) Propostes de solució generades per la biblioteca [2], (5) Diferència de solucions (4) – (3) i (6) reducció de cost total de la millor solució trobada per la biblioteca.

Problema	Nodes	Propostes (1)	Propostes (2)	Diferència P.	Reducció C.
2	320	12	703	691	320,887
3	400	5	451	446	201,218
4	480	3	309	306	286,293
5	200	72	1824	1752	128,2466
6	280	19	913	894	128,957
7	360	7	558	551	248,258
8	440	4	370	366	195,244
9	255	28	1122	1094	23,553
10	323	11	692	681	15,9151
11	399	5	457	452	18,361
12	483	3	309	306	17,6792
13	252	29	1152	1123	11,4601
14	320	11	720	709	12,142
15	396	5	466	461	15,413
16	480	3	312	309	0
17	240	35	1290	1255	8,9
18	300	15	813	798	12,2825
19	360	7	564	557	5,513
20	420	4	410	406	6,3899

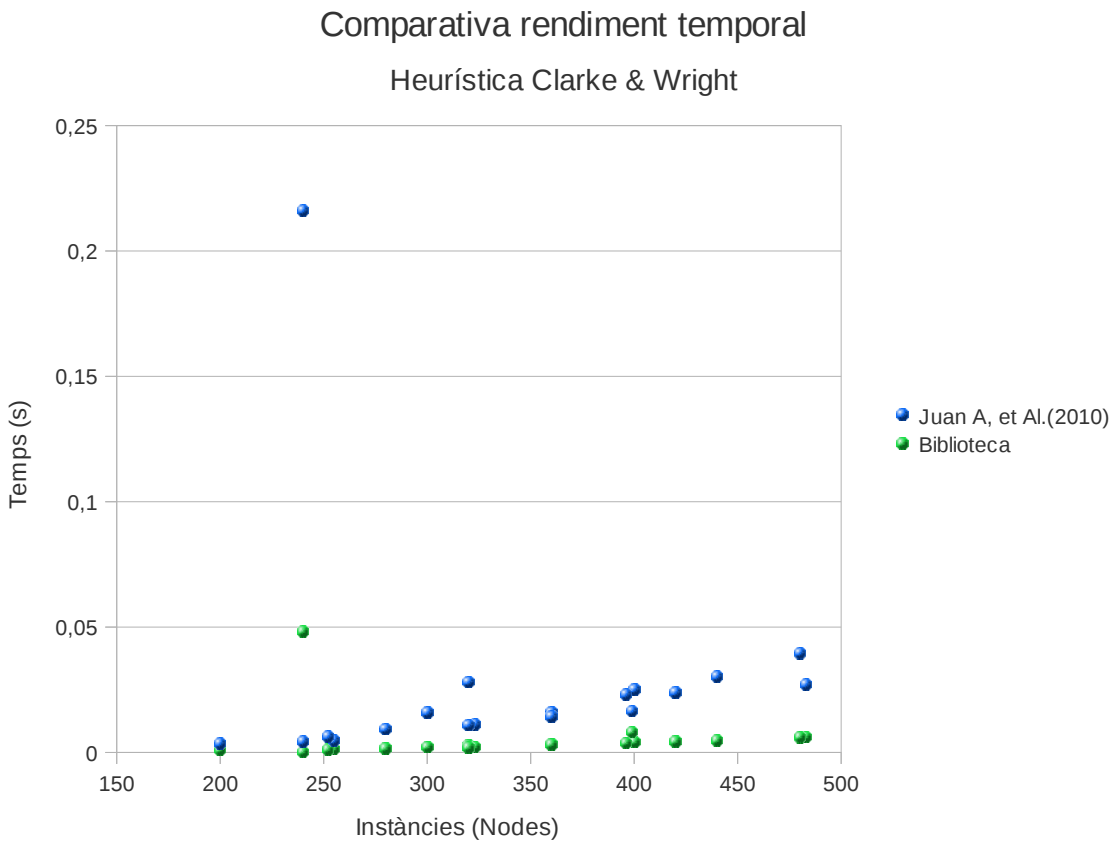
Taula 3. Solucions generades i reducció de cost total.

6. Interpretació dels resultats

Observant els valors de la Taula 1 s'arriba a la conclusió que la biblioteca es capaç de trobar noves combinacions de rutes amb un cost total inferior. Això succeïx en 19 de les 20 instàncies, en cap cas el rendiment de la biblioteca és pitjor que la proposta amb la qual s'ha comparat. Són especialment significatius els casos de les instàncies (1, 2, 3, 4, 5, 6, 7 i 8) on la reducció de cost és especialment significativa.

Per comprendre els motius pels quals la biblioteca ofereix un millor rendiment s'han d'analitzar les dades expressades a les taules 2 i 3 que mostren mesures qualitatives del codi desenvolupat.

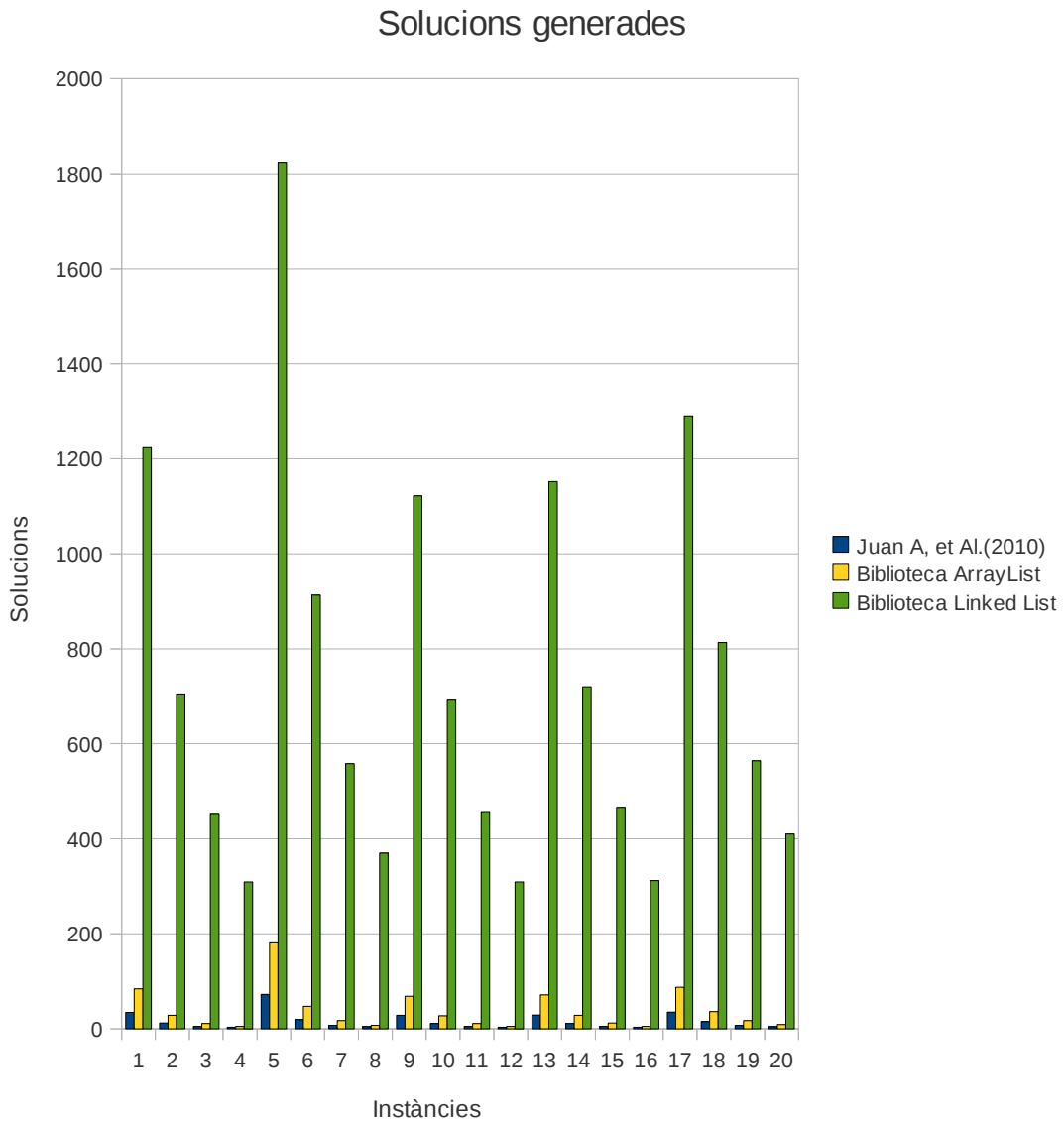
Les dades de la taula 2 demostren que la biblioteca es capaç de resoldre l'heurística constructiva d'en Clarke and Wrigth en un temps notablement inferior a la proposta (Juan, A, et al. 2010). A la gràfica 1 es pot comprovar, de manera visual, l'impacte en el rendiment temporal que ha tingut el conjunt de millores implementades. Per terme mitjà s'ha reduït el temps en un 80%.



Gràfica 1. Temps necessari per resoldre l'heurística CWS.

Cal notar que tot i que la gràfica estigui ordenada per la llargària de les instàncies (nombre de clients a visitar) això no implica un relació directa amb temps de resolució. La complexitat de cada instància també esta relacionada amb la capacitat dels vehicles que supliran els clients i amb el limit establert per la longitud total de les rutes. Així doncs, destaca força la instància 1 que tot i ser la segona amb menor nombre de clients (240) és la que té les limitacions de capacitat de càrrega i distància recorreguda més exigents (550 i 650) això es tradueix en un temps de resolució més elevat.

Per comprendre millor el comportament de la millora no és prou amb la mesura de rendiment de l'heurística constructiva. Cal comprovar si el procediment aleatori és més eficaç ja que això implicarà una major exploració de l'espai de solucions i en conseqüència més probabilitats de trobar propostes de major qualitat. Amb aquest objectiu s'ha construït la taula 3 on es mostren el nombre de solucions construïdes pels dos programaris provats. La gràfica 3 il·lustra el guany obtingut per la biblioteca.



Gràfica 2. Comparativa en solucions generades al llarg de 10 segons.

En primer terme, cal destacar la diferència en rendiment dels contenidors de la biblioteca estàndard de Java, *ArrayList* i *LinkedList*, tal i com s'ha descrit a la secció 4.2. Les barres en groc mostren el nombre de solucions generades per la biblioteca quan el mètode de reordenament fa servir com a mecanisme auxiliar una llista basada en accés aleatori. El comportament es sensiblement superior a la proposta de (Juan, A, et al. 2010) però extremadament inferior en comparació al reordenament recolzat en una llista doblement enllaçada (Barres verdes).

En segon terme, l'increment en el nombre de solucions és extraordinari i permet a la biblioteca més probabilitats de trobar una proposta de cost inferior. No obstant s'ha de tenir en compte que en ser una exploració de caràcter aleatori, tot i estar controlat per una distribució geomètrica, no hi ha una garantia de trobar una solució millor. De fet, es poden donar dos escenaris on tot hi generar una quantitat de solucions significativament superior no hi hagi una traducció directe en una millora de qualitat de la solució final. El primer cas és que les característiques de la instància facin simple la cerca d'una solució propera a l'òptim, això implica poc marge de millora i per tant es redueix l'impacte de l'increment de solucions. El segon es dóna quan, encarà que hi ha un marge de millora significatiu, per atzar no s'hagin generat solucions millors.

Es pot concloure, a partir de les dades de la taula 3, que l'augment de solucions generades s'ha traduït en millores notables per les instàncies (1, 2, 3, 4, 5, 6, 7 i 8). Els dos escenaris descrits al paràgraf anterior es donen per aquelles instàncies on no s'ha pogut trobar una solució millor o on les propostes trobades impliquen una reducció de cost més continguda.

A (Juan, A, et al. 2011) es descriuen dos metodologies afegides a l'algorisme d'en Clarke and Wright (CWS) i simulació Monte Carlo (MCS) que permeten dirigir la cerca en l'espai possible de solucions. La primera és mitjançant l'ús d'una memòria cau, a mesura que va iterant l'algorisme, es va emmagatzemant el millor ordre en el qual una sèrie de clients són servits. La segona és a partir de la partició de l'espai geogràfic de la localització dels clients, dividint la instància en petits problemes a on aplicar l'algorisme híbrid CWS-MCS. Aplicant aquests conceptes sobre la biblioteca desenvolupada es pot reduir les limitacions dels escenaris negatius descrits gràcies a un cerca més acurada en l'espai de solucions. Això pot construir combinacions de millor qualitat.

7. Conclusions

En aquest article s'ha presentat una biblioteca de programari obert desenvolupada en Java per resoldre problemes d'adreçament de vehicles (VRP). La seva característica principal és la d'implementar un mecanisme senzill, no requereix configuracions complexes ni especialitzades, que combina l'heurística constructiva d'en Clarke and Wright amb simulació Monte Carlo. Els resultats mostren que la biblioteca pot generar solucions de qualitat en temps reduït i sobre plataformes de maquinari domèstic.

S'espera que la biblioteca sigui profitosa per investigadors o professionals i que pugui ser una eina útil i flexible per desenvolupar variants complexes del VRP. La biblioteca és accessible de manera lliure mitjançant Internet des d'un repositòri de codi obert [http://cwsmcslib.github.com/CwsMcsVrpLibrary] que també ofereix documentació descriptiva i una plataforma de comunicació entre usuaris i desenvolupadors.

El paquet de programari i la seva documentació es distribueix sota la llicència lliure GNU-GPL v2 [http://www.gnu.org/licenses/gpl-2.0.html]. Aquesta permet copiar, modificar i redistribuir el codi o treballs derivats respectant sempre els principis de la llicència. La versió 2 de la família de llicències GNU-GPL és la que té un major consens respecte als principis de programari lliure o obert.

8. Treball futur

Existeixen dues línies de cerca que poden millorar el funcionament de la biblioteca. La primera és la implementació interna dels mecanismes de llistes o vectors. L'ús de contenidors de la biblioteca estàndard de Java poden potenciar de manera significativa el rendiment de la biblioteca degut a que la seva implementació ha sigut abastament comprovada i proporcionen mètodes amb un comportament proper a l'òptim. Així doncs, la transformació de vectors d'instàncies, arcs, nodes o vehicles en contenidors com *LinkedList* o *ArrayList* podem millorar els resultats.

La segona línia és la implementació de la biblioteca en el llenguatge compilat C++ amb el suport de la biblioteca estàndard STL. Java és un llenguatge que necessita d'una màquina virtual per executar-se. Això té conseqüències positives com: Despreocupació per la gestió de memòria, major rapidesa en el desenvolupament de codi o independència de plataforma. Però té aspectes negatius relatius a la gestió de memòria, ja que no es tant eficient com els llenguatges compilats (C++ o C). El desenvolupament mitjançant C++ i sobre la biblioteca STL pot permetre generar un codi orientat a objectes fent servir on conjunt de contenidors eficaços amb un rendiment significativament superior.

Bibliografia.

Clarke G and Wright J (1964). Scheduling of vehicles from a central depot to a number of delivering points. Opns Res 12: 568-581.

Chris Groër, Bruce Golden, Edward Wasil (2010). A library of local search heuristics for the vehicle routing problem. Mathematical Programing Computation.vol. 2, no. 2, pp. 79-101, 2010.

Cordeau JF, Gendrau M, Hertz A, Laporte G and Sormany JS (2004). New Heuristics for the Vehicle Routing Problem. In: Langevin A and Riopel D (eds). Logistics Systems: Design and Optimization. Kluwer Academic Publishers: Dordrecht (Hingham, MA).

Gillet B, Miller L. In: A heuristic algorithm for the vehicle dispatch problem, Oper Res 22 (1974), 340-349.

Golden, B.L., Wasil, E.A., Kelly, J.P., and Chao, I-M. (1998). Metaheuristics in vehicle routing. In: Crainic, T.G., and Laporte, G. (eds), Fleet Management and Logistics, pages 33–56. Kluwer, Boston.

Guimarans D, Herrero R, Riera D, Juan A, Ramos JJ (2011)."Combining Constraint Programming, Lagrangean Relaxation and Probabilistic Algorithms to solve the Vehicle Routing Problem". Annals of Mathematics and Artificial Intelligence. 3, 299-315.

Juan, A.; Faulin, J.; Ruiz, R.; Barrios, B.; Caballe, S. “The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem”. Applied Soft Computing, Vol. 10, No. 1, (2010) 215-224

Juan, A.; Faulin, J.; Jorba, J.; Riera, D.; Masip; D.; Barrios, B. “On the Use of Monte Carlo Simulation, Cache and Splitting Techniques to Improve the Clarke and Wright Savings Heuristics”. Journal of the Operational Research Society, Vol. 62, No. 6, (2011) 1085-1097

Jussien, Narendra and Rochart, Guillaume and Lorca, Xavier (2008). Choco: an Open Source Java Constraint Programming Library. INRIA a CCSD electronic archive server based on P.A.O.L.

Laporte G, “What You Should Know about the Vehicle Routing Problem”, Naval Researchs Logistics 54 (2007) 811-819

Li, F., Golden, B., Wasil, E. In: “ Very large-scale vehicle routing: new test problems, algorithms, and results. Comput. Oper. Res. 32, 1165-1179 (2005).

Meffert, Klaus et al.(2010): JGAP - Java Genetic Algorithms and Genetic Programming Package. URL: http://jgap.sf.net

Mladenović and Hansen. In: “Variable neighborhood search”, Comput. Oper. Res. 24 (1997), 1097-1100

Taillard, E. In: “Parallel iterative serach methods for vehicle routing problem”. Networks 23, 661-676 (1993).