VNIVERSITAT DE VALÈNCIA

Facultat de Ciències Matemàtiques

Departament d'Estadística i Investigació Operativa

Programa de doctorat en Estadística i Optimització

Tesi Doctoral

Models and solution methods for some hub location problems

Juan José Peiró Ramada

Directors:
Dr. Ángel Corberán Salvador
Dr. Rafael Martí Cunquero

Juliol 2016

N'Ángel Corberán Salvador i En Rafael Martí Cunquero, Catedràtics d'Universitat del Departament d'Estadística i Investigació Operativa de la Universitat de València,

CERTIFIQUEN que la present memòria d'investigació, titulada:

"**Models and solution methods for some hub location problems**"

ha estat realitzada sota la seua direcció per Juan José Peiró Ramada i constitueix la seua tesi per optar al grau de Doctor per la Universitat de València Estudi General.

I perquè així conste, en compliment amb la normativa vigent, n'autoritzen la presentació davant la Facultat de Ciències Matemàtiques de la Universitat de València perquè en puga ser tramitada la lectura i defensa pública.

Burjassot, 26 de juliol de 2016.

Ángel Corberán Rafael Martí

Avaluació

Seguint el *Reglament sobre depòsit, avaluació i defensa de la tesi doctoral de la Universitat de València (CG 29-XI-2011. Modificat CG 28-II-2012. Modificat en CG 29-X-2013. Modificat CG 28-VI-2016)*, la Comissió de Coordinació Acadèmica del Programa de Doctorat en Estadística i Optimització ha nomenat als següents sis experts per jutjar aquesta tesi:

- Professor Dr. José Manuel Belenguer Ribera (Universitat de València)
- Professor Dr. Rafael Caballero Fernández (Universidad de Málaga)
- Professora Dra. Paola Festa (Università degli Studi di Napoli FEDERICO II)
- Professora Dra. Mercedes Landete Ruiz (Universidad Miguel Hernández de Elche)
- Professor Dr. Jose Antonio Lozano Alonso (Euskal Herriko Unibertsitatea)
- Professora Dra. Hande Yaman (Bilkent University)

Ajudes institucionals rebudes

Ajudes:

- *Ministerio de Economía y Competitividad de España (MINECO/FEDER)*:
  - Ayuda predoctoral para la formación de doctores BES-2013-064245.
  - Ayuda a la movilidad predoctoral para la realización de estancias breves en centros de I+D EEBB-I-15-09778.
  - Ayuda a la movilidad predoctoral para la realización de estancias breves en centros de I+D EEBB-I-16-11254.

Projectes d'investigació públics que han ajudat a obtindre resultats d'aquesta tesi:

- *Ministerio de Economía y Competitividad de España (MINECO/FEDER)*:
  - Projectes TIN2009-07516, TIN2012-35632-C02, TIN2015-65460-C02-01.
  - Projectes MTM2009-14039-C06-02, MTM2012-36163-C06-02, MTM2015-68097.
- *Generalitat Valenciana*:
  - Projecte Prometeo 2013/049.

Agraïments

Als professors Àngel Corberán i Rafa Martí, per tota l'ajuda prestada aquests anys. Cada u, amb el seu estil personal, m'ha ensenyat coses que mai haguera somiat aprendre. Moltes gràcies per tot. També al professor Paco Montes, col·laborador necessari —que dirien els jutges— del delicte de fer anar les coses endavant, faça sol o ploga.

A més a més, a totes les persones que formen el *Departament d'Estadística i Investigació Operativa de la Universitat de València.* A les que avui estan i a les que ja no estan físicament. També a tots els membres de la Facultat de Ciències Matemàtiques.

A les persones que formen la *Universidade de Lisboa* (Portugal), la *Universidad Miguel Hernández de Elche*, la *University of Colorado Boulder* (Estats Units), la *Universidad Rey Juan Carlos* i la *Hogskolen i Molde* (Noruega), en especial a Francisco Saldanha da Gama, Mercedes Landete, Manuel Laguna, Abraham Duarte i Arild Hoff, per acollir-me com un membre més quan els he visitat.

A la societat que, a través del Ministeri d'Economia i Competitivitat d'Espanya, m'atorgà l'ajuda rebuda amb el contracte predoctoral, les ajudes rebudes per fer les estades fora del meu centre, i amb els projectes d'investigació que hem disfrutat en el passat i que disfrutem avui. També a la Generalitat Valenciana pel suport a l'activitat investigadora i docent que hem gaudit.

A tota la meua família, en especial als meus pares i a la meua germana, per l'ajuda rebuda al llarg de la vida.

Als meus amics i a les meues amigues pels ànims que sempre m'han donat.

al meu amor Manel,
a Conchita i a Juan

Resum

En aquesta tesi doctoral estudiem alguns dels problemes de localització de concentradors de tràfics (en anglès, *hub location problems*) en el context de les xarxes de transport. Aquests són problemes d'optimització combinatòria que apareixen en situacions en què existeix necessitat de transportar tràfics (també anomenats *fluxos*) com persones, articles i informació, des de molts orígens cap a moltes destinacions. En lloc d'enviar aquests fluxos utilitzant un enviament directe entre tots els parells de nodes de la xarxa, un subconjunt d'aquests nodes se selecciona per a ser utilitzat com a concentradors, amb l'objectiu de consolidar i distribuir els fluxos. D'aquesta manera, els concentradors indueixen una subxarxa que envia els tràfics d'una manera més eficient i amb menor cost, ja que permeten aconseguir economies d'escala quan es transporten grans quantitats de tràfics entre els nodes d'aquesta subxarxa.

Els problemes de localització de concentradors apareixen en un gran nombre d'aplicacions com, per exemple, les telecomunicacions, la logística i la distribució. Un exemple ilustratiu de l'ús de concentradors es troba en la indústria de transport aeri de passatgers. En l'actualitat no existeixen vols directes entre tots els parells d'aeroports del món. Tot el contrari, existeixen certs aeroports que són utilitzats com a punts de connexió. Si algú desitja viatjar, per exemple, des de València (Espanya) a San Francisco (Estats Units d'Amèrica), pot escollir entre diverses alternatives, com viatjar a Madrid i després a Chicago, per a finalment arribar a San Francisco, o viatjar a París i després a Atlanta abans d'arribar a San Francisco. Els aeroports de Madrid, París, Chicago i Atlanta s'estan utilitzant, en aquest cas, com a punts de connexió entre la ciutat origen i la ciutat de destinació. Als punts de connexió arriben vols des de molts orígens i també parteixen vols cap a moltes destinacions, i és en aquests punts de connexió on es concentren els passatgers en la xarxa de transport.

Bons resums dels treballs realitzats en l'àrea de localització de concentradors i de les seues aplicacions són el capítol de Contreras [22] que forma part del llibre Location Science [61], l'article de Alumur i Kara [6] i l'article de Campbell i O'Kelly [20]. Aquestes publicacions, a més de les referències que contenen, mostren que actualment es poden identificar diferents tipus de problemes dins d'aquesta família, cadascun d'ells amb les seues característiques diferenciadores, les seues aplicacions potencials i el seu nivell de complexitat.

En aquesta tesi estudiem diverses variants del problema de localització de concentradors. Les variants estudiades tracten de modelitzar diferents situacions i característiques

de la realitat. En totes elles desitgem minimitzar el cost d'enviar els tràfics per la xarxa de transport.

Aquesta tesi es divideix en sis capítols:

Capítol 1 Aquest capítol comença amb una breu introducció a l'optimització, on definim, de manera general, un problema d'optimització. Proposem una classificació dels problemes d'optimització, atenent a diferents criteris, com l'espai de solucions i la seua convexitat, la forma de la funció objectiu i de les restriccions del problema, el domini de les seues variables, etc. Després es contextualitzen els problemes d'optimització combinatòria, realitzant un breu repàs a les diferents classes d'aquests problemes, atenent a la seua complexitat computacional (problemes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complets i $\mathcal{NP}$-durs). Açò ens permet classificar els problemes abordats en aquesta tesi dins dels problemes $\mathcal{NP}$-durs i emmarcar el seu contingut: el disseny d'algorismes heurístics que proporcionen bones solucions en un temps raonable per a problemes d'aquest tipus.

Després ens endinsem en els problemes de localització d'instal·lacions (en anglès, *facility location problems*), que són una família de problemes d'optimització combinatòria amb multitud d'aplicacions. Aquests problemes tracten de cercar la millor localització per a instal·lar els serveis que es desitgen proveir a uns clients. Fem un resum dels problemes de localització d'instal·lacions més notables: el problema de la p-mitjana, el p-centre, la localització d'instal·lacions nocives, els problemes de localització amb objectius d'equitat, els problemes d'instal·lació de concentradors i els problemes de màxima cobertura de serveis. Després d'açò, ens detenim per explicar amb més profunditat els problemes de localització de concentradors. En justifiquem la importància, repassem la Literatura existent i proposem una classificació d'aquests problemes, basada en la que proposa Farahani et al. [35].

Molts dels problemes de localització de concentradors clàssics són $\mathcal{NP}$-durs, per la qual cosa resulta natural trobar en la Literatura molts treballs que presenten algorismes heurístics per a la seua resolució. Per açò, repassem també la bibliografia sobre aquest tema, atenent a les diferents variants del problema i de les metodologies emprades: problemes amb i sense restriccions de capacitat, problemes de tipus p-hub amb diferents patrons d'assignació de terminals a concentradors, les versions estocàstiques propostes, etc. Al llarg d'aquesta tasca, identifiquem les metodologies metaheurístiques proposades per resoldre les diferents variants.

Després, comentem molt breument les metodologies metaheurístiques que utilitzem en la tesi, classificades segons el paradigma:

- Algorismes evolutius, basats en mecanismes inspirats en l'evolució biològica. Aquests algorismes construeixen poblacions de solucions i les transformen seguint un procés com la selecció natural de les espècies. Dins d'aquest paradigma són molt coneguts els *algorismes genètics* [49, 70, 99] i la *cerca dispersa* [42, 45, 59].

- Algorismes basats en el mostreig de l'espai de solucions, com per exemple GRASP (*greedy randomized adaptive search procedures*) [36, 37, 40], que és un mètode de multiarrencada que, bàsicament, construeix una solució factible per a un problema d'optimització i després aplica un procés de millora fins a aconseguir una solució òptima local. En cada iteració de l'algorisme, els elements de la solució són seleccionats d'una manera aleatòria d'entre un conjunt d'elements candidats escollits voraçment.
- Algorismes basats en trajectòries, com per exemple la cerca tabú (en anglès, *tabu search*) [42, 44], que és una metodologia que cerca solucions sense detenir-se quan troba una solució òptima local. Açò s'aconsegueix gràcies a inhibir la possibilitat de tornar a una solució visitada anteriorment, després d'haver guardat en memòria els atributs de les solucions visitades prèviament. La programació de memòria adaptativa i l'oscil·lació estratègica també estan basades en aquest paradigma.

Aquesta breu introducció es completa, dins de cada capítol, amb una explicació més detallada de la metodologia utilitzada.

Finalment, presentem les famílies d'instàncies que resolem utilitzant els mètodes proposats en la tesi. Aquestes són tres:

- Civil Aviation Board. La instància original conté dades de passatgers de vols entre 25 ciutats importants dels Estats Units, així com la distància entre aquestes. Aquesta família d'instàncies va ser presentada per O'Kelly [76].
- Australian Post. La instància original arreplega dades sobre el servei de correus prestat en 200 localitats d'Austràlia. Va ser presentada per Ernst i Krishnamoorthy [31].
- USA423, la grandària de la qual és de 423 nodes. La instància original conté dades sobre passatgers entre 423 ciutats d'Estats Units. Va ser presentada per Peiró et al. [79].

Capítol 2 En aquest capítol estudiem la variant del problema de localització de concentradors en la qual els fluxos de cada client poden enviar-se a través d'un únic concentrador i on les connexions entre concentradors tenen capacitats modulars. El problema és conegut en anglès com el *capacitated hub location problem with modular link capacities* i va ser proposat per Yaman i Carello [104]. Aquest problema és una variant del problema clàssic de localització de concentradors en el qual el cost d'utilització d'arestes entre els concentradors de la xarxa no és lineal, sinó que és escalonat. En la literatura trobem que el model de programació matemàtica proposat per Yaman i Carello [104] per a aquest problema és no lineal i amb variables senceres, la qual cosa el converteix en un problema molt difícil de resoldre.

Nosaltres ens enfrontem al problema amb dos algorismes heurístics diferents:

- El primer algorisme que proposem està basat en la metodologia de l'oscil·lació estratègica (en anglès, *strategic oscillation*), una metodologia original-

ment presentada en el context de la cerca tabú. El mètode que proposem incorpora diversos dissenys per a la construcció i destrucció de solucions, així com diverses cerques locals per a intentar millorar les solucions factibles oposades. L'esquema de l'algorisme permet balancejar els processos de diversificació i d'intensificació d'aquesta cerca, amb la finalitat de fer-la més efectiva.

- El segon algorisme està basat en el paradigma de la programació de memòria adaptativa (en anglès, *adaptive memory programming*), una metodologia que implementa estructures de memòria per a crear mètodes sofisticats per a trobar bones solucions. Històricament, les estructures de memòria han sigut àmpliament implementades en el context de la cerca tabú, usualment embegudes en els algorismes de cerca local. En aquest algorisme explorem un disseny alternatiu en el qual les estructures de memòria constitueixen el cor dels mètodes de construcció i del procés de postoptimització, aquest últim dut a terme seguint un procés de reencadenament de trajectòries (en anglès, *path relinking*).

Els resultats obtinguts amb els algorismes proposats mostren que aquests són capaços d'aconseguir solucions de millor qualitat i de manera més eficient que els algorismes de resolució heurístics proposats anteriorment en la Literatura per a aquest problema.

Capítol 3 En aquest capítol ens centrem en la variant del problema de localització de concentradors en la qual cal trobar la ubicació de p concentradors i es permet connectar cada client amb r d'aqueixos p concentradors (tant p com a r són valors exògens). Aquest problema és conegut en anglès com el *uncapacitated r-allocation p-hub median problem* i va ser proposat per Yaman [103], que va proposar la possibilitat de permetre assignar cada terminal fins a un màxim de r dels p concentradors de la xarxa. Veiem que aquest problema generalitza dos problemes molt estudiats en la Literatura: el problema de localització de hubs amb assignació *única* (en anglès, *single assignment* p-hub location problem) quan $r = 1$, i el problema de localització de hubs amb assignació *múltiple* (en anglès, *multiple assignment* p-hub location problem) quan $r = p$.

Una diferència substancial amb el problema estudiat en el Capítol 2 és que aquest problema no contempla l'existència de capacitats dels concentradors ni de les connexions utilitzades.

Per a resoldre el problema hem proposat dos algorismes diferents:

- Un algorisme basat en la metogología GRASP, que realitza el procés de construcció de solucions en tres passos: localitza els p concentradors, assigna a cada terminal r dels p concentradors, i envia els tràfics per la xarxa dissenyada en els passos previs. Considerem tres tipus de cerques locals: una basada a canviar la localització dels concentradors i altres dos a canviar les assignacions dels terminals.

La naturalesa combinatòria d'aquest problema pot portar-nos a entorns d'exploració molt grans i, amb açò, a temps de computació elevats, per la qual cosa proposem també regles heurístiques de filtrat de solucions que descarten aquelles que no semblen ser candidates a millorar després d'aplicar-los els mètodes de cerca local proposats. Aquestes regles fan que els temps de computació milloren substancialment.

- El segon algorisme que proposem està basat en la metodologia de la cerca dispersa (en anglès, *scatter search*). La cerca dispersa és una metodologia poblacional que ha resultat ser eficient en molts problemes d'optimització combinatòria. Aquesta metodologia utilitza diverses estratègies per a generar solucions, combinar-les i millorar-les segons un esquema algorítmic concret. En particular proposa la utilització de mètodes de millora en les soluciones candidates a ser introduïdes en el conjunt conegut com *RefSet*, així com diversos mètodes generals de combinació de solucions. Nosaltres proposem algunes modificacions a aquest esquema algorítmic per accelerar el procés de cerca sense perdre qualitat en les solucions. Específicament, proposem utilitzar els mètodes de millora únicament al final de l'esquema original, que en uns casos són aplicats a totes les solucions en el *RefSet* i en altres casos només a la millor solució. A més, en lloc d'utilitzar els mètodes generals, utilitzem la metodologia del reencadenament de trajectòries com a mètode de combinació de solucions.

Els resultats que obtenim amb els algorismes proposats en 465 instàncies mostren que aquests són capaces d'aconseguir solucions de qualitat en un temps raonable. Els resultats són comparats amb aquells obtinguts amb algorismes exactes, que es mostren ineficients a l'hora de trobar solucions de qualitat en temps raonables en instàncies de grandària mitjana/gran. A més, quan intentem resoldre les versions clàssiques del problema (assignació única i múltiple), els nostres algorismes heurístics són competitius amb els millors algorismes proposats prèviament per a aquestes versions.

Capítol 4 En aquest capítol estudiem alguns models que tracten d'optimitzar la qualitat del servei que s'ofereix als clients d'una xarxa quan el transport es realitza a través de concentradors. En particular estem interessats a estudiar els problemes de localització que s'apliquen a situacions per les quals són importants alguns elements d'equitat i igualtat. Donada una xarxa, uns costos de transport i un conjunt de nodes amb tràfics per transportar, pensem que és possible calcular (o conèixer com un input del problema) un cost mínim *ideal* per a cada parella de nodes amb tràfics. Assumim que desitgem trobar solucions que estiguen orientades a satisfer als clients, les quals tenen com a cost real un valor proper al seu ideal. Les funcions objectiu de la Literatura no produeixen aquest tipus de solucions, ja que algunes estan enfocades a minimitzar el cost total de la xarxa i unes altres poden produir algunes solucions que estiguin orientades a millorar el cost d'alguns tràfics, però que no mesuren la desviació pel que fa al cost ideal de cadascuna de les possibles

parelles de nodes amb tràfic per transportar entre ells. Per això, nosaltres presentem un model que busca minimitzar les grans desviacions entre els costos reals i els costos ideals per a totes les parelles de nodes de la xarxa que tinguin tràfics per transportar. Ja que els costos poden variar significativament entre parelles de nodes, el model que proposem està basat a minimitzar les desviacions relatives.

Pensem que les persones que s'enfronten a tasques de decisió sobre la localització de concentradors en situacions reals també estaran interessades a comparar solucions que estiguen dissenyades per minimitzar el cost de la xarxa i per produir solucions del grat dels clients. En aquest context creiem que una proposta raonable pot ser un model biobjectiu que considere simultàniament solucions que milloren el cost de la xarxa i que milloren la qualitat del servei que ofereixen als seus clients. Per a això, presentem una variant del problema estudiat en el Capítol 3, que denominem en anglès *the uncapacitated r-allocation p-hub median and equitable center problem*. Per a aquest problema proposem una formulació bi-objectiu per construir un conjunt de solucions eficients.

Per resoldre el problema de millorar la qualitat del servei ofert als clients, proposem un algorisme basat en la metodologia GRASP. A més, per al problema biobjectiu, adaptem aquest algorisme per obtenir solucions aproximades de la frontera d'eficiència. Els mètodes que proposem són capaços d'aconseguir solucions d'alta qualitat, especialment si les comparem amb aquelles obtingudes amb diferents paquets comercials de propòsit general (com CPLEX, LocalSolver i OptQuest).

Capítol 5 En aquest capítol estenem el uncapacitated r-allocation p-hub median problem (Capítol 3) en dues direccions:

- considerant incertesa en la quantitat de tràfic a transportar i en els costos de transport, i
- considerant la possibilitat de transportar tràfic directament entre dues terminals (sense passar per concentradors), en cas que aquest tipus d'enviaments resulte en un cost de transport menor.

En particular, la nostra primera proposta és un model determinista que inclou costos fixos d'assignació i la possibilitat de transportar tràfics directes entre terminals (en anglès, *non-stop services*). Després, enriquim el model amb la incorporació d'incertesa en els costos de transport i en les quantitats de tràfic per transportar, construint un model estocàstic bietàpic, el qual denominem *stochastic uncapacitated r-allocation p-hub median problem with non-stop services*.

Si assumim que la incertesa presentada pot ser capturada a través d'un conjunt finit d'escenaris, cadascun amb una probabilitat d'ocurrència, és possible desenvolupar una formulació compacta que ajude a resoldre-ho. No obstant això, fins i tot per a instàncies xicotetes d'aquest problema, aquest model resulta ser massa gran per a ser fàcilment tractable amb solvers de propòsit general. Aquest fet fa que siga interessant que proposem el desenvolupament d'un procediment de resolució aproximat. Aquest procediment utilitza com a punt de partida una solució factible

del problema que resulta d'aïllar cadascun dels escenaris. Aquestes solucions són embegudes en un procés, inspirat en la metodologia del reencadenament de trajectòries, que gradualment incorpora atributs de les solucions de cada escenari, per a construir una solució del problema estocàstic global.

L'estudi computacional mostra els resultats de l'algorisme proposat, els quals són comparats amb els obtinguts utilitzant solvers de propòsit general. També els comparem amb els obtinguts per als models que no tenen en compte aquesta incertesa. Els resultats computacionals mostren que el mètode proposat és capaç d'aconseguir solucions d'alta qualitat en poc temps de computació, comparat amb els solvers de propòsit general que necessiten de supercomputadores amb molts recursos per a poder resoldre'ls.

A més, l'esquema algorítmic que presentem per a resoldre aquest problema pot utilitzar-se com un mètode per a resoldre altres problemes estocàstics amb característiques similars.

Capítol 6 Finalment, en aquest capítol presentem les conclusions generals i les línies futures de recerca, on es resumeixen els problemes de localització de concentradors tractats i els resultats obtinguts.

Aquests problemes, com s'ha dit anteriorment, són problemes $\mathcal{NP}$-durs, els quals hem tractat de resoldre amb algorismes basats en metodologies metaheurístiques. Tots els algorismes proposats es basen en procediments de construcció de solucions i en procediments de cerca local que intenten millorar aquestes solucions.

Els problemes de localització de concentradors poden ser entesos com una combinació de tres problemes d'optimització interrelacionats: un problema de localització d'instal·lacions, un problema d'assignació de clients a instal·lacions i un problema de transport. Cada algorisme constructiu que hem proposat ha seguit sempre la lògica d'aquesta interrelació: seleccionar els nodes que s'utilitzaran com a concentradors, assignar clients a aquests nodes i transportar els tràfics utilitzant la xarxa resultant. Depenent de la variant del problema que estiguem resolent, cadascun d'aquests passos es realitza atenent unes regles heurístiques que exploten les característiques particulars de la variant.

També hem dissenyat diversos procediments de cerca local, cadascun associat a un entorn:

- Els mètodes de cerca local que tenen com a objectiu millorar la localització dels concentradors exploren l'espai de solucions factibles de cada problema d'una manera més extensa, però cada solució veïna sol ser prou diferent com perquè la seua avaluació siga computacionalment costosa. Açò fa que, amb l'objectiu que els algorismes puguen ser utilitzats en un temps raonable, calga proposar regles heurístiques que guien les cerques d'aquest tipus d'entorns. Referent a açò, el mètode LS_{cluster} que presentem en la Secció 2.5.2 mostra que és possible explorar aquest tipus d'entorns amb un cost computacional relativament baix.

- Els mètodes que exploren els veïnats resultants del problema d'assignació de terminals a concentradors solen ser efectius per a trobar bones solucions. Nosaltres recomanem que s'apliquen cada vegada que es realitze un canvi en el conjunt de nodes que utilitzem com a concentradors. En variants del problema on existisquen restriccions de capacitat en els concentradors, els entorns basats en "intercanvis" solen funcionar millor que els basats en "insercions". La raó és que aquests últims solen produir més solucions infactibles que els primers en assignar (*inserir*) terminals a concentradors que ja estan plens. També recomanem els entorns basats en intercanvis que impliquen diversos terminals, ja que permeten més opcions d'intercanvi en alliberar molt més espai.

Les metodologies que hem utilitzat per a resoldre aquests problemes de localització inclouen GRASP, la cerca tabu, l'oscil·lació estratègica, la cerca de memòria adaptativa, la cerca dispersa i el reencadenament de trajectòries. Aquestes metodologies han demostrat que poden ser implementades de manera senzilla per a aquest tipus de problemes combinatoris i que són capaços d'explorar de manera eficient l'espai de solucions factibles de cada variant, ja siga per si mateixes o combinades amb altres mètodes.

També veiem a través de diversos experiments al llarg la tesi que els solvers de propòsit general són bones eines per a resoldre els problemes de localització de concentradors de grandària xicoteta. Per a problemes de grandària mitjana i gran, que són els que solen trobar-se en situacions reals, aquests solvers no són eficients, principalment perquè necessiten d'una gran quantitat de recursos de computació. Per açò pensem que proposar mètodes heurístics especialment dissenyats per a cada variant és la forma adequada d'atacar el problema. Els algorismes que hem presentat poden ser executats en ordinadors de sobretaula d'ús comú, a diferència dels solvers comercials que generalment necessiten de servidors d'altes prestacions.

L'ús d'algorismes heurístics té dos inconvenients: no garanteixen ni l'optimalitat ni la qualitat de la solució obtinguda. No obstant això, els resultats obtinguts amb els algorismes proposats han demostrat ser de gran qualitat quan ha sigut possible comparar-los i, per açò, també poden ser utilitzats com a ajuda als mètodes exactes en la tasca de cercar la solució òptima de cada problema.

Existeixen diversos problemes de localització de concentradors que ens agradaria explorar en un futur pròxim com, per exemple, algunes variants on no s'assumeix que els concentradors formen una subxarxa completa i per açò les decisions sobre l'ús de connexions entre concentradors també són variables del problema. Aquests problemes són coneguts en la literatura com "incomplete hub location problems".

També creiem que els problemes d'optimització que combinen decisions de localització i de rutes de vehicles poden modelitzar moltes situacions reals que, avui dia, estan sent modelitzades i resoltes de manera separada, produint solucions subòptimes. Aquest tipus de problemes es coneixen en la Literatura com a problemes de "Location–Routing" i també ens agradaria explorar-los.

Finalment, ja que molts problemes de localització que incorporen incertesa no han sigut explorats, creiem que existeix la necessitat que siguen modelitzats, estudiats i resolts. Aquests problemes semblen ser bastant difícils. No obstant això, pensem que els models estocàstics reflecteixen molt millor els problemes de la vida real, on per exemple existeixen canvis en els costos i en les demandes, per als quals els models deterministes no descriuen igual de bé aquests canvis a futur.

Algunes de les principals contribucions d'aquesta tesi han sigut sotmeses o acceptades per a la seua publicació. Aquestes són:

- Corberán, Á., Peiró, J., Campos, V., Glover, F., and Martí, R.
  Strategic oscillation for the capacitated hub location problem with modular links.
  Journal of Heuristics, 22 (2): 221 – 244, 2016.
- Hoff, A., Peiró, J., Corberán, Á., and Martí, R.
  Adaptive memory programming for solving the capacitated hub location problem with modular link capacities.
  Universitat de València Technical report, June 2016. Submitted.
- Martí, R., Corberán, Á., and Peiró, J.
  Scatter search for an uncapacitated p-hub median problem.
  Computers & Operations Research, 58: 53 – 66, 2015.
- Martí, R., Corberán, Á., and Peiró, J.
  The scatter search methodology: An experimental evaluation on hub location problems.
  To appear in *Handbook of Heuristics*. Springer International Publishing.
- Peiró, J., Corberán, Á., Laguna, M., and Martí, R.
  Models and solution methods for the uncapacitated r-allocation p-hub equitable center problem.
  Universitat de València Technical report, April 2016. Submitted.
- Peiró, J., Corberán, Á., and Martí, R.
  GRASP for the uncapacitated r-allocation p-hub median problem.
  Computers & Operations Research, 43: 50 – 60, 2014.
- Peiró, J., Corberán, Á., Martí, R., and Saldanha-da-Gama, F.
  Heuristic solutions for the stochastic uncapacitated r-allocation p-hub median problem with non-stop services.
  Universitat de València Technical report, May 2016. Submitted.

Resumen

En esta tesis doctoral estudiamos algunos de los problemas de localización de concentradores de tráficos (en inglés, *hub location problems*) en el contexto de las redes de transporte. Éstos son problemas de optimización combinatoria que aparecen en situaciones donde existe necesidad de transportar tráficos (también llamados *flujos*) como personas, artículos e información, desde muchos orígenes hacia muchos destinos. En lugar de enviar estos flujos utilizando un envío directo entre todos los pares de nodos de la red, un subconjunto de estos nodos se selecciona para ser utilizado como concentradores, con el objetivo de consolidar y distribuir los flujos. De esta manera, los concentradores inducen una subred que envía los tráficos de una manera más eficiente y con menor coste, ya que permiten conseguir economías de escala cuando se transportan grandes cantidades de tráficos entre los nodos de esta subred.

Los problemas de localización de concentradores aparecen en un gran número de aplicaciones como, por ejemplo, las telecomunicaciones, la logística y la distribución. Un ejemplo ilustrativo del uso de concentradores se encuentra en la industria de transporte aéreo de pasajeros. En la actualidad no existen vuelos directos entre todos los pares de aeropuertos del mundo. Todo lo contrario. Existen ciertos aeropuertos que son utilizados como puntos de conexión. Si alguien desea viajar, por ejemplo, desde Valencia (España) a San Francisco (Estados Unidos de América), puede escoger entre varias alternativas, como viajar a Madrid y después a Chicago, para finalmente llegar a San Francisco, o viajar a París y luego a Atlanta antes de llegar a San Francisco. Los aeropuertos de Madrid, París, Chicago y Atlanta se están utilizando, en este caso, como puntos de conexión entre la ciudad origen y la ciudad destino. A los puntos de conexión llegan vuelos desde muchos orígenes y también parten vuelos hacia muchos destinos, y es en estos puntos de conexión donde se concentran los pasajeros en la red de transporte.

Buenos resúmenes de los trabajos realizados en el área de localización de concentradores y de sus aplicaciones son el capítulo de Contreras [22] que forma parte del libro Location Science [61], el artículo de Alumur y Kara [6] y el artículo de Campbell y O'Kelly [20]. Estas publicaciones, además de las referencias que contienen, muestran que actualmente se pueden identificar distintos tipos de problemas dentro de esta familia, cada uno de ellos con sus características diferenciadoras, sus aplicaciones potenciales y su nivel de complejidad.

En esta tesis estudiamos diversas variantes del problema de localización de concentradores. Las variantes estudiadas tratan de modelizar diferentes situaciones y característi-

cas de la realidad. En todas ellas deseamos minimizar el coste de enviar los tráficos por la red de transporte.

Esta tesis se divide en seis capítulos:

Capítulo 1 Este capítulo comienza con una breve introducción a la optimización, donde definimos, de manera general, un problema de optimización y proponemos una clasificación de los problemas de optimización, atendiendo a diferentes criterios, como el espacio de soluciones y su convexidad, la forma de la función objetivo y de las restricciones del problema, el dominio de sus variables, etc. Después se contextualizan los problemas de optimización combinatoria, realizando un breve repaso a las diferentes clases de estos problemas, atendiendo a su complejidad computacional (problemas $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-completos y $\mathcal{NP}$-duros). Ésto nos permite clasificar los problemas abordados en esta tesis dentro de los problemas $\mathcal{NP}$-duros y enmarcar su contenido: el diseño de algoritmos heurísticos que proporcionen buenas soluciones en un tiempo razonable para problemas de este tipo.

Después nos adentramos en los problemas de localización de instalaciones (en inglés, *facility location problems*), que son una familia de problemas de optimización combinatoria con multitud de aplicaciones. Estos problemas tratan de buscar la mejor localización para instalar los servicios que se desean proveer a unos clientes. Hacemos un resumen de los problemas de localización de instalaciones más notables: el problema de la p-mediana, el p-centro, la localización de instalaciones nocivas, los problemas de localización con objetivos de equidad, los problemas de instalación de concentradores y los problemas de máxima cobertura de servicios. Tras ello, nos detenemos para explicar con más profundidad los problemas de localización de concentradores. Justificamos su importancia, repasamos la Literatura existente y proponemos una clasificación de estos problemas, basada en la que propone Farahani et al. [35].

Muchos de los problemas de localización de concentradores clásicos son $\mathcal{NP}$-duros, por lo que resulta natural encontrar en la Literatura muchos trabajos que presentan algoritmos heurísticos para su resolución. Por ello, repasamos también la bibliografía al respecto, atendiendo a las diferentes variantes del problema y de las metodologías empleadas: problemas con y sin restricciones de capacidad, problemas de tipo p-hub con diferentes patrones de asignación de terminales a concentradores, las versiones estocásticas propuestas, etc. A lo largo de esta tarea, identificamos las metodologías metaheurísticas propuestas para resolver las diferentes variantes.

Después, comentamos muy brevemente las metodologías metaheurísticas que utilizamos en la tesis, clasificadas según su paradigma:

- Algoritmos evolutivos, basados en mecanismos inspirados en la evolución biológica. Estos algoritmos construyen poblaciones de soluciones y las transforman siguiendo un proceso como la selección natural de las especies. Dentro de este paradigma son muy conocidos los *algoritmos genéticos* [49, 70, 99] y la *búsqueda dispersa* [42, 45, 59]

- Algoritmos basados en el muestreo del espacio de soluciones, como por ejemplo GRASP (*greedy randomized adaptive search procedures*) [36, 37, 40], que es un método multiarranque que, básicamente, construye una solución factible para un problema de optimización y luego aplica un proceso de mejora hasta conseguir una solución óptima local. En cada iteración del algoritmo, los elementos de la solución son seleccionados de una manera aleatoria de entre un conjunto de elementos candidatos escogidos vorazmente.
- Algoritmos basados en trayectorias, como por ejemplo la búsqueda tabú (en inglés, *tabu search*) [42, 44], que es una metodología que busca soluciones sin detenerse cuando encuentra una solución óptima local. Esto se consigue gracias a inhibir la posibilidad de volver a una solución visitada anteriormente, tras haber guardado en memoria los atributos de las soluciones visitadas previamente. La programación de memoria adaptativa y la oscilación entratégica también están basadas en este paradigma.

Esta breve introducción se completa, dentro de cada capítulo, con una explicación más detallada de la metodología utilizada.

Finalmente, presentamos las familias de instancias que resolvemos utilizando los métodos propuestos en la tesis. Éstas son tres:

- Civil Aviation Board. La instancia original contiene datos de pasajeros de vuelos entre 25 ciudades importantes de los Estados Unidos, así como la distancia entre éstas. Esta familia de instancias fue presentada por O'Kelly [76].
- Australian Post. La instancia original recoge datos sobre el servicio de correos prestado en 200 localidades de Australia. Fue presentada por Ernst y Krishnamoorthy [31].
- USA423, cuyo tamaño es de 423 nodos. La instancia original contiene datos sobre pasajeros entre 423 ciudades de Estados Unidos. Fue presentada por Peiró et al. [79].

Capítulo 2 En este capítulo estudiamos la variante del problema de localización de concentradores en la que los flujos de cada cliente pueden enviarse a través de un único concentrador y donde las conexiones entre concentradores tienen capacidades modulares. El problema es conocido en inglés como el *capacitated hub location problem with modular link capacities* y fue propuesto por Yaman y Carello [104]. Este problema es una variante del problema clásico de localización de concentradores en el que el coste de utilización de aristas entre los concentradores de la red no es lineal, sino que es escalonado. En la literatura encontramos que el modelo de programación matemática propuesto por Yaman y Carello [104] para este problema es no lineal y con variables enteras, lo que lo convierte en uno muy difícil de resolver.

Nosotros nos enfrentamos al problema con dos algoritmos heurísticos diferentes:

- El primer algoritmo que proponemos está basado en la metodología de la oscilación estratégica (en inglés, *strategic oscillation*), una metodología originalmente presentada en el contexto de la búsqueda tabú. El método que proponemos incorpora varios diseños para la construcción y destrucción de soluciones para este problema, así como varias búsquedas locales para intentar mejorar las soluciones factibles encontradas. El esquema del algoritmo permite balancear los procesos de diversificación y de intensificación de esta búsqueda, con el fin de hacerla más efectiva.
- El segundo algoritmo está basado en el paradigma de la programación de memoria adaptativa (en inglés, *adaptive memory programming*), una metodología que implementa estructuras de memoria para crear métodos sofisticados para encontrar buenas soluciones. Históricamente, las estructuras de memoria han sido ampliamente implementadas en el contexto de la búsqueda tabú, usualmente embebidas en los algoritmos de búsqueda local. En este algoritmo exploramos un diseño alternativo en el cual las estructuras de memoria constituyen el corazón de los métodos de construcción y del proceso de postoptimización, este último llevado a cabo siguiendo un proceso de reencadenamiento de trayectorias (en inglés, *path relinking*).

Los resultados obtenidos con los algoritmos propuestos muestran que éstos son capaces de conseguir soluciones de mejor calidad y de manera más eficiente que los algoritmos de resolución heurísticos propuestos anteriormente en la Literatura para este problema.

Capítulo 3 En este capítulo nos centramos en la variante del problema de localización de concentradores en la que hay que encontrar la ubicación de p concentradores y se permite conectar cada cliente con r de esos p concentradores (tanto p como r son valores exógenos). Este problema es conocido en inglés como el *uncapacitated r-allocation p-hub median problem* y fue propuesto por Yaman [103], que propuso la posibilidad de permitir asignar cada terminal hasta un máximo de r de los p concentradores de la red. Vemos que este problema generaliza dos problemas muy estudiados en la Literatura: el problema de localización de hubs con asignación *única* (en inglés, *single assignment* p-hub location problem) cuando $r = 1$, y el problema de localización de hubs con asignación *múltiple* (en inglés, *multiple assignment* p-hub location problem) cuando $r = p$.

Una diferencia sustancial con el problema estudiado en el Capítulo 2 es que este problema no contempla la existencia de capacidades de los concentradores ni de las conexiones utilizadas.

Para resolver el problema hemos propuesto dos algoritmos diferentes:

- Un algoritmo basado en la metogología GRASP, que realiza el proceso de construcción de soluciones en tres pasos: localiza los p concentradores, asigna a cada terminal r de los p concentradores, y envia los tráficos por la red

diseñada en los pasos previos. Consideramos tres tipos de búsquedas locales: una basada en cambiar la localización de los concentradores y otras dos en cambiar las asignaciones de las terminales.

La naturaleza combinatoria de este problema puede llevarnos a entornos de exploración muy grandes y, con ello, a tiempos de computación elevados, por lo que proponemos también reglas heurísticas de filtrado de soluciones que descartan aquellas que no parecen ser candidatas a mejorar tras aplicarles los métodos de búsqueda local propuestos. Estas reglas hacen que los tiempos de computación mejoren sustancialmente.

- El segundo algoritmo que proponemos está basado en la metodología de la búsqueda dispersa (en inglés, *scatter search*). La búsqueda dispersa es una metodología poblacional que ha resultado ser eficiente en muchos problemas de optimización combinatoria. Esta metodología utiliza varias estrategias para generar soluciones, combinarlas y mejorarlas según un esquema algorítmico concreto. En particular propone la utilización de métodos de mejora en las soluciones candidatas a ser introducidas en el conjunto conocido como *RefSet*, así como varios métodos generales de combinación de soluciones. Nosotros proponemos algunas modificaciones a este esquema algorítmico para acelerar el proceso de búsqueda sin perder calidad en las soluciones. Específicamente, proponemos utilizar los métodos de mejora únicamente al final del esquema original, que en unos casos son aplicados a todas las soluciones en el *RefSet* y en otros casos sólo a la mejor solución. Además, en lugar de utilizar los métodos generales, utilizamos la metodología del reencadenamiento de trayectorias como método de combinación de soluciones.

Los resultados que obtenemos con los algoritmos propuestos en 465 instancias muestran que éstos son capaces de conseguir soluciones de calidad en un tiempo razonable. Los resultados son comparados con aquellos obtenidos con algoritmos exactos, que se muestran ineficientes a la hora de encontrar soluciones de calidad en tiempos razonables en instancias de tamaño medio/grande. Además, cuando intentamos resolver las versiones clásicas del problema (asignación única y múltiple), nuestros algoritmos heurísticos son competitivos con los mejores algoritmos propuestos previamente para estas versiones.

Capítulo 4 En este capítulo estudiamos algunos modelos que tratan de optimizar la calidad del servicio que se ofrece a los clientes de una red cuando el transporte se realiza a través de concentradores. En particular estamos interesados en estudiar los problemas de localización que se aplican a situaciones para las cuales son importantes algunos elementos de equidad e igualdad. Dada una red, unos costes de transporte y un conjunto de nodos con tráficos a transportar, pensamos que es posible calcular (o conocer como un input del problema) un coste mínimo *ideal* para cada pareja de nodos con tráficos. Asumimos que deseamos encontrar soluciones que estén orientadas a satisfacer a los clientes, las cuales tienen como coste real un valor cercano a su ideal. Las funciones objetivo de la Literatura no producen este

tipo de soluciones, ya que algunas están enfocadas a minimizar el coste total de la red y otras pueden producir algunas soluciones que estén orientadas a mejorar el coste de algunos tráficos, pero que no miden la desviación con respecto al coste ideal de cada una de las posibles parejas de nodos con tráfico a transportar entre ellos. Por ello, nosotros presentamos un modelo que busca minimizar las grandes desviaciones entre los costes reales y los costes ideales para todas las parejas de nodos de la red que tengan tráficos que transportar. Ya que los costes pueden variar significativamente entre parejas de nodos, el modelo que proponemos está basado en minimizar las desviaciones relativas.

Pensamos que las personas que se enfrenten a tareas de decisión sobre la localización de concentradores en situaciones reales también estarán interesadas en comparar soluciones que estén diseñadas para minimizar el coste de la red y para producir soluciones del agrado de los clientes. En este contexto creemos que una propuesta razonable puede ser un modelo bi-objetivo que considere simultáneamente soluciones que mejoran el coste de la red y que mejoren la calidad del servicio que ofrecen a sus clientes. Para ello, presentamos una variante del problema estudiado en el Capítulo 3, que denominamos en inglés *como uncapacitated r-allocation p-hub median and equitable center problem*. Para este problema proponemos una formulación bi-objetivo en aras de construir un conjunto de soluciones eficientes.

Para resolver el problema de mejorar la calidad del servicio ofrecido a los clientes proponemos un algoritmo basado en la metodología GRASP. Además, para el problema bi-objetivo, adaptamos este algoritmo para obtener soluciones aproximadas de la frontera de eficiencia. Los métodos que proponemos son capaces de conseguir soluciones de alta calidad, especialmente si las comparamos con aquellas obtenidas con diferentes paquetes comerciales de propósito general (como CPLEX, LocalSolver y OptQuest).

Capítulo 5 En este capítulo extendemos el uncapacitated r-allocation p-hub median problem (Capítulo 3) en dos direcciones:

- considerando incertidumbre en la cantidad de tráfico a transportar y en los costes de transporte, y
- considerando la posibilidad de transportar tráfico directamente entre dos terminales (sin pasar por concentradores), en caso de que este tipo de envíos resulte en un coste de transporte menor.

En particular, nuestra primera propuesta es un modelo determinista que incluye costes fijos de asignación y la posibilidad de transportar tráficos directos entre terminales (en inglés, *non-stop services*). Después, enriquecemos el modelo con la incorporación de incertidumbre en los costes de transporte y en las cantidades de tráfico a transportar, construyendo un modelo estocástico bi-etápico, el cual denominamos *stochastic uncapacitated r-allocation p-hub median problem with non-stop services*.

Si asumimos que la incertidumbre presentada puede ser capturada a través de un conjunto finito de escenarios, cada cual con una probabilidad de ocurrencia, es posible desarrollar una formulación compacta que ayude a resolverlo. Sin embargo, incluso para instancias pequeñas de este problema, este modelo resulta ser demasiado grande para ser fácilmente tratable con solvers de propósito general. Este hecho hace que sea interesante que propongamos el desarrollo de un procedimiento de resolución aproximado. Este procedimiento usa como punto de partida una solución factible del problema que resulta de aislar cada uno de los escenarios. Estas soluciones son embebidas en un proceso, inspirado en la metodología del reencadenamiento de trayectorias, que gradualmente incorpora atributos de las soluciones de cada escenario, para construir una solución del problema estocástico global.

El estudio computacional muestra los resultados del algoritmo propuesto, los cuales son comparados con los obtenidos utilizando solvers de propósito general. También los comparamos con los obtenidos para los modelos que no tienen en cuenta esta incertidumbre. Los resultados computacionales muestran que el método propuesto es capaz de conseguir soluciones de alta calidad en poco tiempo de computación, comparado con los solvers de propósito general que necesitan de supercomputadores con muchos recursos para poder resolverlos.

Además, el esquema algorítmico que presentamos para resolver este problema puede utilizarse como un método para resolver otros problemas estocásticos con características similares.

Capítulo 6 Finalmente, en este capítulo presentamos las conclusiones generales y las líneas futuras de investigación, donde se resumen los problemas de localización de concentradores tratados y los resultados obtenidos.

Estos problemas, como se ha dicho anteriormente, son problemas $\mathcal{NP}$-duros, los cuales hemos tratado de resolver con algoritmos basados en metodologías metaheurísticas. Todos los algoritmos propuestos se basan en procedimientos de construcción de soluciones y en procedimientos de búsqueda local que intentan mejorar estas soluciones.

Los problemas de localización de concentradores pueden ser entendidos como una combinación de tres problemas de optimización interrelacionados: un problema de localización de instalaciones, un problema de asignación de clientes a instalaciones y un problema de transporte. Cada algoritmo constructivo que hemos propuesto ha seguido siempre la lógica de esta interrelación: seleccionar los nodos que se utilizarán como concentradores, asignar clientes a dichos nodos y transportar los tráficos utilizando la red resultante. Dependiendo de la variante del problema que estemos resolviendo, cada uno de estos pasos se realiza atendiendo a unas reglas heurísticas que explotan las características particulares de la variante.

También hemos diseñado varios procedimientos de búsqueda local, cada uno asociado a un entorno:

- Los métodos de búsqueda local que tienen como objetivo mejorar la localización de los concentradores exploran el espacio de soluciones factibles de cada problema de una manera más extensa, pero cada solución vecina suele ser lo suficientemente diferente como para que su evaluación sea computacionalmente costosa. Esto hace que, con el objetivo de que los algoritmos puedan ser utilizados en un tiempo razonable, haya que proponer reglas heurísticas que guíen las búsquedas de este tipo de entornos. A este respecto, el método LS_{cluster} que presentamos en la Sección 2.5.2 muestra que es posible explorar este tipo de entornos con un coste computacional relativamente bajo.
- Los métodos que exploran los vecindarios resultantes del problema de asignación de terminales a concentradores suelen ser efectivos para encontrar buenas soluciones. Nosotros recomendamos que se apliquen cada vez que se realice un cambio en el conjunto de nodos que utilizamos como concentradores. En variantes del problema donde existan restricciones de capacidad en los concentradores, los entornos basados en "intercambios" suelen funcionar mejor que los basados en "inserciones". La razón es que estos últimos suelen producir más soluciones infactibles que los primeros al asignar (*insertar*) terminales a concentradores que ya están llenos. También recomendamos los entornos basados en intercambios que impliquen varios terminales, ya que permiten más opciones de intercambio al liberar mucho más espacio.

Las metodologias que hemos utilizado para resolver estos problemas de localización incluyen GRASP, la búsqueda tabu, la oscilación estratégica, la búsqueda de memoria adaptativa, la búsqueda dispersa y el reencadenamiento de trayectorias. Estas metodologías han demostrado que pueden ser implementadas de manera sencilla para este tipo de problemas combinatorios y que son capaces de explorar de manera eficiente el espacio de soluciones factibles de cada variante, ya sea por sí mismas o combinadas con otros métodos.

También vemos a través de varios experimentos a lo largo la tesis que los solvers de propósito general son buenas herramientas para resolver los problemas de localización de concentradores de tamaño pequeño. Para problemas de tamaño medio y grande, que son los que suelen encontrarse en situaciones reales, estos solvers no son eficientes, principalmente porque necesitan de una gran cantidad de recursos de computación. Por ello pensamos que proponer métodos heurísticos especialmente diseñados para cada variante es la forma adecuada de atacar el problema. Los algoritmos que hemos presentado pueden ser ejecutados en ordenadores de sobremesa de uso común, a diferencia de los solvers comerciales que generalmente necesitan de servidores de altas prestaciones.

El uso de algoritmos heurísticos tiene, sin embargo, dos inconvenientes: no garantizan ni la optimalidad ni la calidad de la solución obtenida. No obstante, los resultados obtenidos con los algoritmos propuestos han demostrado ser de gran calidad cuando ha sido posible compararlos y, por ello, también pueden ser utilizados como ayuda a los métodos exactos en la tarea de buscar la solución óptima

de cada problema.

Existen varios problemas de localización de concentradores que nos gustaría explorar en un futuro próximo como, por ejemplo, algunas variantes donde no se asume que los concentradores forman una subred completa y, por ello, las decisiones sobre el uso de conexiones entre concentradores también son variables del problema. Estos problemas son conocidos en la literatura como "incomplete hub location problems".

También creemos que los problemas de optimización que combinan decisiones de localización y de rutas de vehículos pueden modelizar muchas situaciones reales que, hoy en día, están siendo modelizadas y resueltas de manera separada, produciendo soluciones sub-óptimas. Este tipo de problemas se conocen en la Literatura como problemas de "Location–Routing" y también nos gustaría explorarlos.

Finalmente, ya que muchos problemas de localización que incorporan incertidumbre no han sido explorados, creemos que existe la necesidad de que sean modelizados, estudiados y resueltos. Estos problemas parecen ser bastante difíciles. Sin embargo, pensamos que los modelos estocásticos reflejan mucho mejor los problemas de la vida real, donde por ejemplo existen cambios en los costes y en las demandas, para los que los modelos deterministas no describen igual de bien estos cambios a futuro.

Algunas de las principales contribuciones de esta tesis han sido sometidas o aceptadas para su publicación. Éstas son:

- Corberán, Á., Peiró, J., Campos, V., Glover, F., and Martí, R.
  Strategic oscillation for the capacitated hub location problem with modular links.
  Journal of Heuristics, 22 (2): 221 – 244, 2016.
- Hoff, A., Peiró, J., Corberán, Á., and Martí, R.
  Adaptive memory programming for solving the capacitated hub location problem with modular link capacities.
  Universitat de València Technical report, June 2016. Submitted.
- Martí, R., Corberán, Á., and Peiró, J.
  Scatter search for an uncapacitated p-hub median problem.
  Computers & Operations Research, 58: 53 – 66, 2015.
- Martí, R., Corberán, Á., and Peiró, J.
  The scatter search methodology: An experimental evaluation on hub location problems.
  To appear in *Handbook of Heuristics*. Springer International Publishing.
- Peiró, J., Corberán, Á., Laguna, M., and Martí, R.
  Models and solution methods for the uncapacitated r-allocation p-hub equitable center problem.
  Universitat de València Technical report, April 2016. Submitted.

- Peiró, J., Corberán, Á., and Martí, R.
  GRASP for the uncapacitated r-allocation p-hub median problem.
  Computers & Operations Research, 43: 50 – 60, 2014.
- Peiró, J., Corberán, Á., Martí, R., and Saldanha-da-Gama, F.
  Heuristic solutions for the stochastic uncapacitated r-allocation p-hub median problem with non-stop services.
  Universitat de València Technical report, May 2016. Submitted.

Summary

In this thesis, we study some hub location problems in the context of transportation networks. These are combinatorial optimization problems appearing in situations where there is a need of transporting some traffic (also called *flows*), like items, people, and information, from many origins to many destinations. Instead of sending these flows using a direct shipment between all pairs of nodes in the network, a subset of these nodes is selected to use as hubs, with the aim of consolidating and distribute the flows. Thus, hubs induce a subnetwork that sends the traffic more efficiently and at a cheaper cost, allowing economies of scale when large amounts of traffic between nodes on this subnet are transported.

Hub location problems appear in a large number of applications, like telecommunications, logistics, and distribution. An illustrative example of the use of hubs can be found in the passenger air transportation industry. At present, there is not a direct flight between all pairs of airports worldwide. Air lines make use of certain airports as connecting points. If someone wishes to travel, for example, from Valencia (Spain) to San Francisco (USA), there exist several alternatives from which to choose, such as traveling to Madrid, and then to Chicago, to finally reach San Francisco, or traveling to Paris, and then to Atlanta, before arriving in San Francisco. Madrid, Paris, Chicago, and Atlanta airports are being used, in this example, as connection points between the origin city and the destination city. We will assume that a connecting point receives flights from many origins and also sends flights to many destinations, and it is there, in this connection points, where passengers are concentrated in the transportation network.

Good summaries of the work done in hub location and its applications are the chapter by Contreras ([22]) that is part of the book Location Science ([61]), the paper by Alumur and Kara ([6]) and the paper by Campbell and O'Kelly ([20]). These publications, along with references therein, show that nowadays we can identify different types of problems within this family, each of it with its distinctive characteristics, its potential applications, and its level of complexity.

In this thesis we study different variants of hub location problems. The studied variants try to model several real world situations and characteristics. In all of them, we aim to minimize the cost of sending traffic through the transportation network.

This thesis is divided into six chapters:

Chapter 1 is a brief introduction to hub location problems in transportation networks. The problems are contextualized within the area of Combinatorial Optimization and,

specifically, as part of the family of the facility location problems. We also summarize the methodologies we use to solve these problems.

In Chapter 2 we study the variant of the hub location problem in which flows of each client can be sent through a single hub and where connections between hubs have modular capabilities. The problem is known as the *capacitated hub location problem with modular link capacities*. We solve this problem with two different algorithms, the first is based on the strategic oscillation methodology and the second is based on the paradigm of adaptive memory programming. The results obtained with the proposed algorithms show that they are capable to get better quality solutions and more efficiently than the algorithms previously proposed in the literature for this problem.

In Chapter 3 we focus on the variant of the hub location problem where we need to find the location of p hubs and allow each node to be allocation to, at most, r of the p ubs (both p and r are exogenous values). A substantial difference with the previous variant is that this problem does not contemplate the existence of capacities of the hubs neither the capacity of the links used. This problem is known as the *uncapacitated r-allocation p-hub median problem*. To solve it we propose two different algorithms, a standard one based on the GRASP methodology and a more sophisticated one based on the scatter search methodology. The results we obtain with the proposed algorithms show that they are able to find high quality solutions within a reasonable computing time when compared with exact resolution techniques.

In Chapter 4 we study some models that try to optimize the quality of service offered to customers when transport them using hub networks. To this end, we propose a mathematical programming model that minimizes the relative cost of each route with respect to the costs that customers consider the most reasonable. We have called this problem the *uncapacitated r-allocation p-hub equitable center problem*. To solve it, we propose a algorithm based on the GRASP methodology. In addition to this, we study the problem of designing the transport network when you wish to optimize the total cost of transport and, at the same time, the quality of service offered, resulting in a bi-objective problem for which we also propose a solution method. our proposals are able to achieve efficiently high quality solutions when compared with the solutions obtained with different commercial solvers (CPLEX, LocalSolver, and OptQuest).

In Chapter 5 we present several mathematical models that introduce "uncertainty" in hub location problems. Uncertainty, in our case, is presented in transportation costs and in the amount of flows that travel through the network. We propose an algorithm to solve the resulting stochastic programming problem (which we call the *stochastic uncapacitated r-allocation p-hub median problem with non-stop services*). Furthermore, this algorithm can be used as a method for solving other stochastic problems with similar characteristics. The computational results show that the proposed method is capable of achieving high quality solutions in a short computation time compared to general purpose solvers, which need supercomputers with high amount of resources to solve it.

Finally, in Chapter 6, general conclusions and future research lines are presented.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Summary

In this chapter we briefly introduce optimization problems, particularly combinatorial optimization problems. Then, we present the family of facility location problems, where we focus on hub location problems. The chapter finishes by shortly summarizing some metaheuristic methodologies that are used in this thesis, as well as the instances that have been used in the computational experiments.

1.1 Optimization problems

Optimization is a fundamental discipline in Mathematics. An optimization problem essentially consists of finding the best solution (called *optimum*) of a problem from a set of feasible solutions by using an (objective) function whose maximum (or minimum) value has to be obtained.

From a formal point of view, an optimization problem P consists of finding the optimal value of a certain function f in a domain S. The problem P can be stated as:

$$P = \begin{cases} \text{Maximize or minimize} & f(x) \\ \text{Subject to} & x \in S \subseteq \mathbb{R}^n, \end{cases} \tag{1.1}$$

where,

- S corresponds to the set of feasible solutions of the problem, and
- $f(x) : S \longrightarrow \mathbb{R}$ is the objective function that assigns to any feasible solution $x \in S$ a value in $\mathbb{R}$.

If P is a maximization problem, solving P deals with finding a solution $x^* \in S : f(x^*) \geq f(x), \forall x \in S$. Similarly, if P is a minimization problem, solving P consists of finding a solution $x^* \in S : f(x^*) \leq f(x), \forall x \in S$. In both cases, x^* is called an *optimal solution* of P.

1.1.1 A classification of optimization problems

There are different types of optimization problems. Regarding different criteria, we provide a possible classification here:

- The domain of the solution space
  - The domain is not restricted: *classical* or *unconstrained* optimization problems.
  - The domain is restricted: optimization problems *with constraints*.
- The convexity of the solution space
  - The solution space is a convex set: *convex* optimization problems.
  - The solution space is not a convex set: *non-convex* optimization problems.
- The form of the functions defining the objective and constraints of the problem (if any)
  - All functions are linear: *linear* optimization problems.
  - Otherwise: *non-linear* optimization problems.
- The variables' domain
  - Variables can take any real value: *continuous* optimization problems.
  - Variables are restricted to take integer values: *discrete* or *integer* optimization problems.
  - Some variables can take any real value while other take only integer values: *mixed integer* optimization problems.
- The coefficients of the model
  - All coefficients are known in advance: *deterministic* optimization problems.
  - Some coefficients are not known in advance: *stochastic* optimization problems.

1.1.2 Combinatorial optimization problems

Combinatorial optimization is one of the youngest and most active areas of discrete mathematics [53]. It has its roots in Combinatorics, Operations Research, and Theoretical Computer Science. Thousands of real-life problems can be formulated as combinatorial optimization problems (COP). These problems are a class of problems in which an optimal solution is usually searched in a finite set of solutions. Generally, the number of solutions of a combinatorial optimization problem is a huge number. Therefore, enumerating all the solutions and selecting the best one is not an appropriate method for solving them, and more efficient methods are needed.

Most combinatorial optimization problems can be formulated in a natural way in terms of graphs and as (integer) linear programs. Let E be a finite set (called ground set) with an associated cost function, and a family $\mathcal{F}$ (finite or countably infinite) of subsets of E, called feasible solutions. The COP with linear objective function is defined as the problem of finding a subset $F^* \in \mathcal{F}$ such that $c(F^*) = \sum_{e \in F^*} c_e x_e$ is maximum (or minimum), where x_e denotes the number of times that $e \in E$ appears in F^*.

Since the early 1970s, the difficulty of combinatorial optimization problems has been studied by many authors, resulting in a new and exciting research area called Computational Complexity. It was Edmonds who in mid 1960s started distinguishing among "easy" and "difficult" problems and called an algorithm "efficient" if its running time is bounded by a polynomial in the size of the problem representation. The shortest path problem, the max flow problem, the assignment problem, and the linear programming problem are examples of COPs for which an efficient algorithm exists. On the other hand, no polynomial-time algorithm is known for the solution of the traveling salesman problem or the uncapacitated facility location problem, to name just a few.

In order to obtain a more rigorous classification of problems according to their computational complexity, we should consider first decision problems, i.e., those problems whose instances have only two possible answers: "yes" or "no". The class of decision problems with the property that for any instance for which the answer is "yes" is called $\mathcal{NP}$. For example, the problem P: "given a graph G, does it contain Hamiltonian cycle?" is a decision problem. Note that an instance I of P whose solution is "yes" contains a Hamiltonian cycle and, knowing it, it is possible to check the "correctness" of the solution in polynomial time (just checking that the answer is a Hamiltonian cycle).

Now we define the class of "easy problems". $\mathcal{P}$ is the class of decision problems in $\mathcal{NP}$ for which there exists a polynomial algorithm. Obviously, $\mathcal{P} \subset \mathcal{NP}$. The question is $\mathcal{P} = \mathcal{NP}$? Nowadays, the answer to it is still an open problem that is among the most important open problems in Mathematics and Computer Science, although it is widely assumed that $\mathcal{P} \subsetneq \mathcal{NP}$.

Consider now two $\mathcal{NP}$ problems, P and Q. We say that P is polynomially reducible to Q if any instance of P can be converted in polynomial time to an instance of Q. This definition implies that if we know an algorithm for problem Q, it can be used to solve problem P with an "overhead" that is polynomial in the size of the instance. The class of $\mathcal{NP}$-complete problems is the subset of problems $P \subset \mathcal{NP}$ such that for all $Q \in \mathcal{NP}$, Q is polynomially reducible to P. The above definition means that $\mathcal{NP}$-complete problems are the most difficult problems in $\mathcal{NP}$. Moreover, it also gives a way of proving that a problem is $\mathcal{NP}$-complete. It has to belong to $\mathcal{NP}$ and another $\mathcal{NP}$-complete problem is polynomially reducible to it.

The problems we deal with in this thesis are COPs, which are not decision problems (although they have decision versions that can be proved equivalent to the optimization versions from the complexity point of view), and therefore cannot be $\mathcal{NP}$-complete problems. For these problems, it is commonly used the term $\mathcal{NP}$-hard. Many combinatorial optimization problems are $\mathcal{NP}$-hard and, therefore, if we assume that $\mathcal{P} \neq \mathcal{NP}$, we will not be able to find efficient (polynominal-time) algorithms for its solution. For

this reason, the research on $\mathcal{NP}$-hard problems focuses on three branches:

- Study special versions of some $\mathcal{NP}$-hard problems that can be solved in polynomial time.
- The design of heuristic algorithms that, in short computing times, provide good feasible solutions for $\mathcal{NP}$-hard problems.
- The design of exact methods that are able to find the optimal solution on, usually, small-medium instances.

There are many techniques to implement efficient algorithms for an approximate solution of a problem P, being remarkable those known as *heuristic* and *metaheuristic* techniques. These techniques, in which the speed of the search process is as important as the quality of the obtained solution, provide a general framework to create new hybrid algorithms in order to find good solutions, usually combining concepts from mathematics, statistics and probability, artificial intelligence and biological evolution.

1.2 Facility location problems

Facility location problems are an important family of problems in combinatorial optimization [60]. They refer to modeling, formulating, and solving the problems that can be described, following ReVelle and Eiselt [85], as siting facilities in some given space. We can find four components that characterize facility location problems:

1. Customers. They are the users of a given service. We assume that they are already at points or in a route.
2. Facilities. They are the providers of a given service. We assume that they need to be located.
3. A space in which customers and facilities are located.
4. A metric that usually indicates a distance (or a *cost*) between customers and facilities.

A facility location problem consist of determining the "best" location for one or several facilities or equipments in order to serve a set of demand points.

As stated by Owen and Daskin [77], the development or acquisition of a new facility is tipically a costly project. Before a facility can be purchased or constructed, good locations have to be identified, appropriate facility capacity specifications must be determined, and large amount of capital must be allocated. The high costs associated with this process make almost any location project a long-term investment. Thus, facilities which are located today are expected to remain in operation for an extended time. Determining the best locations for new facilities is thus an important strategic challenge.

Usually, researchers distinguish between problems in a n-dimensional real space and in a network. Both cases can be also subdivided into continuous and discrete location problems. In continuous problems, the points to be sited can generally be placed anywhere in the plane or in the network. An example of this case is placing an ambulance in a street in order to give quick emergency medical service to a geographical region. However, in discrete problems, the facilities can be placed only at a limited number of points on the plane or network.

As it is very well explained in Laporte, Nickel, and Saldanha-da-Gama [60], the papers [46, 47] by Hakimi are considered some of the first papers paying attention to network-based problems, providing important research directions in facility location. Hakimi introduced the concepts of *absolute median* and *absolute center* of a graph, and presented the p-median problem. Hakimi also proved the existence of at least one optimal solution which has all p facilities located solely at the nodes of the network, thus reducing the set of optimal solutions from a potentially infinite set to a finite set. This means that many network location problems can be cast into a discrete setting and thus leading to the possibility of using integer programming and combinatorial optimization techniques to tackle this family of problems.

During the following decades, researchers have focused on the study of theoretical properties of the solutions of the facility location problems and on the developments of solution procedures for them, especially on continuous, discrete, and network location problems. Some of the most important problems are:

- p-median problems, which usually refer to problems that search for facilities that minimize the transportation cost.

- p-center problems, whose objective is to minimize the largest customer–facility distance.

- Location of noxious (or obnoxious) facilities problems, which objectives are to locate undesirable facilities as far from the customers as possible. An example is the problem of locating a nuclear plant in a territory, which is a facility most people consider undesirable.

- Equity problems, whose objective is to locate facilities in such a way the customer–to–facility distances are as similar to each other as possible.

- Hub location problems, which usually refer to problems of using facilities as connecting points (hubs). They gather and distribute traffics (clients and/or goods) through these facilities instead of sending them from their origin to their destinations using a direct link, with the objective of minimizing the cost of transportation.

- Maximal covering location problems, which seek to maximize the amount of demand covered within an acceptable service distance by allocating a fixed number of facilities.

In addition to the above problems, other research lines are being investigated by researchers worldwide as, for example, location-routing problems, multi-period problems, multi-criteria location problem, and location under uncertainty. The book [29] by Eiselt and Marianov summarizes some works that are considered the basis of Location Science. Other excellent review papers on facility location are [9, 16, 48, 85, 86, 87, 96]. Among all location problems, we will focus on hub location problems.

1.3 Hub location problems

Among all combinatorial optimization problems, discrete facility location problems related to the design of transportation networks are among the most extensively studied problems due to their variety and importance. In all of them, a network $G = (V, E)$ is given with a set of demand nodes V and a set of edges E. For each pair of nodes i and $j \in V$, there is a traffic t_{ij} (of goods, people, deliveries, etc.) to be transported using the network. Depending on each variant of the problem, additional specific characteristics may be specified, such as a fixed cost of opening a facility at a potential location, or a limitation on its capacity. The reader can find in [34, 57, 61, 68, 72, 85, 86] excellent descriptions of facility location problems.

Hub-and-spoke architectures are usually deploy in transportation, communications, and computer networks to efficiently route traffics (flows) between many origins and many destinations in a network. Instead of shipping the traffic directly between nodes, a subset of them is selected for becoming hubs, thus consolidating and distributing the flow. This induces a transportation network that helps making the shipment more efficient and cheaper. For instance, we can take advantage from economies of scale when transporting large amounts of traffic between hubs.

In order to satisfy the demand of traffics with a fully connected network with $|V| = n$ nodes and with no hub node, the network should have $n(n-1)$ links to connect all the traffics. However, as it is said in Farahani et al. [35], if a node is selected to be used as a transshipment point and to connect all other nodes with each node, the network may only need $2(n-1)$ links to serve the traffics. This reduction in the number of links, that can be understood as an efficiency, comes from the usage of these kind of transshipment or consolidation points, commonly known as *distribution centers*, *hub nodes*, or simply *hubs*. The rest of the nodes in the network are called *terminal nodes*, *spokes*, or simply *terminals*. Hub-and-spoke architectures bring all the traffics from an origin to several destinations to a consolidating point, and then these traffics are routed to the different destinations by gathering all other flows from different origins to the same destination.

Applications of hub-and-spoke architectures can be found in transportation and communication systems: passenger transportation, postal services, goods for delivery, air freight, data packages on telecommunication services, etc. Goods, people, and commodities are transported using physical networks like airlines, vessels, trucks, trains, optic fiber links, and co-axial cables. The nodes that act as hubs are facilities, like airports and port terminals, sorting facilities in the case of postal services, and routers in telecommunication. A practical example can be found in the airline industry, where

passengers from any city do not usually travel using a direct flight to any destination they desire to travel. Opposite to this, airlines transport passengers to a connecting point where they transit to other flights to their destinations.

In this work we are interested in studying some variants of hub location problems (HLPs), in which a set H of locations is selected from a given set of potential locations V in order to be used as hubs for the network. In them, location and network design decisions have to be taken, and interrelations between the two levels of decision are involved. One level scrutinizes the selection of the set of nodes to locate the hubs, and the other level studies the design of the hub-and-spoke network to decide the use of the links to connect origins, destinations, and hubs, as well as the routing of flows through the network. The goal on HLPs is to identify an optimal subset of hubs in order to minimize a transportation cost function while satisfying a set of constraints. Generally, it is assumed in HLPs that direct transportation between terminals is not possible and, therefore, the traffic $t_{i,j \in V}$ travels along a path $i \to k \to l \to j$, where i and j are assigned to hubs k and l, respectively.

The study of hub location problems began with the work of O'Kelly [75] for continuous models, and O'Kelly [74, 76] for discrete models. The extensive work that has been developed in this area, as well as the applications that have been studied, are very well summarized in Contreras [22], and in Campbell [18], Alumur and Kara [6], and Campbell and O'Kelly [20]. These works, and their references, show that nowadays, we can identify several problem classes in this field.

Farahani et al. [35] propose a classification of hub location problems that we use here to introduce their different characteristics. For instance, we find HLPs in which the number of hubs is exogenously defined while, in others, this is an outcome of the decision making process. Additionally, hubs may be capacitated (when there is some limit for the traffic that can go through them) or uncapacitated. In fact, within the context of hub location problems, many aspects can be isolated, each of which helping in the characterization of the problem at hand. We can observe different variations of the problem and how they are commonly known, regarding the following aspects:

- Possible domain of the candidate nodes to be hub
  - The whole set V: network HLP
  - Only a subset of nodes in V can be hubs: discrete HLP
  - Any point in the plane or sphere generated by G: continuous HLP
- Form of the objective function
  - It minimizes the total cost of installing the hubs, allocating terminals to hubs, and routing traffics: min-sum HLP
  - It minimizes the maximum cost of routing the traffics: min-max HLP
- Allocation pattern for the terminals
  - Single allocation: a terminal is allocated to exactly one hub

  - Multiple allocation: a terminal can be allocated to several hubs (without a limit)
  - r-allocation: a limit, r, is imposed on the maximum number of hubs to which a terminal can be allocated

- Way of determining the number of hubs to be used
  - The number of hubs (denoted by p) is given a priori or in a exogenous way: p-HLP
  - The number of hubs is not given a priori (it is determined as part of the problem resolution): HLP

- Capacity constraints of the hubs
  - If we do not consider any capacity constraint for the hubs: uncapacitated HLP
  - If hubs have a maximum capacity in terms of traffics: capacitated HLP

- Regarding the costs of instalation of hubs
  - No costs are considered
  - Fixed costs are considered
  - Variable costs are considered

- Regarding the cost of connecting terminals to hubs
  - No costs
  - Fixed costs
  - Variable costs

- The coefficients of the model are known or uncertain
  - The coefficients are determined a priori: deterministic HLP
  - The coefficients are uncertain: stochastic HLP

Most of the "classical" hub location problems define a challenging class of $\mathcal{NP}$-hard problems. Accordingly, the same holds for many of their extensions and, thus, it is not surprising to find many articles presenting heuristic procedures in this field. Along with the first mathematical formulation for the single allocation p-hub median problem, O'Kelly [76] presented two specially tailored heuristics for obtaining feasible solutions to the problem. Since then, many heuristics have been developed. The uncapacitated single allocation p-hub median problem was further studied by Klincewicz [52] (who proposed a tabu search and a GRASP), Campbell [19] (greedy procedure), Ernst and Krishnamoorthy [31] (simulated annealing), Kratica et al. [55] (genetic algorithms),

Smith et al. [97] (neural networks), and Ilic et al. [50] (variable neighborhood search algorithm).

The capacitated version of the problem was investigated by Stanimirovic [98], who proposed a genetic algorithm. The single allocation p-hub center problem was tackled by Pamuk and Sepil [78] using tabu search and by Meyer et al. [69], who developed a 2-phase method based upon ant colony optimization.

The use of metaheuristics for approximating the optimal solution to hub location problems goes much beyond the problems for which an exogenous number of hubs, p, is imposed. In fact, the uncapacitated single allocation hub location problem was studied by Abdinnour-Helm and Venkataramanan [2] whose genetic algorithms improved the results presented by Abdinnour-Helm [1], who proposed a hybridization between genetic algorithms and tabu search. Other heuristics for the problem include those developed by Pirkul and Schilling [80] (lagrangean heuristic), and Cunha and Silva [26] (genetic algorithms). The multiple allocation version of the problem was investigated by Kratica et al. [56], who presented a genetic algorithm.

The capacitated single allocation hub location problem was first tackled heuristically by Ernst and Krishnamoorthy [30] (using the simulated annealing methodology), and afterwards by Chen [21] (combining simulated annealing with tabu search), Randall [81] (ant colony optimization), Silva and Cunha [94] (tabu search method), and Contreras et al. [24, 25] (lagrangean heuristics). The multiple allocation version of the problem was considered by Kratika et al. [54] (genetic algorithms), and Rodríguez-Martín and Salazar-González [89] (iterative local search). We note that in these two works, unlike the other works already quoted, the hub level network can be incomplete, i.e., it does not need to be a complete graph. These are problems with (hub level) network design decisions. Also considering incomplete hub networks we find the work by Calik et al. [17] on a hub covering problem whose optimal solution is approximated using tabu search. Other works containing heuristics for hub location problems include those by Marianov et al. [66] on competitive hub location (tabu search was considered), Eiselt and Marianov [28] (using heuristic concentration—[90]), and Lüer et al. [64] (genetic algorithm).

The variety of heuristics for hub location problems covered by the literature includes other more specific hub location problems not quoted above. This is the case of the papers by Yaman [104] (local search algorithm), Marianov and Serra [65] (tabu search), Sasaki et al. [93] (a greedy approach that generalizes the procedures suggested by Campbell [19]), and Alumur and Serper [8] (variable neighborhood search method).

As far as stochastic hub location problems are concerned, to the best of our knowledge, the only contribution to the literature so far is the paper by Bollapragada et al. [15]. The authors study a fixed-wireless network-planning problem with a two phase planning horizon and a different budget for each phase. They consider different hub types (regarding costs and capacities) and assume stochastic demands. A greedy algorithm is proposed for maximizing the expected covered demand. The development of metaheuristics for stochastic combinatorial optimization problems is not a new topic as can be observed in the survey paper by Bianchi et al. [14]. Nevertheless, most of the work it has been done on the stochastic traveling salesman problem, on stochastic

vehicle routing problems, and on stochastic scheduling problems.

Still today, there are certain instances of hub location problems whose solution using exact methods is not possible. Therefore, we need to use approximate methods to solve these instances. We believe that we can provide new techniques and methodologies, in the framework of metaheuristic optimization, to solve some versions of HLPs efficiently. This efficiency is related to the solving methods designed to provide high quality solutions in runtimes that can be acceptable for the user.

We believe that algorithms that are based on metaheuristic frameworks may be good candidates as generic resolution techniques in the context of HLP. As a result, we have designed a set of heuristic methods to deal with some versions of this kind of problems. We do a progressive and incremental approach to address different versions of HLPs. We have started with simple greedy strategies and, at each step, we have provided a new level of heuristic abstraction, using the characteristics of previous levels to enrich them with different criteria to explore the solution space in a smarter way. Also, we have designed hybrid methods that are capable of dealing with complex situations and problems of considerable size.

1.4 Metaheuristic methodologies

In this section we briefly summarize the principles of the metaheuristic frameworks that we have used to solve the problems studied in this thesis.

When speaking about heuristic optimization, constructive methods are algorithms that build a solution for a problem P from scratch. The most basic constructive methods are the *greedy methods*, which are guided by mechanisms to incrementally select an element for the new solution s to solve P. Greedy methods always select the best option available (the one with least cost in our case) from a set of unselected feasible elements. Other constructive methods randomly select (according to a probability distribution) the elements to incorporate to s. Those methods are called *random constructive methods*.

Local search methods start from a given solution s and try to improve it by successively applying small modifications to s until the solution cannot be further improved. These methods often provide high-quality solutions and some of them might be optimal. Nevertheless, in difficult problems, they can become prematurely trapped in local optima.

Metaheuristic methodologies are high-level procedures that coordinate simple heuristics to find high-quality solutions to difficult problems by going beyond the solution obtained after applying a construction method and a local search method for P. These metaheuristic methodologies are categorized depending on some paradigms:

- **Genetic algorithms, evolutionary paradigm**, ([49, 70, 99]) which are based on the mechanisms inspired in biological evolution and natural selection. They evolve populations of solutions that are combined to generate offsprings by applying mutation processes that create solutions with new characteristics. Usually, the

search process finishes after reaching a number of generations without improvement of the best solution found so far for P.

- **GRASP, space sampling paradigm** ([36, 37, 40]), which stands for greedy randomized adaptive search procedures, are multistart procedures that consist, basically, in constructing and improving a solution at each global iteration. Each construction is guided by a greedy mechanism and the selection of the next element to incorporate to a solution is chosen randomly from a restricted candidate list of elements previously filtered by the greedy mechanism. The guiding function recognizes previous selections of elements already incorporated. We refer the reader to [38, 39] for excellent annotated bibliographies on GRASP.

- **Tabu search, trajectory paradigm** ([42, 44]), which is a search procedure that explores the solution space beyond local optimality. In basic terminology, it applies a local search method until it finds a local optimum. At such points, instead of stopping there, the algorithm moves to points of the solution space with worse solutions quality (trying to escape from the local optimum) expecting that, after some steps, a better solution will be found. To avoid cycling, the procedure makes use of memory that contains attributes of recently visited solutions to prevent visiting them and cycling. Usually, the search process finishes after reaching a number of iterations without improving the best solution found so far.

1.5 The instances we will use

It is typical, when proposing any method for solving a combinatorial $\mathcal{NP}$-hard problem, to test it with some sets of instances (also known as examples). These tests give an estimation of the behavior of the method proposed for solving any instance from the universe of problems of this class. The scientific community has historically found and, hence, proposed some instances that establish a challenging benchmark of instances to be solved. In what follows, we describe the main characteristics of the three sets of instances that we will use for testing our algorithms along this dissertation. The specific list of instances used will be explained in detail in the computational section of each chapter.

CAB (Civil Aviation Board) data set. It is based on airline passenger flows between some important cities in the United States. It consists of a data file, presented by O'Kelly [76] in 1987, with the distances and flows of a 25 nodes graph.

AP (Australian Post) data set. It is based on real data from the Australian postal service and was presented by Ernst and Krishnamoorthy [31] in 1996. The size of the original data file is 200 nodes. Smaller instances can be obtained using a code from ORLIB [13]. These instances do not have symmetric flows, i.e., for a given pair of nodes i and j, t_{ij} is not necessarily equal to t_{ji}. Moreover, flows from one node to itself can be positive.

USA423 data set. This family of instances was introduced in [79] and is based on real airline data provided by Data In, Information Out (DIIO), which is a world leader firm in aviation business intelligence tools. The instance consists of a data file concerning 423 cities in the United States, where real distances and passenger flows for an accumulated three months period are considered.

The entire set of instances is available at OPTSICOM Project.

Chapter 2

Solution methods for the capacitated single assignment hub location problem with modular link capacities

Summary

In this chapter we propose two heuristic algorithms to solve the capacitated single assignment hub location problem with modular link capacities (CSHLPMLC). This problem is a variant of the classical hub location problem in which the cost of using edges is not linear but stepwise, and the hubs are restricted in terms of transit capacity rather than in the incoming traffic.

We first propose a metaheuristic algorithm based on strategic oscillation, a methodology originally introduced in the context of tabu search. Our method incorporates several designs for constructive and destructive algorithms, together with associated local search procedures, to balance diversification and intensification for an efficient search.

Then, we propose another metaheuristic algorithm based on adaptive memory programming that implements memory structures to create advanced search methods. Memory structures have been widely implemented in the context of the tabu search methodology, usually embedded in local search algorithms. In this algorithm we explore an alternative design in which memory structures constitute the core of the constructive method and also of a path relinking post-processing.

Computational results on a large set of instances show that, in contrast to exact methods that can only solve small instances optimally, our metaheuristics are able to find high-quality solutions on larger instances in short computing times. In addition, the new methods outperform the previous tabu search implementation.

2.1 Introduction

Among the family of hub location problems, we focus here on a specific variant known as the capacitated single assignment hub location problem with modular link capacities (CSHLPMLC). This problem was formulated as a quadratic mixed integer programming problem by Yaman and Carello [104]. These authors also proposed a branch-and-cut algorithm to solve optimally the problem together with a metaheuristic to obtain good initial solutions. As proved in [102], the CSHLPMLC is $\mathcal{NP}$-hard. In what follows we summarize the characteristics of this problem:

- G is a connected network. V is the set of demand nodes. For each pair of nodes i and j, there is a traffic t_{ij} to be transported through the edges E.
- All the nodes in the network are demand points (i.e. terminal nodes), as well as potential hub locations (i.e. hub nodes).
- Each node i is assigned to only one hub h_i.
- Hubs can be located at any node i of the network, with an associated installation cost C_{ii}.
- The number of hubs used is not fixed a priori. A solution can have any number of hubs (from 1 to $|V|$).
- All hubs have the same capacity Q^h, which limits the total traffic transiting through them.
- Edges between hubs have capacity Q^b. If the traffic between two hubs exceeds this amount, additional edges with Q^b capacity are added.

The CSHLPMLC consists of selecting a subset of nodes to be hubs and assigning the rest of the nodes to them in such a way the transportation cost is minimized while satisfying the capacity constraints.

Many heuristics and metaheuristics have been proposed to solve different variants of hub location problems, including VNS [50], tabu search [104], and several complex hybrid techniques.

In this chapter we first present a simple, easily adaptable and powerful algorithm, based on the iterated greedy–strategic oscillation (SO) methodology [41, 44]. The purpose of this paper is to investigate the SO proposal, which alternates between constructive and destructive phases as a basis for creating a competitive method for this hub location problem. We begin by summarizing the previous work by Yaman and Carello [104], which as far as we know is the only published paper devoting attention to this specific problem. The problem definition, the notation used, as well as the formulation proposed in [104] are described in Section 2.2, while the previous heuristic method in [104] is described in Section 2.3. We then describe in Section 2.4 the elements of our SO method, including the memory structures employed in our implementation. Specifically, Section 2.4.6 presents several experiments to determine the values of the critical search parameters, a comparison between methods and optimal results. Computational outcomes on a large set of instances show that, while only small instances can be optimally

solved with exact methods, our metaheuristic is able to find high-quality solutions on larger instances in short computing times, and outperforms the previous tabu search implementation.

After, we propose a new heuristic based on Adaptive Memory Programming in Section 2.5. We basically introduce memory structures to enhance the performance of our methods. Memory-based strategies, which are the hallmark of the well-known tabu search methodology [44], and coined under the term adaptive memory programming, are founded on a quest for "integrating principles", by which alternative forms of memory are appropriately combined with effective strategies for exploiting them. Specifically, we propose different construction methods in Section 2.5.1, and local search methods in Section 2.5.2, and then we study the effectiveness of memory structures and compare them with memory-less variants in Section 2.5.4. The experiments in Section 2.5.4.2 show that the adaptive memory features are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, these methods contrast with memoryless designs that heavily rely on semi-random processes that implement a form of sampling. Statistical tests confirm the superiority of our proposal with respect to other developments. Finally, we will present some concluding remarks in Section 2.5.5.

2.2 A non-linear programming formulation

Let $G = (V, E)$ be a network, where $V = \{1, \ldots, n\}$ is the set of nodes and E is the set of edges. For any pair of nodes $i, j \in V$, t_{ij} denotes the traffic to be transported from i to j.

Each node i is either a terminal node or a hub node (*terminal* and *hub* for short). A terminal can only be assigned to a single hub. A hub is assigned to itself. The hubs and the edges among them define a complete subgraph. Opening a hub at node i has a fixed installation cost C_{ii}. Each hub i has a capacity Q^h limiting the total amount of traffic transiting through i.

There are two types of edges between nodes: edges of the first type are used to connect terminals with hubs, and we call them *access edges* in reference to the access to the network they provide. Let m_i be the number of access edges needed to route the incoming and outgoing traffic at node i, and let Q^a be the capacity an access edge allows to transfer through it in each direction. So,

$$m_i = \max \left\{ \left\lceil \frac{\sum_{j \in V} t_{ij}}{Q^a} \right\rceil, \left\lceil \frac{\sum_{j \in V} t_{ji}}{Q^a} \right\rceil \right\}.$$

The cost of installing m_i edges between terminal i and hub k is denoted by C_{ik}. Edges of the second type are used to transfer traffics between hubs, and we call them *backbone edges*. Each backbone edge has a maximum traffic capacity of Q^b (in each direction). We define A as the set of (directed) arcs associated with the (undirected) edges in E, $A = \{(k, l) : k, l \in V, k \neq l\}$.

If nodes k and l are hubs, the amount of traffic on arc (k,l), denoted as z_{kl}, is the traffic that has to be transported from nodes assigned to k to nodes assigned to l. The capacity Q^b of a given edge $\{k,l\}$ cannot be less than the maximum traffic on its corresponding arcs (k,l) and (l,k), and the cost of installing the edge is denoted by R_{kl}. This edge capacity Q^b can be understood, for example, as the capacity of an airplane. If $2Q^b \geq z_{kl} > Q^b$ holds, two copies of the edge (two airplanes) are needed, even if the second one transports less than Q^b passengers, and a fixed cost R_{kl} for each airplane has to be paid. This reflects the non-linear nature of the costs R_{kl}. This modular link characteristic makes this model much more realistic than the linear cost version.

Three different costs have to be considered in this problem: The opening costs of the hubs (C_{kk}), the assignment costs of terminals to hubs (C_{ik}), and the traffic costs between hubs. While cost C_{ik} corresponds to that of transporting all the traffic involving i through hub k, R_{kl} represents the cost of using only one backbone edge $\{k,l\}$. This last cost has to be multiplied by the number of copies needed of the edge $\{k,l\}$. So, we face to two types of decisions, the *binary decision* of assigning a terminal to a hub and the *integer decision* associated with how many copies of the edges among hubs will be used.

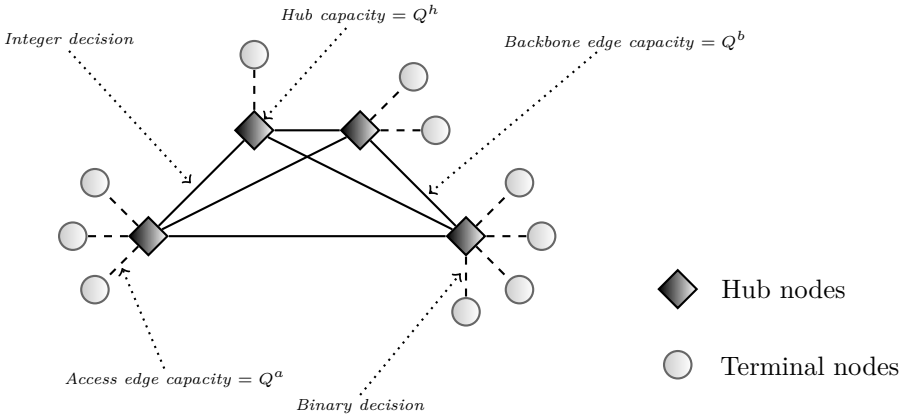

Figure 2.1. Different costs in the CSHLPMLC

Figure 2.1 shows a diagram which represents the hubs as shaded squares, the terminals as circles, the assignments of terminals to hubs in dashed lines, and the different capacities involved.

The following variables are defined in [104] in order to provide the mathematical programming model shown below:

- The assignment variable x_{ik} is equal to 1 if terminal i is assigned to hub k, and 0 otherwise. If node i receives a hub, then x_{ii} takes value 1.
- z_{kl} is the traffic on an arc $(k,l) \in A$ and w_{kl} is the number of copies of the edge $\{k,l\} \in E$.

Then, the capacitated single assignment hub location problem with modular link capacities can be formulated ([104]) as follows:

$$\min \quad \sum_{i\in V}\sum_{k\in V} C_{ik}x_{ik} + \sum_{\{k,l\}\in E} R_{kl}w_{kl} \tag{2.1}$$

subject to:

$$\sum_{k\in V} x_{ik} = 1 \quad \forall i \in V \tag{2.2}$$

$$x_{ik} \leq x_{kk} \quad \forall i \in V, \quad \forall k \in V \setminus \{i\} \tag{2.3}$$

$$\sum_{i\in V}\sum_{j\in V}(t_{ij}+t_{ji})x_{ik} - \sum_{i\in V}\sum_{j\in V} t_{ij}x_{ik}x_{jk} \leq Q^h x_{kk} \quad \forall k \in V \tag{2.4}$$

$$z_{kl} \geq \sum_{i\in V}\sum_{j\in V} t_{ij}x_{ik}x_{jl} \quad \forall (k,l) \in A \tag{2.5}$$

$$Q^b w_{kl} \geq z_{kl} \quad \forall \{k,l\} \in E \tag{2.6}$$

$$Q^b w_{kl} \geq z_{lk} \quad \forall \{k,l\} \in E \tag{2.7}$$

$$x_{ik} \in \{0,1\} \quad \forall i,k \in V \tag{2.8}$$

$$w_{kl} \in \mathbb{Z}_+ \quad \forall \{k,l\} \in E \tag{2.9}$$

$$z_{kl} \geq 0 \quad \forall (k,l) \in A. \tag{2.10}$$

Constraints (2.2) imply that each node has to be assigned to only one hub. Constraints (2.3) force node k to be a hub if a node i is assigned to it. Constraints (2.4) specify that the capacity of a given hub k cannot be less than the amount of traffic that transits through it, thus prohibiting allocations to k beyond its maximum capacity Q^h. Note that the flow between two terminals assigned to the same hub is counted twice in the first term of the inequalities, and hence it has to be subtracted once in the second term. Constraints (2.5) add up the traffics through a given arc (k,l). Finally, constraints (2.6) and (2.7) fix the number of copies needed of each edge.

We have tested this formulation on a small set of instances to check if our metaheuristic would be able to find the optimal solution obtained using exact methods on this formulation. Results of this comparison can be found in Section 2.4.6.4.

2.3 Previous methods

A metaheuristic and a branch-and-cut algorithm for the CSHLPMLC were proposed in [104]. The metaheuristic consists of a tabu search (TS) to solve the hub location subproblem and a local search for assigning terminals to hubs. The solution provided by the metaheuristic is used as an initial upper bound in the branch-and-cut algorithm and to limit the number of variables considered by the exact method. In addition to the best solution, the metaheuristic produces also a subset of nodes that represents, in a sense, the best potential locations for the hubs. The hubs selected in the best solution belong to this subset, as well as the two other hubs which appear most often in the best solutions found by the metaheuristic. This set is called the *concentration set*. The resulting reduced problem, where hubs can be chosen only among the nodes of the concentration set, is called the *concentrated problem*, and is the problem solved using the branch-and-cut method.

A comparison between the metaheuristic proposed in [104], PrevTS, and the one we describe in Section 2.4 is presented in Section 2.4.6.5.

2.4 A strategic oscillation algorithm

The structure of a neighborhood in tabu search goes beyond that used in local search by embracing the types of moves used in constructive and destructive processes (where the foundations for such moves are accordingly called constructive neighborhoods and destructive neighborhoods). Following basic tabu search principles, memory structures can be implemented within a constructive process to favor (or avoid) the inclusion of certain elements in the solution previously identified as attractive (or unattractive). Such expanded uses of the neighborhood concept reinforce a fundamental perspective of TS, which is to define neighborhoods in dynamic ways that can include serial or simultaneous consideration of multiple types of moves.

This dynamic neighborhood approach applies not only to the types of neighborhoods used in "solution improvement methods" (sometimes called "local search methods") but also applies to constructive neighborhoods used in building solutions from scratch - as opposed to transitioning from one solution to another. Although it is commonplace in the metaheuristic literature to restrict the word "neighborhood" to refer solely to transitions between solutions as embodied in improvement methods, constructive neighborhoods have been proposed as an important ingredient of search processes from the very beginning of the TS methodology, as documented by Glover and Laguna [44]. Nevertheless, tabu search methods for exploiting constructive neighborhoods have rarely been applied in computational studies.

The strategic oscillation methodology (SO) [41, 44] is closely linked to the origins of tabu search, and operates by orienting moves in relation to a critical level, as identified by a stage of construction. In particular, we consider a constructive/destructive type of strategic oscillation, where constructive steps "add" elements and destructive steps "drop" elements.

More recently, constructive and destructive neighborhoods have been applied within a simplified and effective method known as iterated greedy (IG) [51], which generates a sequence of solutions by iterating over a greedy constructive heuristic which, like strategic oscillation, uses two main phases: *destruction* and *construction*. IG is an easy-to-implement memoryless metaheuristic that has exhibited state-of-the-art performance in some settings (see for example [33, 63, 91]). We sketch the form of this method because its simplicity gives a convenient foundation for embedding it in a more complete strategic oscillation approach.

As shown in Algorithm 1, the IG method starts from a complete initial solution S (Initialise()) and then iterates through a main loop. The loop first generates a partial candidate solution S_p by removing a fixed number of elements (n_h hubs and terminals in our case) from the complete candidate solution S (Destruction-phase(S, n_h)) and next reconstructs a complete solution S_c starting with S_p (Construction-phase(S_p)). In the local search phase (Local-Search-phase(S_c)), an improvement procedure is performed in order to find better solutions near the reconstructed solution. Before continuing with the next loop, an acceptance criterion (AcceptanceCriterion(S, S_i)) decides whether the solution returned by the local search procedure, S_i, becomes the new incumbent solution. The process iterates through these phases until a computation time limit t_{max} is reached. The best solution, S_{best}, generated during the iterative process is kept to provide the final result.

```
   Input: G, t_max, n_h
   Output: S_best
 1 S ← Initialise();
 2 S_best ← S;
 3 while t_max is not reached do
 4 |   S_p ← Destruction-phase(S, n_h);
 5 |   S_c ← Construction-phase(S_p);
 6 |   S_i ← Local-Search-phase(S_c);
 7 |   if S_i is better than S_best then
 8 |   |   S_best ← S_i;
 9 |   if AcceptanceCriterion(S, S_i) then
10 |   |   S ← S_i;
```

Algorithm 1: Iterated Greedy pseudocode

We have considered two different acceptance criteria in the scheme shown in Algorithm 1:

- *Replace–if–better acceptance criterion.* The new solution is accepted only if it provides a better objective function value [105].
- *Random–walk acceptance criterion.* An IG algorithm using the replace–if–better criterion may lead to stagnation situations of the search due to insufficient diver-

sification [92]. At the opposite extreme is the random–walk acceptance criterion, which always applies the destruction phase to the most recently visited solution, irrespective of its objective function value. This criterion clearly favors diversification over intensification, because it promotes a stochastic search in the space of local optima.

The metaheuristic we propose for solving the capacitated single assignment hub location problem with modular link capacities integrates the iterated greedy approach within the strategic oscillation method by including simple recency and frequency memory strategies derived from tabu search, as proposed in the original SO methodology.

2.4.1 Finding an initial feasible solution

We define a feasible solution S to be an assignment of the terminal nodes to hubs in such a way that traffic from every origin to every destination can be transferred using these hubs.

Let h be a candidate location node for a hub. For any node j that can be assigned to h, with cost C_{jh}, we have to consider that all the traffic from and to j has to be routed through h. In order to evaluate the attractiveness of h as a hub, $g(h)$, we consider first the nodes with the lowest C_{jh} value, adding as many nodes as the capacity permits. Let us assume, without loss of generality, that they are $j_1, \ldots, j_{u(h)}$. In mathematical terms,

$$g(h) = \frac{C_{hh}}{u(h)} + \sum_{s=1}^{u(h)} C_{j_s h},$$

where the first term in the expression corresponds to the installation cost per assigned node.

The hub h_1 with lowest evaluation $g(h)$ is selected as a hub and the terminals used in the computation of $g(h_1)$ are assigned to it. Then, the attractiveness function is computed again for the remaining nodes without considering the terminals already assigned to h_1. This iterative procedure is applied until we have selected enough hubs to assign all the nodes in the network.

At this stage, a feasible solution $S = (H, \mathbf{A})$ is available, where H is the set of hubs, and $\mathbf{A}$ the set of assignments. We represent by (i, h) the assignment of terminal i to hub h. The set $\mathbf{A}$ contains the n pairs reflecting these assignments. Since hub h is assigned to itself, $\mathbf{A}$ contains the pair (h, h). Moreover, $\mathbf{A}^h$ denotes the set of nodes assigned to h, i.e. $\mathbf{A}^h = \{i \in V : (i, h) \in \mathbf{A}\}$, and E_B represents the set of backbone edges, defined as $E_B = \{\{k, l\} : k, l \in H, k < l\}$.

2.4.2 Evaluation of a feasible solution

Different costs are involved when evaluating S:

- The fixed cost of opening/installing hubs.

- The fixed cost of assigning each terminal to its associated hub.
- The cost of installing the backbone edges needed to transfer the traffic between hubs. This cost is computed as follows. Given two hubs k and l, the total amount of traffic on the arc $(k,l) \in A$ is obtained as

$$z_{kl} = \sum_{i \in A^k} \sum_{j \in A^l} t_{ij}.$$

Then, the maximum amount of traffic that will travel through the edge $\{k,l\}$ is $T_{kl} = max\{z_{kl}, z_{lk}\}$. Since each edge $\{k,l\}$ has a maximum capacity Q^b, it will be necessary to replicate this edge $w_{kl} = \left\lceil \frac{T_{kl}}{Q^b} \right\rceil$ times. Given that R_{kl} is the cost of installing a copy of edge $\{k,l\}$, the total cost is $\sum_{\{k,l\} \in E_B} R_{kl} w_{kl}$.

2.4.3 Destruct and construct to improve the hub selection

Following the strategy described in the iterated greedy approach, once a solution S is constructed (using $g(h)$ to guide the process), we partially deconstruct it by removing some of its elements, obtaining S_p. In our context, it means that we deselect some of its hubs, as well as some of the nodes assigned to the remaining selected hubs (these deselected nodes are denoted by $\widetilde{\mathbf{A}}^h, h \in H_{S_p}$). Note that the terminals that were assigned to the unselected hubs are now unassigned. We call the unassigned nodes, including the deselected hubs, *orphan* nodes, and the set of all of them, $(\bigcup_{h \in H_S \setminus H_{S_p}} \mathbf{A}^h) \cup (\bigcup_{h \in H_{S_p}} \widetilde{\mathbf{A}}^h)$, is denoted as $\mathbf{O}$.

The first step in our reconstruction process is to assign as many orphan nodes as possible to the hubs belonging to S_p (thus removing them from set $\mathbf{O}$). Afterwards, we select as a new hub the node $h^* \in \mathbf{O}$ with the lowest $g(h)$ value and assign to it the terminals used in the computation of $g(h^*)$. We remove h^* and its terminal nodes from $\mathbf{O}$, and iteratively perform more construction steps until all the nodes in $\mathbf{O}$ have been assigned or selected as hubs, obtaining a new feasible solution S_c. This destructive-constructive process is repeated until a stopping criterion is met, which in our algorithm is simply a maximum number of iterations. A parameter δ indicates the percentage of the total number of hubs ($|H|$), as well as the percentage of terminals assigned to the still selected hubs, that will be destructed. For example, a value of 0.3 for δ would mean that a 30% of the hubs are removed and that a 30% of the nodes assigned to the remaining hubs are also removed. In our SO method, the hubs are removed from the solution at random, while the removed terminals are selected according to its assignment cost. Terminals with higher assignment costs to their associated non-removed hubs are selected and destructed. Other alternatives to determine the terminals that will be unassigned from the hubs have been tried and tested in a small subset of instances. They were based on costs $C_{jh} - \overline{C}_j$, $C_{jh}/\overline{C}_j$, and $C_{jh}/\widehat{C}_j$, where $\overline{C}_j = min\{C_{jh}, h \in H_{S_p}\}$ and $\widehat{C}_j = min\{C_{jh}, h \in H_{S_p} \cup \mathbf{O}\}$. However, in all the instances tested the results obtained were worse than those associated with the proposed strategy based only on the assignment costs C_{jh}.

This procedure implies a diversification component in the search which balances the intensification of our greedy constructive algorithm. In Section 2.4.6 we study the performance of the proposed algorithm, denoted as SO1, for different values of δ.

We also consider a second SO algorithm, SO2, where we use a classic *tabu list* (the customary type of recency memory). In SO2 we construct an initial solution and randomly remove a percentage δ of the hubs. The removed hubs are added to a tabu list and become tabu for a given number of iterations (constructions) denoted by τ (*tabu tenure*). The same construction method as SO1 is applied to reconstruct the solution by selecting new non-tabu hubs. In order to speed up the process and search for new solutions, we have included a slight modification in the assignment of orphan nodes in the SO2 procedure. In particular, when we check if an orphan node can be assigned to a non-removed hub, we follow the order given by the tabu list. We first try to assign the tabu nodes since they cannot be hubs in this iteration. Moreover, within the tabu nodes, we try first those that recently gain the tabu status (those that we strongly forbid to be hubs). As is usually done in tabu search implementations, we include an aspiration criterion to override the tabu status by permitting, in this case, a tabu node to be a hub if the capacity constraints would compel this in the assignment process (*aspiration by feasibility*).

2.4.4 Improvements on the assignments

When a new feasible solution $S_c = (H, \mathbf{A})$ is obtained, an improvement procedure on the assignment of terminals to hubs is applied. Two neighborhoods, N_{pairs} and N_{alone}, are proposed to improve S_c:

N_{pairs} implements a classical exchange in which two terminals i and j, assigned to hubs k and l respectively, swap their corresponding hubs. This exchange can be done when nodes i and j are not hubs, they are assigned to different hubs, and when the new assignments do not violate the capacity constraints. To compute the cost of this exchange:

- We update only the assignment costs of i and j: $C_{ik} + C_{jl}$ is subtracted from the total assignment cost and the new assignment costs $C_{il} + C_{jk}$ are added.
- The cost of the backbone edges also needs to be recomputed. Given a backbone edge $\{p, q\}$, the new traffic T'_{pq} traversing $\{p, q\}$ is $T'_{pq} = max\{z'_{pq}, z'_{qp}\}$, where the values of z'_{pq} and z'_{qp} are computed as follows:
  - If $p \neq k$ or l and $q \neq k$ or l, the traffic through backbone edge $\{p, q\}$ does not change, $z'_{pq} = z_{pq}$ and $z'_{qp} = z_{qp}$.
  - If exactly one end node of $\{p, q\}$ is k or l, for instance $p = k$ $(q \neq l)$, in order to compute the new traffic from hub p to hub q we modify the old one by subtracting the traffic from i to all the terminals assigned to q and by adding the traffic from j to the terminals assigned to q, i.e.:

$$z'_{pq} = z_{pq} - \sum_{s \in A^q} t_{is} + \sum_{s \in A^q} t_{js}.$$

Analogously,

$$z'_{qp} = z_{qp} - \sum_{s \in A^q} t_{si} + \sum_{s \in A^q} t_{sj}.$$

– Similarly, if $\{p, q\} = \{k, l\}$,

$$z'_{pq} = z_{pq} - \sum_{s \in A^l} t_{is} - \sum_{s \in A^k \setminus \{i\}} t_{sj} + \sum_{s \in A^l \cup \{i\}} t_{js} + \sum_{s \in A^k} t_{si}$$

and

$$z'_{qp} = z_{qp} - \sum_{s \in A^k} t_{js} - \sum_{s \in A^l \setminus \{j\}} t_{si} + \sum_{s \in A^k \cup \{j\}} t_{is} + \sum_{s \in A^l} t_{sj}.$$

From the new traffic T'_{pq} traversing the backbone edges $\{p, q\} \in E_B$, we compute the number of copies that are needed for each edge and its cost, $\left\lceil \frac{T'_{pq}}{Q^b} \right\rceil \times R_{pq}$.

Only if the total cost of the new assignment is lower than the cost of the current solution (first improvement strategy), the exchange is done. This procedure is performed for each pair of terminals i and j, which are enumerated by means of a natural ordering.

Sometimes N_{pairs} turns out to be a poor neighborhood as it is quite restrictive. For this reason we also propose N_{alone}, which implements another classical movement: an insertion. In N_{alone}, a terminal i, previously assigned to hub k, is now assigned to another hub l. To compute the cost of the new assignment:

- Only the assignment cost of i needs to be updated from the total assignment cost: C_{ik} is subtracted from the total cost and C_{il} is added.
- The cost of the backbone edges also needs to be recomputed. Let $i \in \mathbf{A}^k$. We try to assign i to another hub l in order to get a cost reduction. As in N_{pairs}, given a backbone edge $\{p, q\}$, the values of z'_{pq} and z'_{qp} to obtain T'_{pq} are computed as follows:
  - If $p \neq k$ or l and $q \neq k$ or l, the traffic through backbone edge $\{p, q\}$ do not change, $z'_{pq} = z_{pq}$ and $z'_{qp} = z_{qp}$.
  - If $p = k$ or $q = k$, since hub k looses its assigned node i (suppose $p = k$),

$$z'_{pq} = z_{pq} - \sum_{s \in A^q} t_{is} \quad \text{and} \quad z'_{qp} = z_{qp} - \sum_{s \in A^q} t_{si}.$$

  - If $p = l$ or $q = l$, since node i is assigned now to hub l (suppose $p = l$),

$$z'_{pq} = z_{pq} + \sum_{s \in A^q} t_{is} \quad \text{and} \quad z'_{qp} = z_{qp} + \sum_{s \in A^q} t_{si}.$$

– If $p = k$ and $q = l$,

$$z'_{pq} = z_{pq} - \sum_{s \in A^l} t_{is} + \sum_{s \in A^k} t_{si} \quad \text{and} \quad z'_{qp} = z_{qp} - \sum_{s \in A^l} t_{si} + \sum_{s \in A^k} t_{is}.$$

As in N_{pairs}, from the new traffic T'_{pq} we compute the cost of the copies needed for each backbone edge.

Again, the exchange is done only if the new cost is lower than the cost of the current solution.

2.4.5 Singular solutions

There are two types of singular solutions that have to be examined: solutions where there is **only one hub** and all nodes are assigned to it, and solutions where **all nodes are hubs** and each node is assigned to itself. These are singular solutions because in both cases the assignment of terminals to hubs cannot be modified, rendering the above local search procedures useless.

Once the whole process of destruction-construction ends, if such special cases correspond to feasible solutions, we compare these to the best solution found during the strategic oscillation process.

2.4.6 Computational experiments

In this section we describe the computational experiments performed to test the efficiency of the proposed strategic oscillation metaheuristic. The metrics that we use to measure the performance of the algorithms are:

- Value: Average objective value of the best solutions obtained with the algorithm on the instances considered in the experiment.
- Dev: Average percentage deviation from the best-known solution (or from the optimal solution, if available).
- Best: Number of instances for which a procedure is able to find the best-known solution.
- CPU: Average computing time in seconds employed by the algorithm.

2.4.6.1 Test instances

To test the performance of the proposed metaheuristic, we have generated a new set of 170 instances from CAB, AP, and USA423 sets. Unfortunately, it has not been possible to obtain the original instances used by Yaman and Carello [104]. A detailed description of our instances follows:

1. The **CAB** (Civil Aviation Board) data set. From this original file, a total of 23 instances with 10, 15, 20 and 25 nodes have been generated.

2. The **AP** (Australian Post) data set. We have extended this set of instances by generating 70 instances with n ranging from 10 to 200. Regarding the flows between nodes, these instances do not have symmetric flows (i.e., for a given pair of nodes i and j, t_{ij} is not necessarily equal to t_{ji}). Moreover, in the original instance some flows from one node to itself are positive (i.e., $t_{ii} > 0$ for a given i).

3. The **USA423** data set. From the original data, 77 instances have been generated with n ranging from 20 to 250.

Each original instance includes the traffic and the traveling cost-per-unit matrices. From these two matrices, we have generated the matrices t_{ij}, C_{ij}, R_{kl}, and the capacity values Q^a, Q^b, Q^h. While the t_{ij} traffic matrix is the original one, matrices C_{ij} and R_{kl} have been created to incorporate the assignment, installation, and inter-hub transportation costs.

The experiments that follow are divided into two main blocks. The first block (Section 2.4.6.2) is devoted to study the behavior of the components of the solution procedure, as well as to determine the best value for the search parameter δ. The second block of experiments (Sections 2.4.6.4 and 2.4.6.5) has the goal of comparing our procedure with the best published methods. To be able to test the effectiveness of our strategies, the first set of experiments is performed on a subset of instances to test how well our choices generalize to the entire set of problems.

From the 170 instances derived from the CAB, AP and USA423 data sets, the tuning experiments are performed on the following subset of 36 instances: 3 instances from the CAB set with $15 \leq n \leq 25$, 21 instances with $10 \leq n \leq 195$ from the AP set, and 12 instances with $20 \leq n \leq 150$ from USA423. We refer to these 36 instances as the *training set* and to the remaining 134 instances as the *testing set*.

2.4.6.2 Parameter calibration

We initially perform several experiments to study the constructive-destructive method described in Section 2.4 with respect to solution quality and diversification power. In all the preliminary experiments we executed the strategic oscillation method for 100 global iterations.

We first compare in SO1 the random–walk (RW) and the replace–if–better (RIB) acceptance criteria for different values of δ. The results are shown in Table 2.1. This table reports the average percentage deviation of the solution values with respect to the best value obtained in this experiment, Dev, as well as the number of best solutions found, # Best. As can be seen, the best results are obtained with the values $\delta = 30\%$ and $\delta = 40\%$ for the replace–if–better acceptance criterion, showing that replace–if–better is significantly better than the random–walk criterion. In order to compare the sets of results for the two selected values of δ, we have performed the Wilcoxon test, a well-known non-parametric test for pairwise comparisons, which answers the question: Do the two samples (the solutions obtained with both values of δ in our case) represent two different populations? The resulting probability value of 0.06 indicates that there

is no a significant difference between them. Despite this, from the results shown in the table, we have chosen the value 30% for δ. Therefore, from now on, this variant (SO1 with replace–if–better strategy and $\delta = 30\%$, denoted SO1 for short, is the one selected for the rest of experiments.

Table 2.1. Comparison of the two acceptance criteria for different values of δ

			Dev				# Best			
Strategy	**Size**	**# Inst**	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**
RW	small	12	8.5%	5.9%	5.5%	6.0%	1	1	1	0
	medium	12	11.9%	11.7%	9.7%	10.5%	0	0	0	0
	large	12	9.8%	9.2%	10.3%	9.4%	0	0	0	0
	summary	36	10.0%	8.9%	8.5%	8.6%	1	1	1	0
RIB	small	12	4.1%	1.7%	4.9%	3.1%	2	6	0	1
	medium	12	0.5%	4.8%	6.5%	7.1%	8	2	2	0
	large	12	2.2%	3.6%	5.3%	4.6%	7	2	1	2
	summary	36	2.3%	3.3%	5.6%	4.9%	17	10	3	3

The effectiveness of generating multiple solutions in our strategic oscillation method has also been tested, since this algorithm relies on obtaining good and diverse solutions to serve as starting points for the local search procedures. Figure 2.2 shows the boxplot of the SO1 method with and without the local search on a representative instance (150-1000-69-60-80-1-69-USA). The left boxplot shows the values of the 100 solutions found without applying the local search procedures, while the one on the right shows the results obtained after applying them. This plot clearly shows that different solutions are obtained in most of the runs. As expected, the variant with the local search obtains better solutions, as compared with the one without improvements, but with lower dispersion.

Another experiment was carried out to calibrate the value of the tabu tenure parameter, τ, in SO2. Since we did not observe any significant differences among the tested values, we do not report the obtained results. A default value for parameter τ of 4 has been chosen.

2.4.6.3 Algorithm designs

In this section we compare the two strategic oscillation variants according to the memory structure used. The results obtained in this experiment are summarized in Table 2.2, where we report the number of best solutions found, out of 36, by each variant, as well as the average computing time used. This table shows that SO1 obtains better solutions than SO2. In particular, SO1 is able to match all the best known solutions, while SO2 only obtains 11 out of 36 instances, which represents an average percentage deviation of 9.2%. As a result of this experiment, from now on we select SO1 as the focus of our

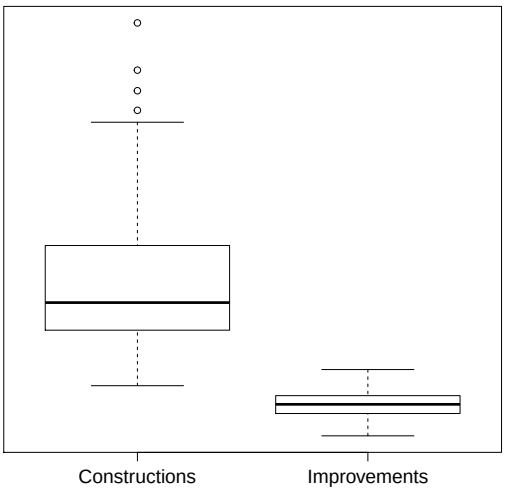

Figure 2.2. Boxplot of 100 iterations for instance 150-1000-69-60-80-1-69-USA

additional experiments.

Table 2.2. Comparison between SO1 and SO2

	SO1			SO2		
	Dev	**# Best**	**CPU**	**Dev**	**# Best**	**CPU**
small	0.0%	12	0.32	9.2%	4	0.24
medium	0.0%	12	6.57	13.2%	1	3.41
large	0.0%	12	181.65	5.3%	6	151.56
summary	0.0%	36	62.85	9.2%	11	51.74

2.4.6.4 Comparison with optimal values

In Section 2.2 we have described the formulation proposed in [104] for the CSHLPMLC, which contains quadratic constraints. We used CPLEX to solve 30 instances with n ranging from 10 to 30, and only 11 instances could be solved to optimality. As far as we know, solving such an instance depends on the properties of its constraint matrix. The results obtained with CPLEX for the optimally solved instances are reported in Table 2.3, as well as those obtained with the SO1 metaheuristic. In particular, it shows the average percentage deviation with respect to the optimal solution obtained with CPLEX and the computing time used by each method.

Table 2.3 shows that the SO1 method is able to obtain the optimal solution in all

cases. As expected, CPLEX required much more computing time than SO1 to obtain the optimal value. It is worth mentioning that CPLEX used the total number of cores of the CPU (8 cores in our case) compared to only a few used by the SO1 algorithm.

Table 2.3. Comparison between CPLEX and SO1 on small-size instances

	CPLEX		SO1	
Instance	**Value**	**CPU**	**Dev**	**CPU**
A1H	72710	24.02	0.0%	0.20
A2H	105477	254.30	0.0%	0.17
A3H	77516	23.69	0.0%	0.23
A4H	188200	139.00	0.0%	0.20
B1H	45636	75.80	0.0%	0.30
B2H	23818	4.66	0.0%	0.34
B3H	51387	31.08	0.0%	2.07
B4H	25410	4.71	0.0%	0.90
C1H	43526	4297.98	0.0%	0.50
C2H	43505	3304.48	0.0%	0.78
C3H	57905	33891.33	0.0%	1.08

2.4.6.5 Comparison with a tabu search algorithm

Since it was not possible to compare the SO1 procedure with CPLEX on larger instances, in order to test its behavior we have compared SO1 with the implementation we have done of the tabu search algorithm described in [104] (PrevTS). In this section we use the instances in the training and testing sets. Table 2.4 shows the average percentage deviation (Dev) with respect to the best solution known (obtained either with SO1 or with PrevTS), the number of best solutions found (# Best), and the computing time (CPU) of both methods on the 36 training set instances.

Table 2.4. Comparison between PrevTS and SO1 on the training set instances

		PrevTS			SO1		
Type	**# Inst**	**Dev**	**# Best**	**CPU**	**Dev**	**# Best**	**CPU**
small	12	13.1%	1	1.4	0.0%	12	0.9
medium	12	15.4%	0	18.6	0.0%	12	14.3
large	12	19.8%	3	240.5	0.8%	10	221.4
summary	36	16.1%	4	86.8	0.3%	34	78.8

Table 2.4 clearly shows that our SO1 method outperforms the previously proposed tabu search approach. In particular, SO1 obtains the best known solution in 34 out of 36

instances, while PrevTs is only able to do this in four cases. Furthermore, SO1 solutions deviate from the best known solutions by an average of 0.3% while PrevTS solutions deviate by 16.1%. On average, SO1 is also faster than PrevTS. It must be noted that, as mentioned in [104], the objective of the authors when developing PrevTS was to obtain relatively good initial solutions for their exact method, while our SO1 has been designed to obtain high quality solutions in short running times.

We compare now the performance of SO1 and PrevTS on the testing set, to measure the ability of our algorithm (SO1) to target instances not included in the training set. In particular, we consider 134 instances classified according to their size into small, medium and large. Note that the large set includes instances with $n = 250$. Table 2.5 shows

Table 2.5. Comparison between PrevTS and SO1 on the testing set instances

		PrevTS			SO1		
Type	**# Inst**	**Dev**	**# Best**	**CPU**	**Dev**	**# Best**	**CPU**
small	33	8.5%	7	1.0	0.0%	32	0.6
medium	25	10.9%	3	88.1	0.4%	22	41.1
large	76	11.7%	10	1263.1	0.6%	69	545.8
summary	134	10.8%	20	733.1	0.5%	123	317.4

the results of this experiment in terms of the average deviation with respect to the best known value (Dev) and number of instances in which each method is able to match this best value (# Best). Table 2.5 shows that SO1 obtains better results than PrevTS in significantly lower running times. Specifically, PrevTS has an average deviation of 10.8% obtained in 733.1 seconds, while SO1 has an average deviation of 0.5% obtained in 317.4 seconds. Tables 2.6, 2.7, 2.8 and 2.9 show the individual results of this experiment to provide the reader with a detailed information for further experimentation.

In our last experiment, we compare SO1 and PrevTs over the time. Figure 2.3 shows the evolution throughout the search of the best value obtained with each method on a representative instance. The search profile depicted in this figure shows that SO1 (dashed line) obtains better solutions than PrevTS from the very beginning of the search and that PrevTS needs some time to reach relatively good solutions.

2.4.7 Concluding remarks

We have proposed a new metaheuristic based on strategic oscillation for the capacitated single assignment hub location problem with modular link capacities. This problem was introduced by Yaman and Carello [104] as an interesting variant of the classical hub location problem in which the cost of using edges is not linear but stepwise, and the hubs are restricted in terms of transit capacity rather than in the incoming traffic. Our proposed method incorporates several designs for constructive and destructive algorithms, together with associated local search procedures. The computational experiments show

Table 2.6. SO1 and PrevTS on small size instances

	PrevTS			SO1		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
10_600_89_60_40_1_60_CAB	237701	0.0%	0.0	237701	0.0%	0.0
10_700_50_60_8_1_60_AP	302921	10.6%	0.0	273801	0.0%	0.0
10_700_50_60_8_1_60_CAB	243854	0.0%	0.0	243854	0.0%	0.0
10_700_69_40_8_1_50_CAB	246610	0.0%	0.0	246610	0.0%	0.0
10_800_60_60_6_1_69_AP	283138	15.3%	0.0	245513	0.0%	0.0
10_800_60_60_6_1_69_CAB	238144	0.0%	0.0	238144	0.0%	0.0
10_800_60_80_8_1_80_CAB	240872	0.0%	0.0	240872	0.0%	0.0
15_500_50_60_40_1_60_CAB	290334	0.3%	0.0	289339	0.0%	0.0
15_600_80_89_6_1_69_CAB	291191	0.0%	0.1	291780	0.2%	0.0
15_600_80_89_8_1_60_CAB	302735	0.0%	0.0	302735	0.0%	0.0
15_700_89_60_40_1_60_CAB	289953	0.0%	0.0	289953	0.0%	0.0
15_800_50_60_40_1_60_CAB	295521	1.9%	0.1	290026	0.0%	0.0
15_900_80_89_40_1_60_CAB	290970	0.3%	0.1	290134	0.0%	0.0
20_700_50_60_8_1_60_AP	405786	3.0%	0.2	393788	0.0%	0.1
20_700_50_60_8_1_60_CAB	205817	17.5%	0.2	175103	0.0%	0.1
20_700_50_60_8_1_60_USA	137854	8.5%	0.1	127058	0.0%	0.1
20_700_69_40_8_1_50_AP	432081	3.6%	0.3	417187	0.0%	0.1
20_700_69_40_8_1_50_CAB	214323	17.2%	0.2	182900	0.0%	0.3
20_800_60_60_6_1_69_CAB	191608	16.6%	0.2	164351	0.0%	0.3
20_800_60_80_8_1_80_AP	385461	4.5%	0.6	368951	0.0%	0.2
20_800_60_80_8_1_80_CAB	197141	14.3%	0.2	172460	0.0%	0.3
20_800_60_80_8_1_80_USA	130508	7.8%	0.2	121071	0.0%	0.0
20_900_80_89_40_1_80_CAB	157723	4.0%	0.1	151598	0.0%	0.1
25_600_80_60_6_1_40_CAB	245429	16.5%	0.5	210724	0.0%	0.4
25_600_80_89_6_1_69_CAB	223798	9.4%	0.4	204602	0.0%	0.1
25_600_80_89_8_1_60_CAB	242246	16.8%	0.5	207446	0.0%	0.4
25_650_69_69_6_1_50_CAB	235844	16.7%	0.4	202119	0.0%	0.1
25_650_69_69_6_1_70_CAB	221333	40.5%	0.8	157539	0.0%	0.5
25_800_89_60_40_1_80_CAB	182474	1.1%	0.2	180408	0.0%	0.3
25_900_80_89_40_1_80_CAB	190129	5.3%	0.2	180586	0.0%	0.3
30_600_80_89_8_1_60_AP	392149	6.9%	0.1	366726	0.0%	0.2
30_700_69_40_8_1_50_USA	246833	26.6%	0.8	194970	0.0%	0.2
35_600_80_89_8_1_60_AP	478595	10.1%	0.3	434611	0.0%	0.6
35_600_80_89_8_1_60_USA	232360	12.8%	1.3	206063	0.0%	0.9
35_700_80_50_8_1_69_AP	482376	7.9%	1.5	447032	0.0%	0.3
40_600_80_89_8_1_60_USA	339760	23.2%	1.5	275732	0.0%	0.4
40_700_80_50_8_1_69_AP	511799	10.4%	2.3	463568	0.0%	1.2
40_700_80_50_8_1_69_USA	330583	18.2%	3.1	279572	0.0%	2.5
45_600_80_89_8_1_60_AP	578870	7.8%	2.3	536953	0.0%	2.5
45_700_69_40_8_1_50_AP	611473	0.6%	2.5	607672	0.0%	1.8
45_700_69_40_8_1_50_USA	339338	5.7%	3.3	320991	0.0%	3.0
50_600_80_89_8_1_60_USA	518779	48.5%	3.2	349460	0.0%	3.4
50_700_69_40_8_1_50_AP	669803	7.7%	9.1	622172	0.0%	3.0
50_700_80_50_8_1_69_AP	608050	7.6%	5.3	565087	0.0%	2.5
50_700_80_50_8_1_69_USA	374127	12.9%	7.4	331514	0.0%	2.9

Table 2.7. SO1 and PrevTS on medium size instances

	PrevTS			SO1		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
55_500_60_69_60_1_50_AP	525642	0.0%	8.7	542146	3.1%	3.7
55_500_60_69_60_1_50_USA	378822	15.6%	9.6	327659	0.0%	5.2
55_800_69_50_80_1_60_AP	609771	0.0%	7.1	633097	3.8%	4.8
55_800_69_50_80_1_60_USA	416346	18.4%	5.3	351759	0.0%	3.2
60_500_60_69_60_1_50_AP	643461	10.9%	1.9	580361	0.0%	4.1
60_500_60_69_60_1_50_AP	615449	11.0%	5.9	554635	0.0%	6.3
60_600_60_69_60_1_69_USA	391313	19.1%	18.1	328468	0.0%	5.1
60_800_69_50_80_1_60_AP	702917	7.0%	4.9	656894	0.0%	5.8
60_800_69_50_80_1_60_USA	424505	20.4%	18.9	352466	0.0%	7.7
65_500_60_69_60_1_50_AP	665600	10.6%	12.3	601984	0.0%	10.4
65_600_60_69_60_1_69_AP	616365	6.2%	4.1	580367	0.0%	7.0
65_600_60_69_60_1_69_USA	435220	26.3%	8.3	344542	0.0%	7.5
65_800_69_50_80_1_60_USA	402210	8.2%	19.0	371628	0.0%	9.1
70_500_60_69_60_1_50_AP	733741	11.3%	8.2	659281	0.0%	8.7
70_600_60_69_60_1_69_AP	658758	8.4%	22.7	607509	0.0%	8.6
70_600_60_69_60_1_69_USA	428069	15.8%	38.4	369507	0.0%	14.0
70_800_69_50_80_1_60_AP	728949	0.0%	27.7	742895	1.9%	14.5
75_500_60_69_60_1_50_USA	698370	25.7%	14.7	555545	0.0%	11.2
75_600_60_69_60_1_69_AP	747985	4.3%	14.6	717444	0.0%	13.4
75_600_60_69_60_1_69_USA	568456	15.8%	77.6	491048	0.0%	21.0
75_800_69_50_80_1_60_AP	868528	5.9%	37.0	820259	0.0%	20.7
80_500_60_69_60_1_50_AP	780236	11.5%	58.8	699835	0.0%	35.5
80_500_60_69_60_1_50_USA	724155	15.6%	92.9	626601	0.0%	30.1
80_800_69_50_80_1_60_AP	988201	15.6%	15.7	854957	0.0%	15.2
80_800_69_50_80_1_60_USA	690769	5.8%	94.4	653204	0.0%	32.4
85_500_60_69_60_1_50_AP	886289	14.7%	152.4	772714	0.0%	91.2
85_500_60_69_60_1_50_USA	948813	22.2%	38.5	776354	0.0%	17.9
85_800_69_50_80_1_60_AP	1003536	16.0%	96.6	865260	0.0%	104.7
90_500_60_69_60_1_50_USA	904709	23.7%	278.0	731210	0.0%	40.1
90_600_60_69_60_1_69_AP	840072	3.6%	33.0	810722	0.0%	27.2
90_600_60_69_60_1_69_USA	742016	5.6%	166.9	702336	0.0%	84.9
90_800_69_50_80_1_60_AP	1124161	20.0%	23.4	936606	0.0%	29.0
95_500_60_69_60_1_50_AP	972840	15.7%	68.4	841016	0.0%	70.9
95_500_60_69_60_1_50_USA	964716	20.6%	55.3	800139	0.0%	26.7
95_600_60_69_60_1_69_AP	859425	6.5%	86.8	807075	0.0%	89.3
95_600_60_69_60_1_69_USA	723454	7.1%	284.4	675503	0.0%	140.5
100_500_60_69_60_1_50_USA	877730	13.1%	516.4	776085	0.0%	171.0

Table 2.8. SO1 and PrevTS on large instances up to 175 nodes

	PrevTS			SO1		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
110_500_60_69_60_1_50_AP	1451493	38.1%	67.4	1051407	0.0%	44.8
110_600_60_69_60_1_69_AP	1174251	25.2%	154.4	938250	0.0%	52.6
110_700_80_60_89_1_60_USA	1316582	13.4%	261.8	1161100	0.0%	67.7
110_800_69_50_80_1_60_USA	1296900	20.2%	103.6	1079127	0.0%	36.1
110_900_69_50_89_1_60_USA	1204417	4.6%	354.3	1151693	0.0%	61.2
120_500_60_69_60_1_50_AP	1349557	27.1%	113.1	1061782	0.0%	56.8
120_500_60_69_60_1_50_USA	1497383	28.6%	97.0	1164523	0.0%	48.2
120_700_80_60_89_1_60_USA	1247937	0.0%	281.2	1256590	0.7%	94.7
120_900_69_50_89_1_60_USA	1359133	10.9%	377.3	1226069	0.0%	100.7
125_500_60_69_60_1_50_AP	1498128	28.4%	208.4	1167146	0.0%	94.1
125_800_69_50_80_1_60_AP	1713025	23.8%	119.3	1383157	0.0%	66.4
130_600_60_69_60_1_69_AP	1426980	26.7%	146.0	1126476	0.0%	160.2
130_600_60_69_60_1_69_USA	1723043	35.0%	225.1	1276450	0.0%	74.8
130_800_69_50_80_1_60_USA	1663758	36.7%	537.6	1216964	0.0%	150.9
135_600_60_69_60_1_69_AP	1610997	33.0%	263.5	1211303	0.0%	209.3
135_800_69_50_80_1_60_USA	1584036	21.3%	4960.6	1306124	0.0%	180.1
140_500_60_69_60_1_50_AP	1691066	27.7%	579.7	1323974	0.0%	202.5
140_700_80_60_89_1_60_USA	1553768	0.0%	482.1	1561410	0.5%	185.0
140_800_69_50_80_1_60_AP	2130072	36.5%	293.3	1560792	0.0%	215.0
140_900_69_50_89_1_60_USA	1517790	0.0%	325.9	1517822	0.0%	179.4
145_600_80_69_60_1_50_AP	1825162	26.4%	379.8	1443675	0.0%	223.4
145_600_80_69_60_1_50_USA	609564	14.5%	256.3	532241	0.0%	233.0
145_800_69_50_80_1_60_AP	2114457	28.7%	381.0	1643198	0.0%	218.9
145_800_69_50_80_1_60_USA	715853	14.7%	227.1	624365	0.0%	230.2
150_1000_69_60_80_1_69_USA	614369	0.0%	299.1	666691	8.5%	278.6
150_800_69_50_80_1_60_AP	2200307	31.5%	698.3	1672925	0.0%	221.4
150_800_69_50_89_1_60_USA	594115	5.6%	437.9	562570	0.0%	257.6
150_900_69_50_89_1_60_AP	1822301	2.2%	232.0	1782301	0.0%	150.6
150_900_69_60_80_1_89_USA	584419	5.6%	272.3	553654	0.0%	258.1
155_1000_69_60_80_1_69_USA	727403	18.2%	430.0	615630	0.0%	232.5
155_500_60_69_60_1_50_AP	2061735	21.8%	166.2	1692582	0.0%	209.0
155_800_69_50_80_1_60_AP	2049217	28.1%	1408.1	1599216	0.0%	270.6
155_900_69_50_89_1_60_AP	2017260	14.0%	390.2	1769092	0.0%	226.9
155_900_69_50_89_1_60_USA	602944	2.5%	621.3	587975	0.0%	313.8
160_600_60_69_60_1_69_AP	711981	0.0%	183.6	721829	1.4%	363.1
160_800_69_50_80_1_60_AP	749451	0.0%	221.5	869439	16.0%	278.3
160_900_69_50_89_1_60_USA	636074	0.0%	270.2	636298	0.0%	333.4
160_900_80_50_60_1_69_AP	831947	6.7%	883.9	779805	0.0%	365.8
165_1000_69_60_80_1_69_USA	712118	8.5%	784.0	656114	0.0%	272.9
165_800_69_50_80_1_60_AP	1143618	16.9%	166.4	978268	0.0%	246.1
165_800_69_50_80_1_60_USA	767748	11.8%	838.5	686531	0.0%	302.7
170_500_60_69_60_1_50_AP	588784	0.0%	207.8	688647	17.0%	296.1
170_900_69_60_80_1_89_USA	623614	0.0%	872.5	623614	0.0%	298.3
170_900_80_50_60_1_69_AP	882645	6.8%	928.4	826445	0.0%	387.5
175_500_60_69_60_1_50_AP	889474	3.3%	615.2	860907	0.0%	315.0
175_600_60_69_60_1_69_AP	710911	5.2%	695.6	675774	0.0%	283.1
175_800_69_50_80_1_60_USA	790880	2.4%	1358.1	772340	0.0%	367.4
175_900_69_60_80_1_89_USA	652884	7.4%	1466.3	607724	0.0%	360.6

Table 2.9. SO1 and PrevTS on large instances up to 250 nodes

	PrevTS			SO1		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
180_1000_69_60_80_1_69_USA	754360	8.2%	1536.1	697158	0.0%	379.5
180_600_60_69_60_1_69_AP	840658	10.0%	764.8	764080	0.0%	380.4
180_600_89_60_69_1_80_USA	594721	10.7%	1510.2	537212	0.0%	367.2
180_800_89_69_89_1_89_AP	1043978	17.8%	796.1	885947	0.0%	310.7
185_500_60_69_60_1_50_AP	885253	0.0%	282.6	885253	0.0%	486.7
185_600_80_89_89_1_89_AP	851397	8.0%	840.7	788688	0.0%	437.3
185_600_89_60_69_1_80_USA	522188	5.4%	782.2	495212	0.0%	417.0
185_800_69_50_80_1_60_AP	796454	0.0%	1595.3	885195	11.1%	510.2
185_900_69_60_80_1_89_USA	615474	0.0%	576.8	615474	0.0%	390.2
190_600_60_69_60_1_69_AP	970614	18.2%	558.1	821318	0.0%	311.4
190_600_80_89_89_1_89_AP	731291	0.0%	1648.0	731127	0.0%	590.1
190_600_89_60_69_1_80_USA	626170	16.5%	1491.6	537256	0.0%	571.5
190_700_89_69_89_1_89_USA	552331	0.0%	1276.5	552331	0.0%	577.8
190_800_69_50_80_1_60_AP	1212471	16.0%	631.8	1045263	0.0%	559.3
195_600_60_69_60_1_69_AP	759008	6.4%	1971.1	713616	0.0%	618.1
195_800_69_50_80_1_60_AP	1580425	35.3%	617.6	1168347	0.0%	491.6
195_900_89_89_89_1_69_AP	1254423	18.9%	1839.4	1055289	0.0%	623.6
200_500_60_69_60_1_50_AP	868511	25.3%	1641.4	693180	0.0%	766.3
200_700_80_60_89_1_60_USA	866749	6.4%	2353.8	814716	0.0%	660.6
200_800_69_50_80_1_60_AP	815087	6.8%	2419.5	763349	0.0%	745.8
205_800_69_50_80_1_60_USA	953836	8.4%	2046.4	879840	0.0%	780.3
205_900_69_60_80_1_89_USA	815563	15.9%	1604.0	703445	0.0%	750.3
210_800_69_50_80_1_60_USA	937597	6.8%	2462.5	878031	0.0%	873.7
210_900_69_60_80_1_89_USA	775547	8.5%	1055.2	714922	0.0%	840.8
215_800_69_50_80_1_60_USA	903526	2.2%	1784.6	884479	0.0%	825.8
215_900_69_60_80_1_89_USA	826507	11.2%	2103.9	743290	0.0%	907.4
220_800_69_50_80_1_60_USA	1035123	16.9%	1564.6	885251	0.0%	1001.0
220_900_69_60_80_1_89_USA	906900	14.7%	2169.7	790368	0.0%	980.5
225_800_69_50_80_1_60_USA	1094662	2.9%	1368.1	1063619	0.0%	1215.9
225_900_69_60_80_1_89_USA	921238	9.7%	2586.1	839675	0.0%	1160.4
230_800_69_50_80_1_60_USA	1212179	15.6%	2745.3	1048801	0.0%	1242.9
230_900_69_60_80_1_89_USA	977456	13.6%	3057.2	860244	0.0%	1246.3
235_800_69_50_80_1_60_USA	1089635	6.0%	2287.1	1028190	0.0%	1187.2
235_900_69_60_80_1_89_USA	1104375	13.7%	1489.5	971022	0.0%	1274.6
240_800_69_50_80_1_60_USA	1267953	7.7%	3489.7	1177217	0.0%	1283.2
240_900_69_60_80_1_89_USA	1097283	9.2%	2797.9	1004974	0.0%	1497.6
245_800_69_50_80_1_60_USA	1337769	7.6%	3465.0	1243498	0.0%	1596.0
245_900_69_60_80_1_89_USA	1119035	16.6%	1937.0	959396	0.0%	1533.6
250_800_69_50_80_1_60_USA	1219880	4.7%	4679.1	1165109	0.0%	1800.9
250_900_69_60_80_1_89_USA	1075868	9.5%	3418.4	982810	0.0%	1868.1

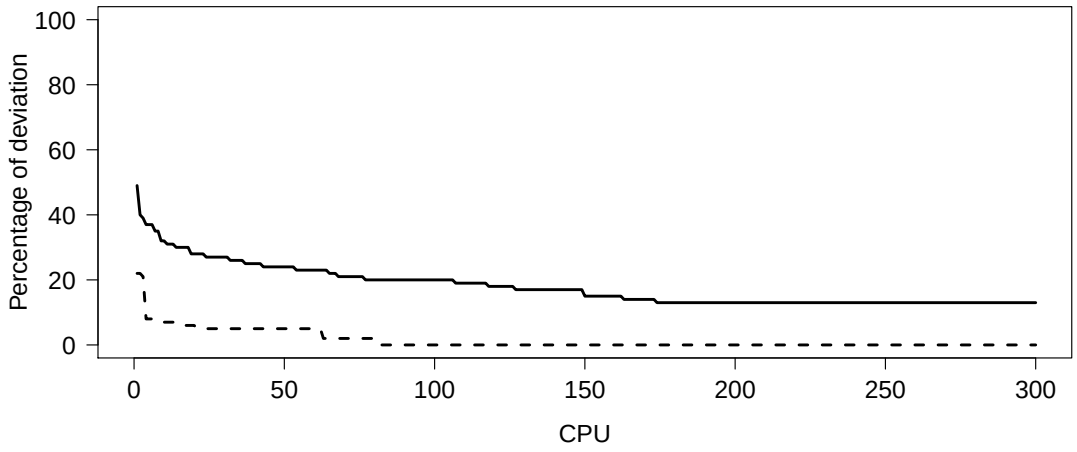

Figure 2.3. Search Profile for SO1 (dashed line) and PrevTs (plain line)

that our algorithm is able to find high-quality solutions in short computing times, and outperforms a previously published tabu search procedure.

We envision that future enhancements of our method may be possible by employing additional strategies derived from the strategic oscillation and tabu search methodology, such as replacing the recourse to randomization with more strategic elements (as by removing hubs strategically using tabu search memory in the destructive phases).

2.5 An adaptive memory programming algorithm

In the previous section we have proposed a heuristic method based on a strategic oscillation over the search space to solve the CSHLPMLC. This method iteratively constructs and partially destructs a solution. In this way, hubs are selected and deselected in search for the optimal set of hubs. This procedure is coupled with two local searches, one based on swapping the assignment of terminals to hubs, and another based on exchanges of terminals to a different hub. The computational experimentation in Section 2.4.6 has showed that this method outperforms the tabu search heuristic in [104], and it is able to match, (see Section 2.4.6.4), the optimal solutions in the small size instances that CPLEX is able to solve.

In this section we propose a new heuristic based on Adaptive Memory Programming (AMP) methodology. We basically introduce memory structures to enhance the performance of our methods. Memory-based strategies, which are the hallmark of the well-known tabu search methodology [44], and coined under the term adaptive memory programming, are founded on a quest for "integrating principles", by which alternative forms of memory are appropriately combined with effective strategies for exploiting

them. Specifically, we propose different construction and local search methods and study the effectiveness of memory structures, and compare them with memory-less variants. Our experiments show that the adaptive memory features are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, these methods contrast with memoryless designs that heavily rely on semi-random processes that implement a form of sampling.

2.5.1 Construction methods

A solution S for our problem consists of a set of hubs H and an assignment of each terminal to a hub. Note that with this assignment, the routing of the traffics between any pair of nodes is univocally determined through their respective hubs.

In Section 2.4 we have proposed a constructive method to obtain an initial solution that selects nodes to be hubs in a greedy fashion. Specifically, we have considered an evaluation function to discriminate among candidate nodes based on the costs. The method iteratively selects the hub nodes until there are enough hubs (in terms of capacity) to assign all the nodes in the network. An important characteristic of this method is that, each time a node is selected as a hub, it performs the associated assignment of terminals to this hub in a greedy fashion ignoring future hub selections. In this section, we propose an alternative construction method that performs the assignment step after the hub selection, thus taking into account the complete set of hubs.

Our construction method starts by estimating the number of hubs p that provides enough capacity to assign all the terminals. We basically consider the total traffic between all pairs i, j of nodes, and divide it by the hub capacity Q^h . In mathematical terms:

$$p = \left\lceil \frac{\sum_{i,j \in V} t_{ij}}{Q^h} \right\rceil .$$

Note that traffics t_{ii} can take a positive value in some applications, as in the case of postal deliveries sent to a central hub for sorting.

Since, for example, nodes with large amount of traffics may not be assigned to the same hub, the value of p above can underestimate the required number of hubs to route all the traffic in the network. In particular, we compute the number of nodes for which their traffic exceeds half of the hub capacity:

$$\sum_{j \in V \setminus \{i\}} t_{ij} + \sum_{j \in V \setminus \{i\}} t_{ji} + t_{ii} > \frac{Q^h}{2}.$$

It is likely that two nodes verifying the expression above do not share the same hub since the sum of their traffics may be larger than Q^h. Therefore, if the number of nodes verifying this expression is larger than our estimation of p, we change p to be this number of nodes.

The method we propose here is a multi-start algorithm. Many multi-start methods in combinatorial optimization resort to randomization to perform multiple constructions.

Among them, GRASP methodology [40] is probably one of the most popular. However, we have developed our constructive algorithm under a different paradigm, the adaptive memory programming. Instead of randomization, we apply frequency values, which record past hub appearances to discourage their selection in future constructions.

To decide which hubs to open, we use an evaluation function, described in what follows. Suppose that some nodes have already been selected as hubs. We denote them by H'. In order to select the next hub, the following four elements are considered:

Traffic value For each node i, we compute the traffic through it (incoming, outgoing, and internal traffic) with origin or destination not in H' as

$$t(i) = \sum_{j \in V \setminus \{i\}} t_{ij} + \sum_{j \in V \setminus \{i\}} t_{ji} + t_{ii} - \sum_{j \setminus H'} t_{ij} - \sum_{j \setminus H'} t_{ji}.$$

We then compute the relative value as:

$$\frac{t(i)}{\max_{j \in V \setminus H'} t(j)}.$$

Opening cost We compute the relative fixed installation cost for each node i as:

$$\frac{C_{ii}}{\max_{j \in V \setminus H'} C_{jj}}.$$

Assignment cost For each node i we compute the $m = \frac{n}{p}$ nodes (i.e. the average number of terminals assigned to a hub) with lowest assignment cost to i. The absolute assignment cost of node i, $a(i)$, is defined as the sum of these m costs. The relative assignment cost is computed as:

$$\frac{a(i)}{\max_{j \in V \setminus H'} a(j)}.$$

Frequency value For each node i, we record in $\text{freq}(i)$ the number of times (previous solutions) in which i has been a hub. The relative frequency is:

$$\frac{\text{freq}(i)}{\max_{j \in V \setminus H'} \text{freq}(j)}.$$

These four elements are merged into a single expression reflecting the attractiveness of a node to be selected as a hub. Since we cannot establish a priori their relative importance, we introduce some factors that will be empirically set. For each node i, attractive(i) is defined as:

$$\begin{aligned} \text{attractive}(i) = +\delta \frac{t(i)}{\max_{j \in V \setminus H'} t(j)} - \alpha \frac{C_{ii}}{\max_{j \in V \setminus H'} C_{jj}} \\ -\beta \frac{a(i)}{\max_{j \in V \setminus H'} a(j)} - \gamma \frac{\text{freq}(i)}{\max_{j \in V \setminus H'} \text{freq}(j)}, \end{aligned} \tag{2.11}$$

where $\alpha, \beta, \gamma, \delta \in [0, 1]$, and $\alpha + \beta + \gamma + \delta = 1$. Typically, a node with large traffics is attractive to be selected as a hub, whereas a high opening cost or large assignment costs discourages it. Regarding the frequencies, since we want to diversify the search, those nodes that have been selected as hubs in previous solutions are penalized in posterior constructions. In our computational experiments section, we will test different values of these parameters, which permits to isolate the effect of each of the three elements considered (apart from the frequency) and evaluate their contribution.

At each iteration, our multi-start constructive method selects the best hubs according to the attractive values. Once the hubs are selected, we proceed to assign the terminals to these hubs. Let H be the set of selected hubs. For each terminal i, we compute the best assignment cost $\overline{c}_i$ to the selected hubs as:

$$\overline{c}_i = \min_{h \in H} C_{ih}.$$

Then, we order all the terminals according to the $\overline{c}_i$–values, where the terminal with the minimum value comes first. Following this order, we assign each terminal to its best hub (the one in H with the minimum cost) if it has enough capacity. If this hub does not have enough capacity to route the traffics of terminal i because of previous assignments, we consider the second best hub in H for i according to the assignment cost. We proceed in this way, trying to assign terminal i to the best hub in H that can manage its traffic.

It may happen that none of the hubs in H can accommodate a given terminal. In this case we add this terminal to a list of *orphan nodes.* When the assignment procedure has explored all terminals, the orphan node with largest traffics is selected as a new hub and we assign as many orphan nodes as possible to it. This procedure, which searches among orphan nodes to select a new hub, is repeated until all nodes are assigned to a hub.

The following example illustrates the constructive method described above, that we call CM1. Suppose that we have a network with 15 nodes where we have estimated $p = 2$. We have applied attractive (i) to all nodes and concluded that $H = \{3,\ 10\}$. Column "Order" of Table 2.10 shows the order of the terminals in V for assigning to hubs according to the $\overline{c}_i$–values. The assignment process starts by terminal 4 and assigns it to hub 3, since it is the preferred hub for this terminal. The procedure continues by assigning terminal 1 to hub 10; terminal 7 to hub 3; terminals 2, 15, and 14 to hub 10; and terminal 9 to hub 3. By the time terminal 13 needs to be assigned, hub 10 is full of traffics, so terminal 13 is assigned to hub 3 (second best hub for terminal 13). Then, terminals 8, 11, and 5 are assigned to hub 3. At this point, hubs 3 and 10 are completely full, hence they cannot accommodate any other terminal. Node 6 (next terminal in the order) is then declared an orphan node. The same happens with node 12. Since no more nodes remain to be assigned, the orphan node with largest traffics is selected as a new hub. Suppose that this is the case of node 12. Therefore $H \leftarrow H \cup \{12\}$ and node 12 is assigned to itself. As there is still enough capacity in the new hub 12, terminal 6 is assigned to it. The assignment process finishes here because all terminals have been assigned to a hub.

Table 2.10. Example of ordered nodes and possible hubs in CM1

Order	Terminal	Ordered hubs	
1st	4	3	10
2nd	1	10	3
3rd	7	3	10
4th	2	10	3
5th	15	10	3
6th	14	10	3
7th	9	3	10
8th	13	10	3
9th	8	3	10
10th	11	3	10
11th	5	3	10
12th	6	10	3
13th	12	10	3

Notice that the process implemented in CM1 is designed under the assumption that the best assignment for a terminal is the hub for which its assignment cost is the lowest. However, considering that this is a greedy process and that hubs are limited in terms of their capacity, it is clear that in many cases we cannot assign all the terminals to their preferred hub. For this reason, we also propose alternative construction methods to implement different search strategies. Construction method CM2 orders the terminals in non-increasing order of the $\overline{c}_i$–values (i.e., the terminal with the largest value comes first). The rationale behind this rule is to assign first the terminal with highest minimum assignment cost to the hubs. Constructive method CM3 computes the lowest and the second lowest assignment cost for each terminal i. This is:

$\overline{c}_i = \min_{h \in H} C_{ih}$, $\overline{\overline{c}}_i = \min_{h \in H \setminus \{h^*\}} C_{ih}$, where $h^* = \arg\min_{h \in H} C_{ih}$.

Then CM3 calculates the difference of the two values above, $d_i = \overline{\overline{c}}_i - \overline{c}_i$, to evaluate how "urgent" is to assign i to its best hub. It is clear that if d_i is large, we should try to assign i to its best hub because otherwise the assignment cost will be greater. On the contrary, if d_i is low, the assignment costs of i to its best and second best hubs are very similar. CM3 orders the terminals according to the d_i–values in non-increasing order and assigns them to their best available hub in this order. Once a solution S is obtained with one of the three methods above, we evaluate it.

As it is customary in multi-start methods, once a solution is constructed, we proceed to improve it. The improvement methods used are described in the next section.

2.5.2 Improvement methods

In the previous section, we consider two neighborhoods, N_{pairs} and N_{alone}, to improve a solution by changing the assignments of terminals to hubs. Recall that N_{pairs} implements a classical exchange in which two terminals i and j, assigned to hubs k and l respectively

($k \neq l$), swap their corresponding hubs (i.e., the move assigns i to hub l, and j to hub k). To do this, we proposed a local search method, LS_{pairs}, which implements this neighborhood with a first improvement strategy, i.e. by scanning the list of terminals and applying this exchange every time terminals are assigned to different hubs and the objective function is reduced (while the capacity limits are satisfied). Note that an efficient computation of the objective value after a move requires a detailed study that is explained in Section 2.4.4.

As said before, sometimes the N_{pairs} neighborhood turns out to be too restrictive due to the capacity constraints, so we complemented it with the N_{alone} neighborhood. This second neighborhood performs a simple insertion move in which the assignment of a terminal is changed from a hub to another hub. As in the previous neighborhood, we proposed a local search method, LS_{alone}, based on this move, which implements a first improvement strategy.

The experimental testing in the Section 2.4.6.2 shows that the combination of the two neighborhoods is able to significantly improve the constructed solutions. Here, we want to go a step further by including the possibility of changing the hub selection of a given solution in the neighborhood exploration. As a matter of fact, we believe that further reductions in the objective function can be achieved by permitting the local search method to test different sets of hubs.

However, including or removing a hub in a solution may cause a great change in S, and consequently the evaluation of such a move can be very costly, especially the computation of the number of backbone edges needed after any move. To overcome this difficulty, we propose a new neighborhood, $N_{cluster}$, in which we consider that each hub, together with its assigned terminals, form a set (also known as *cluster*) in the network. This neighborhood explores the change of hub within each cluster. In this way, the hub in a cluster changes its status to become a terminal and one of the terminals in this cluster is now the new hub of the cluster. The rest of the terminals in the cluster remain the same but assigned now to the new hub.

The proposed neighborhood $N_{cluster}$ exhibits a tradeoff between search power and computational cost. On the one hand, it considers changing the hub in a solution, which is a major change that may lead to different types of solutions in the solutions' space. On the other, as it limits the exploration to changes within a cluster there is no need to calculate the number of copies of the backbone edges to compute the value of the new solution.

The associated procedure, $LS_{cluster}$, works as follows. Let U_h be a cluster of nodes formed by a hub h and the set of its assigned terminals S, $U_h = \{h\} \cup S$. We define an evaluation function of the cluster of h, based on the opening cost of h and on the assignment cost of its terminals, as follows:

$$\text{eval}\,(U_h) = C_{hh} + \sum_{j \in S} C_{jh} \ .$$

This evaluation function induces an order in which the set of clusters will be explored in $LS_{cluster}$, where the cluster with the largest evaluation (the one with highest cost) is

explored first, since we try to improve it in the first place. Steps 5 and 6 in Algorithm 2 show respectively the evaluation and ordering of the clusters.

Input: s
1 continue $\leftarrow$ TRUE
2 **while** continue is TRUE **do**
3 continue $\leftarrow$ FALSE
4 Define $\mathcal{U} = \{U_{h \in H}\}$
5 Compute clusters' eval$(U_h) = C_{hh} + \sum_{j \in U_h \setminus \{h\}} C_{jh}, \forall h \in H : |U_h| \geq 2$
6 Order $h \in H : |U_h| \geq 2$ by non-decreasing value of eval(U_h)
7 **foreach** $h \in H : |U_h| \geq 2$ **do**
8 Compute nodes' evaluation eval$(i) = C_{ii} + \sum_{j \in S \setminus \{i\}} C_{ji}, \forall i \in S$
9 Order $i \in S$ by non-decreasing value of eval(i)
10 **foreach** $i \in S$ **do**
11 $\bar{H} = H \setminus \{h\} \cup i$
12 $\bar{S} = S \setminus \{i\} \cup h$
13 $U_i = i \cup \bar{S}$
14 $\bar{\mathcal{U}} = \{U_{j \in \bar{H}}\}$
15 Compute cost solution value using $\bar{\mathcal{U}}$
16 **if** cost of solution using $\bar{\mathcal{U}}$ < cost of solution using $\mathcal{U}$ **then**
17 $H \leftarrow \bar{H}$
18 $i \leftrightarrow h$
19 continue $\leftarrow$ TRUE
Output: s

Algorithm 2: $LS_{cluster}$

Once a cluster is selected in line 7 of Algorithm 2, we explore its terminals for a possible swapping with the current hub in the order given by their evaluation. Given an element i in cluster $U_h = \{h\} \cup S$, its evaluation is given by:

$$\text{eval}\,(i) = C_{ii} + \sum_{j \in S \setminus \{i\}} C_{ji}.$$

We explore the nodes in the cluster and perform the first improvement move. After we have scanned all the clusters and eventually performed one or more moves, LS_{alone} and LS_{pairs} are applied. It is clear that if the hub of a cluster changes, some of the nodes in another cluster could be assigned to the new hub. As mentioned, $N_{cluster}$ does not check this point. Therefore, we consider the N_{alone} and N_{pairs} neighborhoods to check these reassignments and eventually perform further changes after a hub move. The application of LS_{alone} and LS_{pairs} may change a clusters' composition. We perform further steps in which we first go over the clusters with the $N_{cluster}$ neighborhood exploration, and then apply the LS_{alone} and LS_{pairs}, as long as the solution improves. Our local search ends when no further improvement is possible.

2.5.3 Path relinking post-process

Path relinking (PR) was suggested as an approach to integrate intensification and diversification strategies in the context of tabu search [44]. This approach generates new solutions by exploring trajectories that connect high-quality solutions by starting from one of these solutions, called *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by introducing in the initiating solutions attributes contained in the guiding ones. In this way, we generate a sequence of intermediate solutions that "connect" the initiating solution with the guiding one.

The term relinking reflects the fact that this method links again two or more solutions with a path in the search space. In the original tabu search design, two or more good solutions are recorded during the search. Since tabu search describes a trajectory, these solutions can be viewed as linked in the search space by the chain of moves which originated them. After the tabu search execution, we can consider to create a new trajectory, or chain of moves, to go from one of these high-quality solutions to another one. This new trajectory is directed considering the target solutions, instead of the objective function as in the initial application of tabu search. The method is therefore called path relinking because it creates two paths joining two solutions.

Laguna and Martí [58] adapted PR in the context of GRASP as a form of intensification. The relinking, in the context of multi-start algorithms, consists of finding a path between two solutions generated with the constructive method and, eventually, improve the solution in the path with a local search. Therefore, the relinking concept has a different interpretation within GRASP, since the solutions are not originally linked by a sequence of moves. The authors, however, kept the original name of the methodology in spite of the fact that the two solutions are linked for the first time. Resende et al. [82] explored different implementations to hybridize these two methodologies:

Greedy Path Relinking In this method the moves in the path from a solution to another are selected in a greedy fashion, according to the objective function value.

Greedy Randomized Path Relinking Here the method creates a candidate list with the good intermediate solutions and randomly selects among them.

Truncated Path Relinking In this application of PR the path between two solutions is not completed. It is applied, for example, in problems where good solutions are found close to the end points (original solutions) in the path.

Evolutionary Path Relinking This method iterates over the set of high-quality solutions, applying successively the relinking mechanism. It has many similarities with the scatter search methodology [59].

In this chapter we explore a kind of *Greedy Truncated Path Relinking* to our problem as a post-process method.

Let X and Y be two solutions to the CSHLPMLC, and let H_X and H_Y be their associated sets of hubs. The path relinking procedure $\mathrm{PR}(X, Y)$ starts with X, and

gradually transforms it into Y, by swapping out hubs in X with hubs in Y. The hubs in both solutions, H_{XY}, will remain as hubs in all the intermediate solutions generated in the path between them. Let H_{X-Y} be the hubs in X that are not hubs in Y. H_{Y-X} is defined equivalently. Let $\mathrm{PR}_0(X,Y) = X$ be the initiating solution in the path from X to Y. To obtain the first solution $\mathrm{PR}_1(X,Y)$ in this path, we remove a hub $i \in H_{X-Y}$ and replace it with a hub $j \in H_{Y-X}$, thus obtaining

$$H_{\mathrm{PR}_1(X,Y)} = H_{\mathrm{PR}_0(X,Y)} \setminus \{i\} \cup \{j\}\,.$$

In the PR variant implemented here, the selection of nodes i, j is the one minimizing the objective function. In general, to obtain $\mathrm{PR}_{t+1}(X,Y)$ from $\mathrm{PR}_t(X,Y)$, we evaluate the different hubs $i \in H_{\mathrm{PR}_t(X,Y)-Y}$ to be removed and the hubs $j \in H_{\mathrm{Y}-\mathrm{PR}_t(X,Y)}$ to be selected. The move associated with the minimum cost option is performed.

At each intermediate solution in the path from X to Y, a restricted neighborhood is explored to generate the next solution in the path. The neighborhood is restricted because only moves removing hub $i \in H_{\mathrm{PR}_t(X,Y)-Y}$ and selecting hub $j \in H_{\mathrm{Y}-\mathrm{PR}_t(X,Y)}$ are allowed. As the procedure moves from one intermediate solution to the next, the cardinalities of sets $H_{\mathrm{PR}_t(X,Y)-Y}$ and $H_{\mathrm{Y}-\mathrm{PR}_t(X,Y)}$ decrease by one element.

Consequently, as the procedure nears the guiding solution, there are fewer allowed moves to explore. This is why Resende et al. [82] suggested that the search tends to be less effective in the final stages. In truncated path relinking, a new stopping criterion is used. Instead of continuing the search until the guiding solution is reached, only a limited number of steps, PR_{steps}, are allowed, abandoning the path before reaching the final solution. We consider PR_{steps} as a search parameter, and explore the performance of the method with different values in the next section.

In our algorithm we first apply a constructive method (either *CM1*, *CM2* or *CM3*) and the local search methods ($N_{cluster}$, N_{alone}, and N_{pairs}) to populate a set with high-quality solutions (elite set, ES).

Our PR mechanism is designed to work as a post–process method. For each pair of solutions in ES, the cardinality of hubs and the objective values are compared in order to decide which of them will be the starting and the guiding solution. In the case where the cardinalities of their corresponding set of hubs are different, the solution with the highest number of hubs is chosen as starting solution X, and the path relinking is performed against the guiding solution Y. When all hubs in H_{Y-X} have been incorporated, there are one or more hubs that should be removed in order to finally reach Y. The procedure does not do this and stops at this step, so the whole path is not examined completely. This is the reason to consider our procedure a *Truncated* PR. If H_X and H_Y have the same cardinality, the solution with poorest objective value is chosen as the starting point and the path relinking is performed towards the better solution. The assignment of terminals to the hubs of any intermediate solution explored during the path relinking process is performed using the same strategy of the construction process.

2.5.4 Computational experiments

This section describes the computational experiments that we performed to test the effectiveness and efficiency of the procedures discussed above. The metrics that we use to measure the performance of the algorithms are:

1. Value: Average objective value of the best solutions obtained with the algorithm on the instances considered in the experiment.
2. Dev: Average percentage deviation from the best-known solution.
3. Best: Number of instances for which a procedure is able to find the best-known solution.
4. CPU: Average computing time in seconds employed by the algorithm.

We use the same benchmark of instances that has been proposed for the Strategic Oscillation method of Section 2.4 from three well-known data sets CAB, AP, and USA423.

The experimental part is divided into two main blocks. The first block (scientific testing) is devoted to study the performance of the components of the algorithm, as well as to determine the best values for the key search parameters. They have been performed on the same training set of 36 instances used in Section 2.4.6.2. The second block of experiments (competitive testing) has the goal of comparing our procedure with the Strategic Oscillation method in Section 2.4.

2.5.4.1 Scientific testing

From the 170 instances derived from the CAB, AP and USA423 data sets, the preliminary experiments are performed on the following set of 36 instances: 3 instances with $15 \leq n \leq 25$ from the CAB set, 21 instances with $10 \leq n \leq 195$ from the AP set, and 12 instances with $20 \leq n \leq 150$ from the USA423 set. These instances have been classified as small, medium, and large, with 12 instances in each group.

In our first experiment we have compared the combination of the different elements of the attractive(i) function for the hub selection in the constructive method (see Section 2.5.1). Specifically, we have considered the following five alternative sets of parameter values:

Alt1 $= \{\alpha = 0.25; \beta = 0.25; \gamma = 0.25; \delta = 0.25\}$

AllAlpha $= \{\alpha = 1; \beta = 0; \gamma = 0; \delta = 0\}$, only considers the opening costs

AllBeta $= \{\alpha = 0; \beta = 1; \gamma = 0; \delta = 0\}$, only considers the assignment costs

AllGamma $= \{\alpha = 0; \beta = 0; \gamma = 1; \delta = 0\}$, only considers the frequency values

AllDelta $= \{\alpha = 0; \beta = 0; \gamma = 0; \delta = 1\}$, only considers the traffics

Note that Alt1 is the only alternative among the above ones including the four elements in attractive(i). The other alternatives isolate the effect of each element, so they measure their contribution to the complete evaluation.

Table 2.11. Average percentage deviation from the best solution (Dev)

Size	# inst	Alt1	AllAlpha	AllBeta	AllGamma	AllDelta
small	12	6.60%	30.70%	35.40%	31.40%	17.30%
medium	12	0.40%	17.90%	10.50%	16.10%	9.10%
large	12	2.80%	50.60%	29.20%	27.00%	5.90%
summary	36	3.30%	33.10%	25.10%	24.90%	10.80%

Table 2.11 shows the average percentage deviation from the best solution obtained with each alternative method when constructing 100 solutions on each instance. As expected, the best alternative is Alt1, where the four elements are combined. AllDelta (the alternative with $\delta = 1$) is the best among the remaining alternatives, which seems to indicate that the traffic through a node is a very important factor to choose it as a hub. Both alternatives, Alt1 and AllDelta, produce better values for Dev than AllBeta, which was the one used in the strategic oscillation method in Section 2.4, based on the assignment costs of terminals to hubs.

The Friedman statistical test for multiple paired samples obtains a p-value lower than 0.0001, which confirms that there are differences among the five alternatives tested. Additionally, the ranks obtained with this test are Alt1=1.49, AllAlpha=3.61, AllBeta=3.76, AllGamma=3.64, and AllDelta=2.50, which are in line with the above results.

In the second experiment we extend the previous analysis by including other alternative sets of parameters in the constructive method. The new combinations are:

$$\textbf{Alt2} = \{\alpha = 0.70; \beta = 0.10; \gamma = 0.10; \delta = 0.10\}$$

$$\textbf{Alt3} = \{\alpha = 0.10; \beta = 0.70; \gamma = 0.10; \delta = 0.10\}$$

$$\textbf{Alt4} = \{\alpha = 0.10; \beta = 0.10; \gamma = 0.70; \delta = 0.10\}$$

$$\textbf{Alt5} = \{\alpha = 0.10; \beta = 0.10; \gamma = 0.10; \delta = 0.70\}$$

$$\textbf{Alt6} = \{\alpha = 0.35; \beta = 0.20; \gamma = 0.10; \delta = 0.35\}$$

$$\textbf{Alt7} = \{\alpha = 0.40; \beta = 0.10; \gamma = 0.10; \delta = 0.40\}$$

$$\textbf{Alt8} = \{\alpha = 0.40; \beta = 0.15; \gamma = 0.05; \delta = 0.40\}$$

The results obtained using these alternatives are then compared against Alt1, where the four parameters have the same weight. In Alt2, Alt3, Alt4, and Alt5, one of the parameters receives a larger weight value, while the other coefficients get a smaller

weight. For Alt2 the highest weight is given to α, which is associated with the fixed cost of opening a hub. Similarly, Alt3 gives more weight to the cost of assigning terminals to hubs, while Alt4 penalizes the frequency of occurrence in previous iterations, and Alt5 gives more weight to the amount of traffic through the node. The Alt6, Alt7, and Alt8 alternatives try some other combinations of weights, where α and δ have higher values than the other parameters. This is based on the initial findings indicating that the opening costs and the traffics are the two most important elements when selecting hubs. Table 2.12 shows the average percentage deviation from the best solution found in this experiment after constructing 100 solutions on each instance.

Table 2.12. Average percentage deviation from best solution (Dev)

Size	**# inst**	**Alt1**	**Alt2**	**Alt3**	**Alt4**	**Alt5**	**Alt6**	**Alt7**	**Alt8**
small	12	11.30%	7.30%	14.00%	36.70%	10.60%	5.80%	4.80%	4.90%
medium	12	8.30%	1.10%	8.10%	19.80%	9.30%	7.50%	8.80%	8.40%
large	12	4.90%	5.40%	7.30%	24.60%	6.30%	4.60%	9.30%	10.70%
summary	36	8.20%	4.60%	9.80%	27.00%	8.70%	6.00%	7.60%	8.00%

Results in Table 2.12 clearly show that alternative Alt2 builds solutions with least deviation, followed by alternative Alt6, which performs especially well on large instances. We have performed the Friedman test with these alternatives and obtained a p-value < 0.0001, which confirms that there are significant differences between the alternatives. The test returned the following ranges in line with the deviations in Table 2.12: Alt1 = 3.90, Alt2 = 2.54, Alt3 = 4.26, Alt4 = 7.68, Alt5 = 4.53, Alt6 = 3.97, Alt7 = 4.29, and Alt8 = 4.82.

To complement the analysis above, we represent in Figure 2.4 the boxplots of the percentage deviations of each alternative. These diagrams represent the 36 deviation values obtained with each alternative on the instances of the training set. It can be seen that Alt2 has the highest concentration of lower relative deviations, which means that it produces the best solutions.

After analyzing the methods for selecting hubs, we study the methods of assigning terminals to hubs (CM1, CM2, and CM3). Table 2.13 shows the results of this experiment on the training set. For each construction method, the combinations of parameters that have been found to be best in the previous experiment have been tested. In particular, Alt1, Alt2, Alt6, Alt7, and Alt8 are tested.

Table 2.13. Average percentage deviation for constructive methods

	CM1					***CM2***					***CM3***				
	Alt1	***Alt2***	***Alt6***	***Alt7***	***Alt8***	***Alt1***	***Alt2***	***Alt6***	***Alt7***	***Alt8***	***Alt1***	***Alt2***	***Alt6***	***Alt7***	***Alt8***
small	15%	13%	14%	14%	11%	17%	16%	8%	11%	9%	15%	11%	9%	8%	8%
medium	23%	20%	22%	23%	23%	12%	6%	11%	12%	11%	13%	5%	12%	13%	13%
large	59%	39%	40%	44%	41%	10%	6%	10%	11%	14%	8%	9%	8%	13%	14%
summary	32%	24%	25%	27%	25%	13%	9%	9%	11%	12%	12%	8%	10%	11%	12%

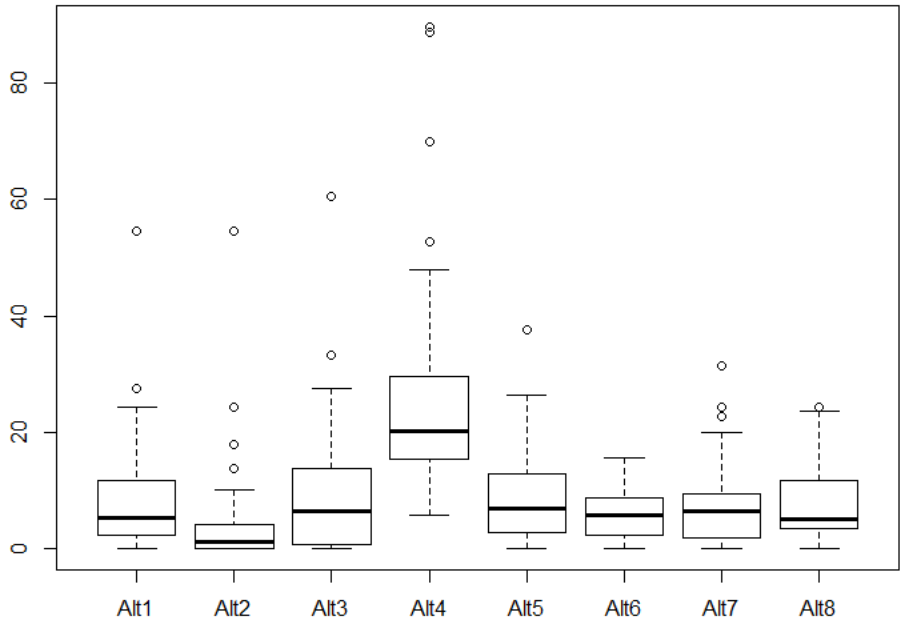

Figure 2.4. Box plot of different alternatives

Table 2.13 show that the lowest average percentage deviation (8%) is obtained with CM3 and parameter combination Alt2. On the other hand, CM2 with Alt2 or Alt6 are close to it with a 9% deviation, while the CM1 strategy gives much poorer results on all parameter combinations. A non-parametric Wilcoxon test performed to compare CM2 and CM3 both using Alt2, gave a p-value of 0.789, which indicates that there is no significant difference between these two configurations. We have chosen CM3 with Alt2 as our configuration for our constructive method and will use it in the following experiments.

In the fourth experiment, we study the contribution of the local search phase of the algorithm applied to the solutions obtained with the constructive method CM3 (and Alt2). In particular, we study five local search variants, as described in Section 2.5.2:

LS_{alone} The method only applies the local search LS_{alone} (Section 2.4.4).

LS_{pairs} The method only applies the local search LS_{pairs} (Section 2.4.4).

$LS_{alone} + LS_{pairs}$ The method combines the two previous local searches.

$LS_{cluster}$ The method only applies the new local search $LS_{cluster}$.

$LS_{Allonce}$ The three methods, $LS_{cluster}$, LS_{alone}, and LS_{pairs} are applied **once** in this order.

It seems natural to ask whether we would be able to improve the solution further after performing the three local searches defining $LS_{Allonce}$. Since LS_{pairs} and LS_{alone} can change the clusters, it might happen that another terminal could be a better hub in the modified clusters. Similarly, when a hub is changed, it could be a better option to assign some terminals to a different hub. We therefore consider a sixth variant, called LS_{All}, in which these methods are repeatedly applied in a loop as shown in Algorithm 3.

```
   Input: s
1  continue ← TRUE
2  while continue is TRUE do
3      continue ← FALSE
4      LS_cluster
5      LS_pairs
6      if s improved after LS_pairs then
7          continue ← TRUE
8      LS_alone
9      if s improved after LS_alone then
10         continue ← TRUE
   Output: s
```

Algorithm 3: LS_{all}

Table 2.14 reports the average percentage reduction from the constructed solution value obtained with these six local search variants by running 50 iterations (construction + local search) on the instances in the training set. It shows that, on average, LS_{alone} improves the solution value by 4.7% with respect to the solutions obtained with the constructive method, while LS_{pairs} improves it by a 5.0% on average.

In line with the results of the strategic oscillation method (Section 2.4), Table 2.14 shows that a further reduction is achieved if both local searches are combined. In particular, $LS_{alone}+LS_{pairs}$ exhibit an improvement of 8.2%. Considering these three previous methods as a basis for comparison, we can observe from this table the good performance of the new local search, $LS_{cluster}$, which obtains better results. Specifically, applying $LS_{cluster}$ only, improves by 11.5% on average the constructed solutions. Another advantage of this search is that it is very fast, since the amount of traffic between the clusters is constant and the number of edges used in the backbone network (W_{kl}) do not change. The $LS_{Allonce}$ column shows the result when applying $LS_{cluster}$ first, then $LS_{alone} + LS_{pairs}$. This process improves the solutions obtained considerably, reaching a reduction of 24% on average on the large size instances, and of 17.8% for all instances in the training set. Finally, The LS_{All} column shows the average results of the improvements after applying the loop procedure described in Algorithm 3. Clearly, it shows that changing the configuration of clusters gives further improvements, thus concluding that

the LS_{All} is the best of the six alternatives. We have applied the Wilcoxon statistical test for two paired samples to $LS_{Allonce}$ and LS_{All}, which returns a p-value lower than 0.0001, confirming the superiority of LS_{All}.

Table 2.14. Local search percentage reduction from construction

Size	LS_{alone}	LS_{pairs}	$LS_{alone} + LS_{pairs}$	$LS_{cluster}$	$LS_{Allonce}$	LS_{All}
small	-2%	-2%	-5%	-9%	-13%	-18%
medium	-5%	-5%	-8%	-12%	-17%	-22%
large	-7%	-8%	-12%	-14%	-24%	-29%
summary	-4.7%	-5.0%	-8.2%	-11.5%	-17.8%	-22.8%

Figure 2.5 shows the search profile of the six methods described above. Specifically, we represent the objective value of the best known solution on a 130-nodes instance when applying different solution methods for 100 global iterations. We have added a seventh method, labeled "Constructions", which represents the best value obtained when applying the constructive method without any local search. The other six lines represent the best solution value obtained with the application of the construction method followed by each of the six local searches described above. Note that in addition to the $LS_{alone} + LS_{pairs}$ combination previously considered, we have also included an eight method, $LS_{pairs} + LS_{alone}$, to study the application of these two methods in reverse order.

The results in this diagram agree with those reported in Table 2.14, where the two methods $LS_{Allonce}$ and LS_{All} present the best results, being LS_{All} the best one. It must be noted that this method is able to obtain the best results from the very beginning of the search process, and continues being the leader in the entire search horizon considered.

In the next experiment, we undertake to assess the performance of the PR post-process. To do this, all the instances in the training set have been executed for 20 iterations to create the solutions in the ES that will be used during PR. The path relinking procedure has been able to improve the previously obtained results in 22 out of the 36 instances in the training set, with an average improvement of 1.4% (denoted as -1.4% in the Summary row of Table 2.15). In this experiment we have also tested the

Table 2.15. Path relinking contribution

Size	**Deviation from best**	**CPU**
small	-1.30%	33.30%
medium	-0.70%	16.40%
large	-2.30%	13.90%
Summary	-1.40%	21.20%

strategy called two-sided path relinking [40], in which the best direction to create the

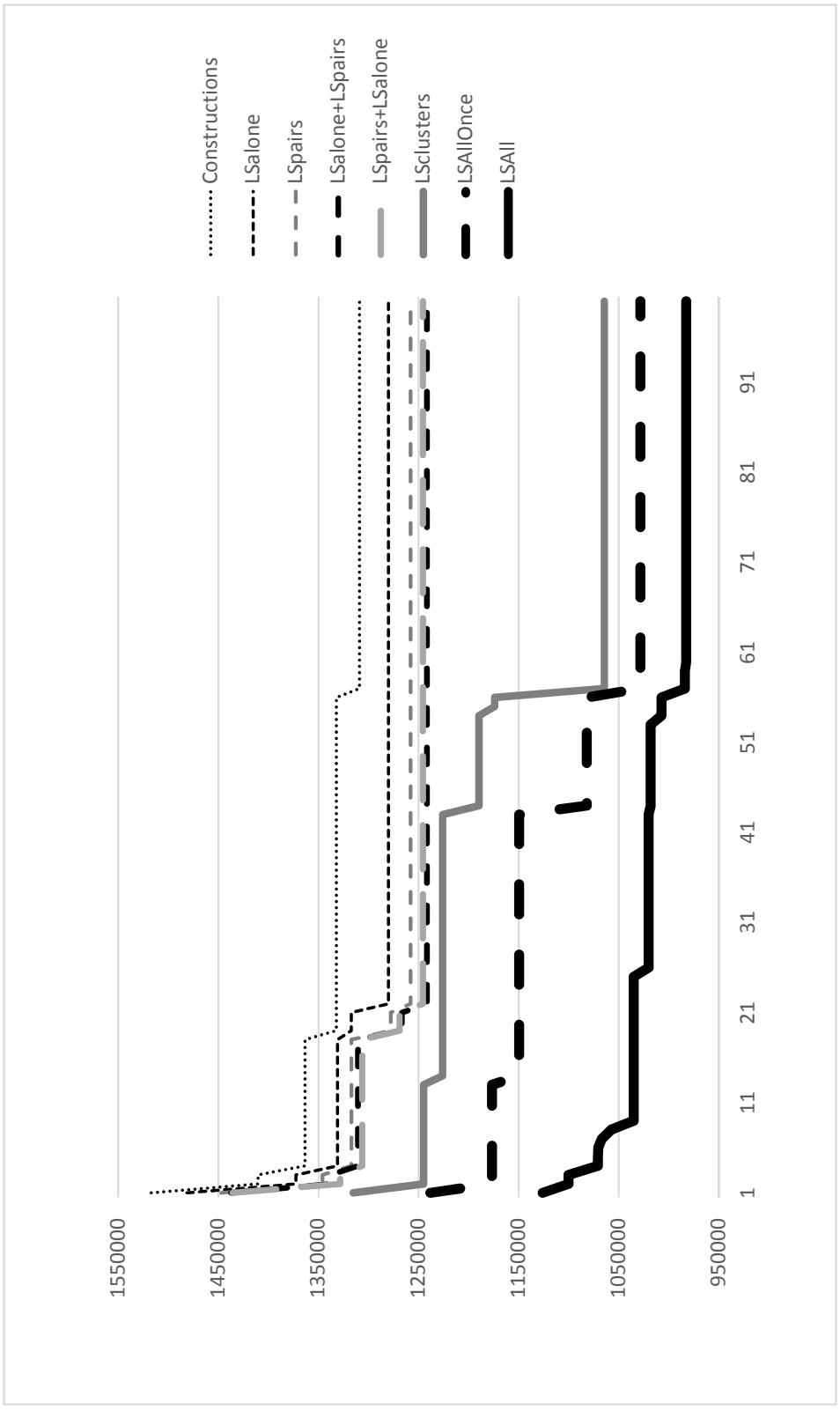

Figure 2.5. Search profile for the different local search methods

path is determined. In particular, we have tested the reversal of the direction considered so far in the PR process, moving from the best towards the poorer solution, and it has not produced any significant difference in the results. In fact, in 26 of the instances, PR gives the same best solution in the path independently of the direction. Table 2.15 shows the results for the different groups of instances, together with the increase in the CPU time for including PR. It can be seen that the increase in the computing time decreases with the size of the instances and, in our opinion, the extra time consumed by PR is worth, especially in the large size instances, where a 2.3% of improvement is obtained.

In our final experiment in the scientific testing, we explore the variant known as Truncated Path Relinking [82]. As described in Section 2.5.3, this strategy considers the early termination of the PR without reaching the guiding solution, thus performing only a limited number of steps, PR_{steps}. Table 2.16 shows the average percentage improvement

Table 2.16. Average percentage reduction in truncating path relinking

	PRsteps					
Size	**1**	**2**	**3**	**4**	**5**	**6**
small	-1.20%	-1.30%	-1.30%	-1.30%	-1.30%	-1.30%
medium	0.00%	-0.30%	-0.40%	-0.60%	-0.70%	-0.70%
large	-0.90%	-1.30%	-1.90%	-2.00%	-2.30%	-2.30%
Summary	-0.70%	-1.00%	-1.20%	-1.30%	-1.40%	-1.40%
CPU increment	2.90%	5.40%	7.00%	8.10%	10.00%	10.60%

(reduction with respect to the initiating solution) achieved at each step of the path. As in previous tables, we divide the results according to the size of the instances and summarize them in an additional row. We include a final row with the average increase in the CPU time, due to the application of PR.

In line with Table 2.15, Table 2.16 clearly shows the overall contribution of PR. Moreover, this table shows that we achieve in two steps the best possible result in the small instances and, therefore, it is a good strategy to truncate the path at an early stage. Medium and large instances require more steps to achieve the best results in the path. At some point (step 5) the improvement stagnates and further steps do not produce better results.

2.5.4.2 Competitive testing

Once we have established the key-search parameters and explored the different variants of our method, we compare two variants of it with the best previous method. The two variants, AMP_{10} and AMP_{20}, correspond to a termination criteria of 10 and 20 global iterations.

Table 2.17 reports the results of the comparison between the proposed procedure and the Strategic Oscillation method, SO. We consider the full set of 170 instances, and report the average percentage deviation (Dev), the running time in seconds (CPU), and

the number of best solutions found with each method (#Best). These instances have been classified as small ($10 \leq n \leq 50$), medium ($55 \leq n \leq 100$), large ($110 \leq n \leq 150$), extra-large ($155 \leq n \leq 200$), and huge ($205 \leq n \leq 250$).

Table 2.17. Comparison with best previous method

		Dev			CPU			# Best		
Size	**# inst**	**SO**	**AMP$_{10}$**	**AMP$_{20}$**	**SO**	**AMP$_{10}$**	**AMP$_{20}$**	**SO**	**AMP$_{10}$**	**AMP$_{20}$**
small	45	1.8%	0.9%	0.4%	3.0	0.3	0.8	13	23	33
medium	36	3.2%	1.1%	0.3%	37.8	11.2	24.8	6	8	30
large	27	10.6%	0.8%	0.0%	310.4	88.9	194.3	0	12	27
extra-large	37	13.2%	1.2%	0.0%	1606.3	185.8	397.8	0	17	37
huge	25	11.6%	1.2%	0.0%	4916.3	679.0	1403.7	0	12	25
Summary	170	7.4%	1.0%	0.2%	1130.7	156.9	329.3	19	72	152

Table 2.17 clearly shows the superiority of the proposed procedure with respect to the previous SO method. Specifically, AMP$_{20}$ is able to obtain 152 best solutions out of the 170 instances considered, and shows an average computing time of 329.3 seconds. This compares favorably with the 19 best solutions obtained with SO, which uses 1130.6 seconds on average. The average deviation (Dev) of 0.2% of AMP$_{20}$ also compares well with the 7.4% of SO. We have performed a Wilcoxon test and the resulting p-value $<$ 0.0001 confirms these conclusions. Regarding the results obtained with AMP$_{10}$, it can be seen that, despite being much better than those produced by SO, they are worse than those provided by AMP$_{20}$, although only half of the computing time is used. Detailed results of each instance are shown in Tables 2.18, 2.19, 2.20, 2.21, and 2.22.

To complement the information in Table 2.17, we have performed a final experiment to compare the evolution of the best solution found by AMP$_{20}$ and SO. Figure 2.6 reports the ratio between the best solution found with both methods at each iteration and the best value reported for the strategic oscillation, where a search horizon of 100 iterations has been considered. This diagram confirms that our proposal consistently outperforms the best previous published method.

2.5.5 Concluding remarks

In this chapter, we have studied a capacitated modular hub location problem that has been modeled in the Literature as a Mixed Integer Non-linear Program, and for whose solution we have proposed a strategic oscillation algorithm and an adaptive memory programming algorithm. We have tested the effects of a variety of search strategies, as those combining different constructive methods and neighborhood structures within a multi-start memory based framework. We have also explored a Path Relinking post-process to obtain improved outcomes. Our experiments show that the both algorithms are capable of searching the solution space economically and effectively, outperforming existing approaches.

Table 2.18. Comparison of SO and AMP on small size instances

	SO			AMP_10			AMP_20		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
10_600_89_60_40_1_60_CAB	237701	1.5%	0.22	234243	0.0%	0.00	234243	0.0%	0.03
10_700_50_60_8_1_60_AP	273801	0.0%	0.23	278212	1.6%	0.00	278212	1.6%	0.03
10_700_50_60_8_1_60_CAB	253255	3.9%	0.22	243854	0.0%	0.00	243854	0.0%	0.03
10_700_69_40_8_1_50_CAB	257691	4.5%	0.22	246610	0.0%	0.00	246610	0.0%	0.03
10_800_60_60_6_1_69_AP	245513	0.0%	0.23	263198	7.2%	0.00	263198	7.2%	0.03
10_800_60_60_6_1_69_CAB	244417	2.6%	0.58	238144	0.0%	0.02	238144	0.0%	0.03
10_800_60_80_8_1_80_CAB	247078	2.6%	0.22	240872	0.0%	0.02	240872	0.0%	0.03
15_500_50_60_40_1_60_CAB	289339	1.1%	0.36	286201	0.0%	0.02	286201	0.0%	0.09
15_600_80_89_6_1_69_CAB	293075	0.8%	0.44	290734	0.0%	0.02	290734	0.0%	0.09
15_600_80_89_8_1_60_CAB	308168	2.5%	0.41	300776	0.0%	0.02	300776	0.0%	0.09
15_700_89_60_40_1_60_CAB	289953	0.7%	0.39	288078	0.0%	0.03	288047	0.0%	0.11
15_800_50_60_40_1_60_CAB	290026	1.3%	0.39	286358	0.0%	0.02	286358	0.0%	0.08
15_900_80_89_40_1_60_CAB	290134	1.3%	0.41	286408	0.0%	0.02	286408	0.0%	0.09
20_700_50_60_8_1_60_AP	406266	3.2%	0.98	393788	0.0%	0.05	393788	0.0%	0.20
20_700_50_60_8_1_60_CAB	176142	0.0%	1.03	176783	0.4%	0.03	176783	0.4%	0.16
20_700_50_60_8_1_60_USA	127058	7.7%	0.92	117986	0.0%	0.05	117986	0.0%	0.17
20_700_69_40_8_1_50_AP	408538	0.0%	1.03	417187	2.1%	0.06	417187	2.1%	0.22
20_700_69_40_8_1_50_CAB	187305	2.3%	0.84	183013	0.0%	0.05	183013	0.0%	0.17
20_800_60_60_6_1_69_CAB	164351	0.0%	0.73	165518	0.7%	0.05	165276	0.6%	0.19
20_800_60_80_8_1_80_AP	363247	0.0%	1.19	363247	0.0%	0.03	363247	0.0%	0.22
20_800_60_80_8_1_80_CAB	170069	1.2%	0.75	167993	0.0%	0.03	167993	0.0%	0.19
20_800_60_80_8_1_80_USA	121071	4.9%	0.87	115370	0.0%	0.03	115370	0.0%	0.17
20_900_80_89_40_1_80_CAB	151342	0.4%	0.80	150767	0.0%	0.03	150767	0.0%	0.16
25_600_80_60_6_1_40_CAB	210578	0.0%	1.70	210808	0.1%	0.06	210808	0.1%	0.36
25_600_80_89_6_1_69_CAB	192213	0.0%	1.39	192388	0.1%	0.06	192388	0.1%	0.36
25_600_80_89_8_1_60_CAB	214944	3.9%	1.64	206901	0.0%	0.06	206814	0.0%	0.39
25_650_69_69_6_1_50_CAB	210278	4.0%	1.84	202169	0.0%	0.08	202169	0.0%	0.37
25_650_69_69_6_1_70_CAB	162862	0.7%	2.14	165762	2.5%	0.09	161771	0.0%	0.36
25_800_89_60_40_1_80_CAB	179893	0.0%	1.86	180375	0.3%	0.08	180210	0.2%	0.34
25_900_80_89_40_1_80_CAB	181207	0.7%	1.61	180028	0.0%	0.08	180028	0.0%	0.34
30_600_80_89_8_1_60_AP	383188	8.5%	3.46	353148	0.0%	0.12	353148	0.0%	0.53
30_700_69_40_8_1_50_USA	211640	0.0%	2.93	214726	1.5%	0.25	214726	1.5%	0.83
35_600_80_89_8_1_60_AP	448064	2.5%	3.98	437119	0.0%	0.28	437119	0.0%	0.89
35_600_80_89_8_1_60_USA	206472	0.0%	4.40	207966	0.7%	0.30	207966	0.7%	0.64
35_700_80_50_8_1_69_AP	436550	0.2%	4.23	435489	0.0%	0.27	435489	0.0%	0.58
40_600_80_89_8_1_60_USA	288503	2.7%	6.19	306713	9.1%	1.14	281012	0.0%	2.86
40_700_80_50_8_1_69_AP	478470	0.0%	6.43	488152	2.0%	0.59	486523	1.7%	1.20
40_700_80_50_8_1_69_USA	282629	2.8%	6.01	277734	1.0%	1.03	275037	0.0%	2.07
45_600_80_89_8_1_60_AP	521958	1.5%	8.44	528549	2.7%	0.78	514433	0.0%	1.84
45_700_69_40_8_1_50_AP	589832	0.6%	8.63	592707	1.1%	0.89	586337	0.0%	1.97
45_700_69_40_8_1_50_USA	345335	1.5%	6.83	340367	0.0%	1.42	340367	0.0%	3.29
50_600_80_89_8_1_60_USA	347675	5.0%	12.34	337159	1.8%	2.39	331097	0.0%	4.99
50_700_69_40_8_1_50_AP	598636	0.0%	11.50	610081	1.9%	1.48	610081	1.9%	3.23
50_700_80_50_8_1_69_AP	562786	2.4%	11.23	552563	0.5%	1.42	549699	0.0%	2.85
50_700_80_50_8_1_69_USA	328853	0.5%	10.87	337165	3.0%	2.32	327249	0.0%	5.05

Table 2.19. Comparison of SO and AMP on medium size instances

	SO			AMP_10			AMP_20		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
55_500_60_69_60_1_50_AP	551996	4.9%	12.28	526290	0.0%	1.70	526290	0.0%	3.71
55_500_60_69_60_1_50_USA	356419	0.1%	11.51	355920	0.0%	3.28	355920	0.0%	7.39
55_800_69_50_80_1_60_AP	627685	1.4%	13.65	622691	0.6%	1.59	618881	0.0%	3.62
55_800_69_50_80_1_60_USA	356039	0.0%	10.31	367214	3.1%	3.45	367214	3.1%	7.36
60_500_60_69_60_1_50_AP	597551	10.4%	14.23	570095	5.3%	3.31	541254	0.0%	6.75
60_600_60_69_60_1_69_USA	324245	0.0%	14.34	339436	4.7%	4.71	339436	4.7%	9.38
60_800_69_50_80_1_60_AP	664374	0.0%	13.32	671543	1.1%	3.43	669005	0.7%	7.44
60_800_69_50_80_1_60_USA	375981	2.2%	14.96	375481	2.0%	4.56	368020	0.0%	10.25
65_500_60_69_60_1_50_AP	594968	0.5%	19.49	597554	0.9%	3.54	592242	0.0%	8.49
65_600_60_69_60_1_69_AP	596768	4.0%	19.06	574420	0.1%	3.78	573660	0.0%	8.07
65_600_60_69_60_1_69_USA	366133	6.1%	21.09	346628	0.5%	5.90	344968	0.0%	12.45
65_800_69_50_80_1_60_USA	405756	9.6%	21.08	370350	0.0%	7.47	370350	0.0%	14.24
70_500_60_69_60_1_50_AP	664745	4.7%	24.56	634622	0.0%	5.43	634622	0.0%	11.68
70_600_60_69_60_1_69_AP	615656	0.5%	22.25	612709	0.0%	4.60	612709	0.0%	11.17
70_600_60_69_60_1_69_USA	383492	1.8%	29.03	380610	1.0%	7.24	376830	0.0%	17.66
70_800_69_50_80_1_60_AP	769108	2.7%	27.27	754050	0.7%	4.88	748826	0.0%	10.47
75_500_60_69_60_1_50_USA	552080	0.0%	29.14	567428	2.8%	12.40	564299	2.2%	28.97
75_600_60_69_60_1_69_AP	665111	1.9%	35.23	656614	0.6%	6.13	652910	0.0%	13.74
75_600_60_69_60_1_69_USA	519474	3.9%	31.44	516299	3.3%	10.83	499866	0.0%	26.01
75_800_69_50_80_1_60_AP	828245	6.8%	32.87	784188	1.1%	6.65	775560	0.0%	14.40
80_500_60_69_60_1_50_AP	722632	2.5%	40.25	712368	1.1%	6.27	704828	0.0%	14.87
80_500_60_69_60_1_50_USA	632672	0.0%	35.09	641698	1.4%	16.58	638106	0.9%	33.81
80_800_69_50_80_1_60_AP	837210	1.4%	39.53	825392	0.0%	6.69	825392	0.0%	16.04
80_800_69_50_80_1_60_USA	673561	4.5%	36.57	644728	0.0%	14.90	644728	0.0%	34.76
85_500_60_69_60_1_50_AP	762920	1.7%	62.71	759237	1.2%	9.42	750455	0.0%	21.76
85_500_60_69_60_1_50_USA	777825	7.1%	53.17	738851	1.8%	20.47	725993	0.0%	48.85
85_800_69_50_80_1_60_AP	903683	4.2%	60.42	880568	1.5%	9.36	867232	0.0%	22.67
90_500_60_69_60_1_50_USA	804494	7.0%	53.98	752493	0.1%	32.56	751816	0.0%	68.49
90_600_60_69_60_1_69_AP	759377	0.3%	70.56	756916	0.0%	12.17	756916	0.0%	24.57
90_600_60_69_60_1_69_USA	708086	2.0%	54.34	694659	0.0%	26.97	694320	0.0%	54.07
90_800_69_50_80_1_60_AP	966669	4.6%	70.72	931584	0.8%	12.43	924069	0.0%	25.85
95_500_60_69_60_1_50_AP	905808	6.8%	75.52	848766	0.1%	15.60	848132	0.0%	30.09
95_500_60_69_60_1_50_USA	780646	4.1%	64.88	758687	1.2%	32.22	749775	0.0%	70.33
95_600_60_69_60_1_69_AP	826451	2.0%	82.84	814028	0.5%	13.04	810103	0.0%	28.94
95_600_60_69_60_1_69_USA	714260	3.7%	65.63	692781	0.6%	30.65	688970	0.0%	72.09
100_500_60_69_60_1_50_USA	792405	0.0%	77.82	794332	0.2%	40.48	794332	0.2%	91.62

Table 2.20. Comparison of SO and AMP on large size instances

	SO			**AMP_10**			**AMP_20**		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
110_500_60_69_60_1_50_AP	996975	3.1%	150.65	970622	0.4%	26.99	967134	0.0%	60.89
110_600_60_69_60_1_69_AP	934826	2.3%	142.15	915806	0.2%	24.99	914181	0.0%	52.43
110_700_80_60_89_1_60_USA	1122255	13.6%	105.91	988074	0.0%	58.69	988074	0.0%	133.62
110_800_69_50_80_1_60_USA	1039753	13.1%	117.52	943367	2.6%	62.79	919493	0.0%	143.06
110_900_69_50_89_1_60_USA	1105052	8.6%	105.43	1025893	0.8%	52.98	1017557	0.0%	136.96
120_500_60_69_60_1_50_AP	1123004	8.0%	206.13	1053603	1.3%	43.59	1039932	0.0%	81.98
120_500_60_69_60_1_50_USA	1117652	14.3%	151.31	977587	0.0%	87.44	977587	0.0%	199.98
120_700_80_60_89_1_60_USA	1219108	11.5%	163.57	1093189	0.0%	74.66	1093189	0.0%	189.00
120_900_69_50_89_1_60_USA	1175745	8.8%	146.52	1103461	2.1%	83.34	1080591	0.0%	204.63
125_500_60_69_60_1_50_AP	1145428	4.7%	192.06	1093738	0.0%	66.02	1093577	0.0%	131.42
125_800_69_50_80_1_60_AP	1371476	9.9%	208.84	1265452	1.4%	45.52	1247724	0.0%	105.47
130_600_60_69_60_1_69_AP	1158759	8.5%	291.91	1067512	0.0%	67.16	1067512	0.0%	144.30
130_600_60_69_60_1_69_USA	1040651	6.3%	180.23	984860	0.6%	107.21	978893	0.0%	254.72
130_800_69_50_80_1_60_USA	1264063	14.2%	227.08	1106959	0.0%	117.71	1106959	0.0%	253.02
135_600_60_69_60_1_69_AP	1229722	9.9%	335.92	1118752	0.0%	66.55	1118752	0.0%	145.82
135_800_69_50_80_1_60_USA	1405752	24.8%	251.49	1126503	0.0%	132.34	*1126503*	0.0%	*296.23*
140_500_60_69_60_1_50_AP	1412597	11.1%	357.75	1271185	0.0%	71.54	1271185	0.0%	162.32
140_700_80_60_89_1_60_USA	1490007	21.6%	286.76	1225642	0.0%	165.41	*1225642*	0.0%	*296.92*
140_800_69_50_80_1_60_AP	1570932	8.9%	377.44	1442113	0.0%	83.24	1442113	0.0%	184.51
140_900_69_50_89_1_60_USA	1444192	16.2%	275.01	1242773	0.0%	136.96	*1242773*	0.0%	*283.20*
145_600_80_69_60_1_50_AP	1462902	7.8%	352.66	1368251	0.9%	90.17	1356427	0.0%	217.72
145_600_80_69_60_1_50_USA	596573	16.2%	710.75	519791	1.2%	112.31	513607	0.0%	239.69
145_800_69_50_80_1_60_AP	1603039	7.4%	357.12	1517218	1.6%	106.58	1493074	0.0%	230.67
145_800_69_50_80_1_60_USA	654681	8.4%	647.73	628893	4.1%	129.81	604117	0.0%	259.14
150_1000_69_60_80_1_69_USA	612031	5.8%	837.07	578217	0.0%	135.62	578217	0.0%	270.54
150_800_69_50_80_1_60_AP	1734477	11.1%	424.82	1584133	1.5%	114.37	1561333	0.0%	274.49
150_900_69_60_80_1_89_USA	606912	9.7%	775.96	562431	1.7%	136.29	553060	0.0%	293.43

Table 2.21. Comparison of SO and AMP on extra-large size instances

	SO			AMP_10			AMP_20		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
155_1000_69_60_80_1_69_USA	650419	13.9%	919.46	571084	0.0%	149.53	571084	0.0%	304.79
155_500_60_69_60_1_50_AP	1427231	16.5%	635.50	1240808	1.2%	124.74	1225509	0.0%	279.15
155_800_69_50_80_1_60_AP	1602365	13.6%	604.55	1442949	2.3%	127.36	1410360	0.0%	275.13
160_600_60_69_60_1_69_AP	685604	13.5%	950.80	613597	1.5%	58.83	604311	0.0%	126.61
160_700_80_60_89_1_60_USA	704872	24.3%	1010.66	588205	3.8%	194.77	566847	0.0%	369.32
160_800_69_50_80_1_60_AP	826772	9.0%	1042.87	786230	3.6%	82.48	758780	0.0%	176.89
160_900_80_50_60_1_69_AP	775178	7.2%	1008.39	741112	2.5%	79.78	722878	0.0%	159.69
160_900_89_50_60_1_69_USA	530376	12.4%	1048.41	481774	2.1%	163.12	471754	0.0%	339.01
165_1000_69_60_80_1_69_USA	665409	9.4%	1302.26	625185	2.8%	172.64	608228	0.0%	369.96
165_800_69_50_80_1_60_AP	876244	14.7%	1109.94	764063	0.0%	114.51	764063	0.0%	227.33
165_800_69_50_80_1_60_USA	704389	6.1%	1302.59	667884	0.6%	244.26	663785	0.0%	466.76
170_500_60_69_60_1_50_AP	714822	21.0%	1152.12	590908	0.0%	168.00	590908	0.0%	293.91
170_600_89_60_69_1_80_USA	551600	15.5%	1304.46	491632	2.9%	260.48	477564	0.0%	485.13
170_700_80_60_89_1_60_USA	609581	6.7%	1361.95	605758	6.0%	240.73	571293	0.0%	471.38
170_900_69_60_80_1_89_USA	613150	4.8%	1403.26	589807	0.8%	196.72	585232	0.0%	474.49
170_900_80_50_60_1_69_AP	754262	2.4%	1251.03	742642	0.8%	116.22	736391	0.0%	266.60
175_500_60_69_60_1_50_AP	649148	8.4%	1427.86	598789	0.0%	100.08	598789	0.0%	206.36
175_600_60_69_60_1_69_AP	648587	5.5%	1664.50	615051	0.0%	134.62	615051	0.0%	291.06
175_800_69_50_80_1_60_USA	716778	6.3%	1625.83	725441	7.6%	261.61	674337	0.0%	559.54
175_900_69_60_80_1_89_USA	602257	0.1%	1434.86	601938	0.0%	201.47	601938	0.0%	447.89
180_1000_69_60_80_1_69_USA	740453	9.8%	1970.72	674578	0.0%	238.30	674578	0.0%	480.79
180_600_60_69_60_1_69_AP	746918	19.5%	1804.62	625016	0.0%	168.64	625016	0.0%	377.00
180_600_89_60_69_1_80_USA	558479	11.8%	1739.88	501098	0.4%	284.38	499331	0.0%	599.79
180_800_89_69_89_1_89_AP	797621	8.8%	1921.67	738870	0.8%	151.14	733082	0.0%	334.86
185_500_60_69_60_1_50_AP	858506	36.3%	1872.10	629924	0.0%	113.62	629924	0.0%	336.56
185_600_80_89_89_1_89_AP	700282	5.5%	2029.67	663550	0.0%	192.09	663550	0.0%	424.61
185_600_89_60_69_1_80_USA	530783	5.9%	1880.00	501049	0.0%	272.53	501049	0.0%	656.61
185_800_69_50_80_1_60_AP	890070	9.5%	1971.64	813219	0.0%	147.86	813219	0.0%	334.44
185_900_69_60_80_1_89_USA	621650	7.4%	2158.96	579041	0.0%	264.73	579041	0.0%	511.57
190_600_60_69_60_1_69_AP	773840	20.5%	2200.58	642125	0.0%	163.49	642125	0.0%	354.57
190_600_80_89_89_1_89_AP	800719	20.5%	2288.98	667778	0.5%	216.77	664307	0.0%	426.08
190_600_89_60_69_1_80_USA	522508	5.4%	2190.87	506162	2.1%	386.28	495751	0.0%	818.66
190_700_89_69_89_1_89_USA	575933	10.9%	2177.05	531110	2.2%	355.05	519531	0.0%	720.86
190_800_69_50_80_1_60_AP	1031614	27.1%	2298.88	811776	0.0%	166.27	811776	0.0%	379.62
195_600_60_69_60_1_69_AP	774165	18.8%	2487.11	651750	0.0%	170.76	651750	0.0%	401.27
195_800_69_50_80_1_60_AP	1107739	34.2%	2431.53	825666	0.0%	249.33	825402	0.0%	500.46
195_900_89_89_89_1_69_AP	1093874	26.2%	2449.22	866814	0.0%	142.29	866814	0.0%	470.57

Table 2.22. Comparison of SO and AMP on huge size instances

	SO			AMP_10			AMP_20		
Instance	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
200_500_60_69_60_1_50_AP	661633	5.0%	2686.72	630174	0.0%	205.88	630174	0.0%	456.88
200_700_80_60_89_1_60_USA	743444	17.5%	2976.42	632652	0.0%	422.74	632652	0.0%	912.87
200_700_89_69_89_1_89_USA	608272	10.0%	2985.89	558004	0.9%	444.05	552794	0.0%	1046.31
200_800_69_50_80_1_60_AP	763358	5.0%	2731.83	726953	0.0%	202.12	726953	0.0%	450.04
200_800_89_69_89_1_89_USA	645449	11.9%	2995.71	577027	0.0%	429.62	577027	0.0%	860.19
205_800_69_50_80_1_60_USA	800686	10.7%	3086.01	723185	0.0%	494.43	723185	0.0%	1002.65
205_900_69_60_80_1_89_USA	670343	6.2%	3570.73	641338	1.6%	477.98	631462	0.0%	881.68
210_800_69_50_80_1_60_USA	811601	9.9%	3832.57	761961	3.2%	474.52	738677	0.0%	1027.60
210_900_69_60_80_1_89_USA	699431	8.9%	3743.74	642226	0.0%	455.71	642226	0.0%	994.66
215_800_69_50_80_1_60_USA	897348	18.7%	3927.14	776169	2.7%	508.17	756041	0.0%	1067.44
215_900_69_60_80_1_89_USA	736413	11.8%	3653.79	662574	0.6%	440.71	658439	0.0%	952.68
220_800_69_50_80_1_60_USA	872528	11.9%	4275.09	785013	0.7%	669.57	779880	0.0%	1414.82
220_900_69_60_80_1_89_USA	725843	8.8%	4485.24	667050	0.0%	661.34	667050	0.0%	1204.99
225_800_69_50_80_1_60_USA	978703	7.4%	4993.80	911123	0.0%	689.23	911123	0.0%	1399.77
225_900_69_60_80_1_89_USA	812905	9.6%	4668.05	752181	1.4%	711.43	741512	0.0%	1366.89
230_800_69_50_80_1_60_USA	994501	11.5%	6096.23	892018	0.0%	694.07	892018	0.0%	1562.39
230_900_69_60_80_1_89_USA	859790	14.3%	5934.23	765328	1.8%	742.37	751951	0.0%	1509.29
235_800_69_50_80_1_60_USA	1067897	7.5%	6223.85	1026170	3.3%	800.88	993119	0.0%	1734.28
235_900_69_60_80_1_89_USA	967918	20.0%	6017.28	835859	3.7%	768.59	806302	0.0%	1662.56
240_800_69_50_80_1_60_USA	1106592	12.2%	6079.38	1019676	3.4%	1122.78	985860	0.0%	2515.60
240_900_69_60_80_1_89_USA	914468	12.2%	6852.66	846005	3.8%	788.71	815008	0.0%	1769.56
245_800_69_50_80_1_60_USA	1209802	20.8%	7913.75	1001725	0.0%	1264.32	*1001725*	0.0%	*2322.69*
245_900_69_60_80_1_89_USA	913041	8.5%	7197.28	841321	0.0%	903.16	841321	0.0%	1799.31
250_800_69_50_80_1_60_USA	1132659	11.7%	7800.79	1045446	3.1%	1299.77	1013736	0.0%	2508.24
250_900_69_60_80_1_89_USA	991167	17.2%	8180.05	845767	0.0%	1302.52	845767	0.0%	2668.66

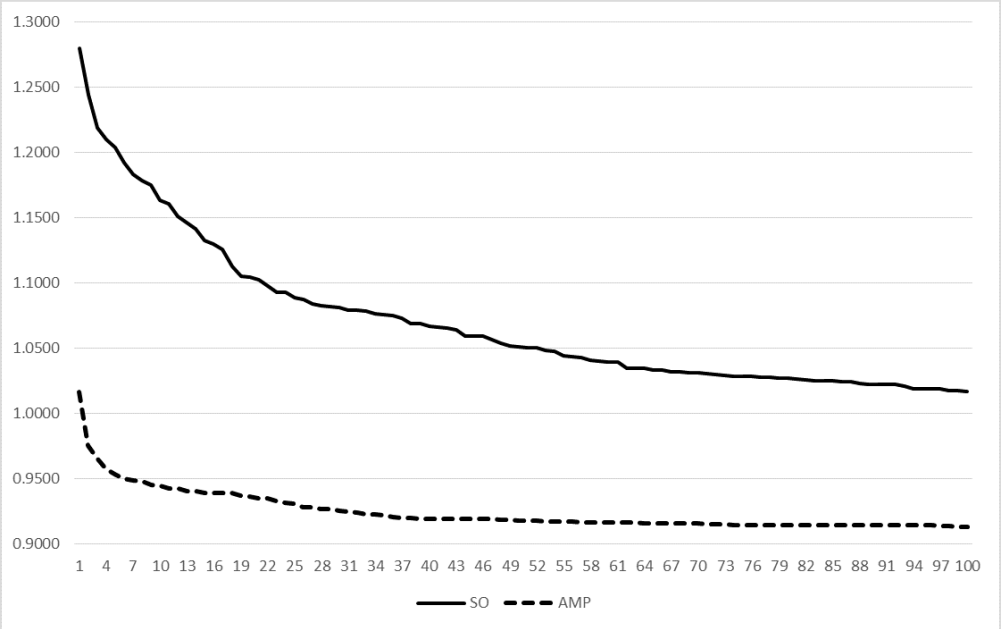

Figure 2.6. Search profile of best methods

Chapter 3

Solution methods for the uncapacitated r-allocation p-hub median problem

Summary

In this chapter we propose two heuristic algorithms to solve the uncapacitated r-allocation p-hub median problem. In the classical p-hub location problem, given a set of nodes with pairwise traffic demands, p of them must be selected as hub locations to route all traffics through them at a minimum cost. An extension of this problem, called the r-allocation p-hub median problem, was recently proposed by Yaman [103], in which every node is assigned to r of the p selected hubs, $r \leq p$, allowing to route the traffic of the nodes through their associated r hubs.

In the algorithm that we propose to solve this problem, the constructive methods have three phases: location, assignment, and routing. Specifically, we first propose a heuristic algorithm based on the GRASP methodology in which we consider three local search procedures. Sometimes, the combinatorial nature of this problem makes them time-consuming. We therefore propose a filtering mechanism to discard low-quality constructions and skip its improvement, saving its associated running time. We perform several experiments to first determine the values of the key-search parameters of our method and then to compare with previous algorithms.

The second algorithm that we propose is based on the Scatter Search methodology. Scatter search is a population-based method that has been shown to yield high-quality outcomes for combinatorial optimization problems. The methodology uses strategies for combining solution vectors that have proved effective in a variety of problem settings. In particular, we propose mechanisms to generate, combine, and improve solutions for this problem. Special mention deserves the use of path-relinking as an extension of the classical combination method.

Computational results on 465 instances show that while only small instances can be optimally solved with exact methods, the heuristics that we propose are able to find high-

quality solutions on larger instances in short computing times. Moreover, when targeting the classical p-hub versions (with $r = 1$ and $r = p$), our heuristics are competitive with the state–of–the–art methods.

3.1 Introduction

The p-hub median problem is a classical optimization problem [76] in which, given a set of nodes with pairwise traffic demands, we have to choose p of them as hub locations and route all the traffic through these hubs at a minimum cost. For each pair of nodes i and j, there is a traffic t_{ij} that needs to be transported. It is generally assumed that direct transportation between non-hub nodes is not possible, and the t_{ij} traffic travels on a path $i \to k \to l \to j$, where i and j are assigned to hubs k and l, respectively.

There are two extensively studied versions of the p-hub location problem regarding the allocation strategy: the single allocation and the multiple allocation versions. In the single allocation p-hub median problem, each node is assigned to one of the p hubs, only allowing to send and receive traffic through this single hub. In the multiple allocation p-hub median problem, each node can send and receive traffic through any of the p hubs.

Transporting the t_{ij} units flow through the path $i \to k \to l \to j$ has an associated cost $c_{ij}(k, l)$, usually computed as $c_{ij} = t_{ij}\,(\chi d_{ik} + \alpha d_{kl} + \delta d_{lj})$, where d_{ik} is the distance between i and k (similarly for the d_{kl} and d_{lj}), and χ, α and δ are unit rates for collection (origin-hub), transfer (hub-hub) and distribution (hub-destination), respectively. Generally, α is used as a discount factor to provide reduced unit costs on arcs between hubs, so typically $\alpha < \chi$ and $\alpha < \delta$. Therefore, a solution is determined by a set of hubs, the node-to-hubs assignments, and the travel paths for each pair of nodes. The sum of the $c_{ij}(k, l)$ values for all (i, j) pairs is the solution cost or value. The problem then consists of finding the hubs, assignments, and paths that minimize the total cost of transportation.

During the last years p-hub location problems have been widely studied due to the increase in the number of applications in telecommunications, transportation and logistics. The p-hub network, based on transshipment nodes, provides a better utilization of transporters. Different versions of the problem include, single and multiple hub allocations (as mentioned above), capacity constraints, fixed costs, and maximum travel time, to mention the most common ones. The p-hub median problem belongs to the class of $\mathcal{NP}$-hard problems. Even when the set of hubs is given, the sub-problem of optimal allocation of non-hub nodes to hubs is also $\mathcal{NP}$-hard [62]. We refer the reader to the excellent surveys on this topics by Alumur and Kara [6], and Campbell and O'Kelly [20].

A new evolutionary approach was presented by Milanovic [71] for solving the multiple allocation version of the problem. Integer encoding of individuals was used to ensure their feasibility, whose quality was evaluated using a fitness function. By applying genetic operators of selection, crossover, and mutation, future generations were produced. Duplicated individuals were removed from the population in the next generation, being also limited to a certain percentage the individuals with the same objective value but different genetic code. Fine grained tournament selection ($FGTS$) was used, as well as

standard one-point crossover operator and the idea of *frozen bits* to increase the diversity of the genetic material. The results demonstrate the usefulness of the proposed approach with new best solutions for three standard instances.

Also, a VNS approach was presented by Ilic et al. [50] for the single allocation version of the problem. Three neighborhoods were proposed for the VNS scheme, using the idea of hubs as clusters: *allocate* tries to change the allocations of every non-hub node, leaving all other elements unchanged; *alternate* preserves all clusters, changing the location of a hub from one node to other from the same cluster, assigning the remaining nodes of the cluster to this new hub; *new locate* tries to increase the diversity of solutions obtained selecting nodes from out of a cluster to be hub and then assigning the nodes at the cluster to other hubs. The authors also presented how to efficiently update data structures for calculating new total flow and cost in the network. Both sequential and nested strategies of the VNS were proposed, outperforming the best-known heuristic in terms of effort and quality solutions for the single allocation version.

Recently, Yaman [103] proposed a very interesting variant of this problem in which each node can be connected to at most r of the p hubs, called the uncapacitated r-allocation p-hub median problem (UrApHMP). The motivation of this variant comes from the fact that the single allocation version, in which a node is assigned to a single hub is too restricted for real-world situations, while the multiple allocation variant, where each node can use any of the p hubs to route its traffic, results in high fixed costs and complicated networks. The r-allocation p-hub median problem, being $r \leq p$, generalizes both versions of the p-hub median problem. When $r = 1$ we are at the single allocation version, whereas if $r = p$, we have the multiple allocation version. Yaman proposed in [103] a mixed integer programming formulation for this generalized version and performed a computational study to first compare the r-allocation version with the single and multiple variants, and then to optimally solve small and medium size instances. She observed in instances with 50 and 75 nodes, and 3, 4, and 5 hubs, that when a node is allowed to be allocated to two hubs, the solutions are significantly cheaper than the single allocation solutions (about 2.0% on average) and slightly more expensive than the multiple allocation version (about 0.3% on average).

3.2 A mixed integer linear programming formulation

The single allocation version of the p-hub median problem was formulated for the first time by O'Kelly [76] as a quadratic integer program. This formulation resulted in a very difficult problem to be solved. Campbell [18] formulated the p-hub median problem as an integer program, but this formulation contained many variables and constraints ($\mathcal{O}(n^4)$).

In [103] three different formulations for the UrApHMP are proposed. As with the single and multiple allocation p-hub median problems, one of the formulations (the first one presented in this paper) uses variables with up to four indexes but is the tighter formulation. The other two use aggregate variables with at most three indexes. Although these two other formulations have fewer variables, the bounds they produce are not as

strong as those obtained with the first formulation. Here, the first one of the three formulations, which will be the one that will be used in the computational results, is described because, as it is stated in Yaman [103], it is the strongest.

Given a network $G = (V, A)$ with a set of nodes V and a set of arcs A, let t_{ij} be the amount of traffic to be routed from node i to node j, i.e., through the arc (i, j), and let d_{ij} be its associated unit routing cost. The r-allocation p-hub median problem is then formulated [103] in terms of the following variables:

- Given a node $k \in V$, $z_{kk} = 1$ if the node is a hub (i.e., if a hub is set or located at this node), and $z_{kk} = 0$ otherwise.
- Given a non-hub node $i \in V$ and a hub $k \in V$, $z_{ik} = 1$ if node i is assigned or allocated to node k, and 0 otherwise.
- Finally, x_{ijkl} is the proportion of the traffic t_{ij} from node $i \in V$ to node $j \in V$ that travels along the path $i \to k \to l \to j$, where k and l are used as hubs.

With these variables, the problem can be formulated as follows:

$$\min \quad \sum_{i\in V}\sum_{j\in V}\sum_{k\in V}\sum_{l\in V} t_{ij}(\chi d_{ik} + \alpha d_{kl} + \delta d_{lj})x_{ijkl} \tag{3.1}$$

Subject to:

$$\sum_{k\in V} z_{ik} \leq r, \qquad \forall i \in V \tag{3.2}$$

$$z_{ik} \leq z_{kk}, \qquad \forall i, k \in V \tag{3.3}$$

$$\sum_{k\in V} z_{kk} = p, \tag{3.4}$$

$$\sum_{k\in V}\sum_{l\in V} x_{ijkl} = 1, \qquad \forall i, j \in V \tag{3.5}$$

$$\sum_{l\in V} x_{ijkl} \leq z_{ik}, \qquad \forall i, j, k \in V \tag{3.6}$$

$$\sum_{k\in V} x_{ijkl} \leq z_{jl}, \qquad \forall i, j, l \in V \tag{3.7}$$

$$x_{ijkl} \geq 0, \qquad \forall i, j, k, l \in V \tag{3.8}$$

$$z_{ik} \in \{0,1\}, \qquad \forall i,k \in V. \tag{3.9}$$

Constraints 3.2 ensure that each node is allocated to at most r hubs, where nodes are assigned to hubs according to 3.3. In addition, constraint 3.4 limits to p the number of hubs. Finally, constraints 3.5 to 3.7 are associated with the routing of the traffic between each pair of nodes i, j through their corresponding hubs k, l.

In our computational experiments, we have tested this formulation and studied the effectiveness of our heuristics in terms of their ability to find the optimal solution on small size instances.

3.3 A GRASP algorithm

The GRASP (Greedy Randomized Adaptive Search Procedure) methodology was developed in the late 1980s by Feo and Resende [36] and the acronym was coined by Feo and Resende [37]. Basically, each GRASP iteration consists in constructing a trial solution with some greedy randomized procedure and then applying local search to the constructed solution. The construction phase is iterative, randomized, greedy, and

```
1  x* ← ∅
2  f(x*) ← ∞
3  while stopping criterion nos satisfied do
4      x ← ∅
5      Compute C with the candidate elements that can be added to x
6      while C ≠ ∅ do
7          Compute g(c), ∀c ∈ C
8          g_min = min_{c∈C} g(c)
9          g_max = max_{c∈C} g(c)
10         RCL = {c ∈ C : g(c) ≤ (g_min + β(g_max − g_min)), β ∈ [0,1]}
11         Select c* at random from RCL
12         x ← x ∪ {c*}
13         Update C with the candidate elements that can be added to x
14     if x is infeasible then
15         Apply a repair procedure to make x feasible
16     x ← LocalSearch(x)
17     if f(x) < f(x*) then
18         x* ← x
   Output: x*
```

Algorithm 4: GRASP for a minimization problem

adaptive. This two-phase process is repeated until some stopping condition is satisfied.

A best local optimum found over all local searches is returned as the solution of the heuristic. We refer the reader to [40] for a recent survey on this metaheuristic.

Algorithm 4 shows the pseudo-code of a generic GRASP for a minimization problem. The greedy randomized construction seeks to produce a diverse set of good-quality solutions from which to apply the local search phase. Let x be the partial solution under construction in a given iteration and let C be the candidate set with all the remaining elements that can be added to x. The GRASP construction uses a greedy function $g(c)$ to measure the contribution of each candidate element $c \in C$ to the partial solution x. A restricted candidate list RCL is the subset of candidate elements from C with good evaluations according to g. In particular, if g_{min} and g_{max} are the minimum and maximum evaluations of g in C, respectively, then $RCL = \{c \in C : g(c) \leq (g_{\min} + \beta(g_{\max} - g_{\min}))\}$ where $\beta \in [0, 1]$.

3.3.1 Construction method

To obtain a solution for the UrApHMP we first select the p hubs from V (step 1) and then determine the assignments of each node to r of the p hubs (step 2). Finally (step 3), for each pair of nodes, we identify the routing of their traffic through the appropriated hubs. Figure 3.1 illustrates these steps. To do this, let h be a candidate location for a

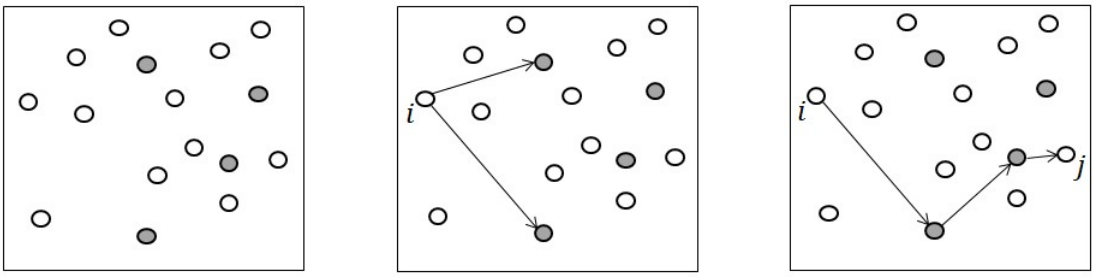

Figure 3.1. Construction steps.

hub. For any node j that could be assigned to h in step 2 consider that all the traffic from j to any other node i, could be routed through h, and in the objective function we would have this traffic t_{ji} multiplied by d_{jh}. We therefore consider the evaluation of this assignment, $e(j, h)$, as: $e\,(j, h) = d_{jh} \sum_{i \in V} t_{ji}$. Note that, since we want to evaluate the attractiveness of h to be a hub, we compute $e\,(j, h)$ for every node j in the graph. On the other hand, it is reasonable to assume that if h is a hub, only a fraction of the nodes will be assigned to it. This is the reason to consider, for the evaluation $g\,(h)$ of h, only the k nodes with lowest $e\,(j, h)$ value. Let us assume, without loss of generality, that they are $j_1, \ldots, j_k$. In mathematical terms, $g\,(h) = \sum_{s=1}^{k} e\,(j_s, h)$.

We now apply the standard method in GRASP to construct a restricted candidate list RCL with good hub locations according to this greedy evaluation g as described above, computing g_{min} and g_{max}, minimum and maximum values respectively over all h. As it is customary in GRASP, g is an adaptive function, i.e., once a hub h_1 is selected, in the following construction steps, when computing $e\,(j, h)$ for a new candidate h, we do

not sum the term $t_{h_1 h}$ since hubs do not need to be assigned to other hubs to route their traffic. We finish step 1 when the p hubs have been selected. Let $H = \{h_1, h_2, \ldots, h_p\}$ be the set of these hubs.

In the step 2 of our constructive method, r of the p hubs are allocated to each node i in the graph. Specifically, for each node i we evaluate its allocation value $alloc\,(i,h)$ to any hub $h \in H$. Note that for any node j to which we need to send traffic from i, this traffic has to be sent through some of their hubs. In other words, to transport the t_{ij} units, a path $i \to h_i \to h_j \to j$ will be used. To simplify the combinatorial problem of determining simultaneously h_i and h_j, we compute $alloc\,(i,h)$ as:

$$alloc\,(i,h) = d_{ih} \sum_{j \in V} t_{ij} + \sum_{j \in V} t_{ij} d_{hj}, \forall i \in V, \forall h \in H,$$

where the first term computes the cost associated with the arc from i to h, and the second one estimates the cost associated with the arcs from h to all destinations j. We then compute $alloc\,(i,h)\,, \forall h \in H$, and assign i to the r hubs with the best (minimum) allocation values. Let $H_i \subseteq H$ be the set of r hubs to which node i is assigned. Note that in step 1 we select hubs in a greedy randomized fashion and in step 2 we assign them to nodes in a greedy way without any random element. We have empirically found that the randomization in step 1 is enough to obtain a diversified set of solutions in our problem. Adding a randomized component in step 2 would result in lower quality solutions.

Finally, in step 3, we route all the traffics through the hub network at their minimum cost. For each pair i and j, we have to determine the hubs $h_i \in H_i$ and $h_j \in H_j$ minimizing the routing cost. In mathematical terms, from the expression of the objective function, and given $h_i \in H_i$ and $h_j \in H_j$, we denote $c_{ij}(h_i, h_j) = t_{ij}\left(\chi d_{ih_i} + \alpha d_{h_i h_j} + \delta d_{h_j j}\right)$.

The routing cost from i to j, c_{ij}, is then obtained by searching the hubs $h_i \in H_i$ and $h_j \in H_j$ minimizing the expression above, i.e.

$$c_{ij} = \min_{h_i \in H_i, h_j \in H_j} c_{ij}(h_i, h_j).$$

Since there is a small number of hubs to which a node is assigned (r is expected to be a small number in typical applications), an exhaustive exploration in this final step can be performed. Specifically, for each pair (i,j) we consider the r^2 associated pairs of hubs to determine the minimum cost c_{ij}. It is specially important to note that even when H_i and H_j have a common hub, it cannot be ensured that the best route will be through that hub, and the computation of all the possibilities mentioned above is needed.

3.3.2 Solution representation

As described in Subsection 3.3.1, three steps are applied to construct a solution: location, assignment and routing. Therefore, to represent a solution we need to specify these three aspects. In particular, a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$ is given by a set of hubs H, a matrix of assignments A, and two matrices of hubs $\mathcal{H}^1$ and $\mathcal{H}^2$ specifying the traffic routes, where:

$$H = \{h_1, h_2, \ldots, h_p\} \subset V$$

$$A = [a_{ij}]_{i=1,\dots,n;j=1,\dots,r}, a_{ij} \in H_i$$

$$\mathcal{H}^1 = \left[h^1_{ij}\right]_{i=1,\dots,n;j=1,\dots,n}, h^1_{ij} \in H_i$$

$$\mathcal{H}^2 = \left[h^2_{ij}\right]_{i=1,\dots,n;j=1,\dots,n}, h^2_{ij} \in H_j.$$

The set H specifies the p hubs in the solution. Each row i of matrix A contains the r hubs assigned to node i, H_i. Finally, for each pair of nodes i and j, we need to represent the path $i \to h_i \to h_j \to j$ used to route the traffic. Matrix $\mathcal{H}^1$ provides the first hub in the route and matrix $\mathcal{H}^2$ the second one, i.e. $\mathcal{H}^1(i,j) = h_i$ and $\mathcal{H}^2(i,j) = h_j$.

Notice that the best hubs to route the traffic t_{ij} from i to j may be different than those to route the traffic t_{ji} from j to i. As previously said, the best hubs to route t_{ij} are those minimizing the cost expression $c_{ij}(h_i, h_j) = t_{ij}\left(\chi d_{ih_i} + \alpha d_{h_ih_j} + \delta d_{h_jj}\right)$, which is not a symmetric expression in terms of i and j when the coefficients χ and δ take different values. Since we need to specify the hubs in the path from i to j and the hubs in the path from j to i, a solution is represented using two separated matrices. Given a pair (i,j), $h_{ia}, h_{ib} \in H_i = \{h_{i1}, \dots, h_{ir}\}$ denote the hubs that node i is assigned to in the two associated paths. Similarly, $h_{jc}, h_{jd} \in H_j = \{h_{j1}, \dots, h_{jr}\}$ are the hubs for j. In particular, $\mathcal{H}^1(i,j) = h_{ia}$, $\mathcal{H}^2(i,j) = h_{jc}$, $\mathcal{H}^1(j,i) = h_{jd}$, and $\mathcal{H}^2(j,i) = h_{ib}$. This

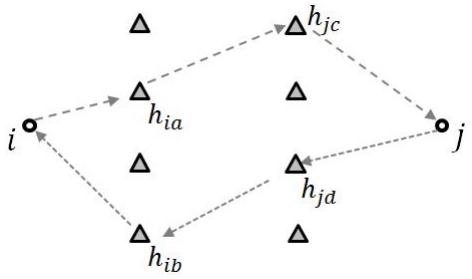

Figure 3.2. Paths between i and j.

is illustrated in Figure 3.2.

3.3.3 Improvement methods

Since the proposed method consists of three steps, and the last one is solved optimally, we apply two types of improvements to the results obtained at step 1 and step 2: changing the hubs selected, and changing the assignments of hubs to nodes.

Given a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, we consider two neighborhoods, N_H and N_A, to improve S. N_H implements a classical exchange in which a hub node h_i is removed from H, and a non-hub node $h'_i \in V \setminus H$ becomes a hub, thus obtaining $H' = \{h_1, \dots, h'_i, \dots, h_p\}$. On the other hand, neighborhood N_A does not affect H but it only considers the nodes assignments. In particular, for a node i assigned to hubs

$H_i = \{h_{i1}, \ldots, h_{ia}, \ldots, h_{ir}\}$, this neighborhood exchanges an assigned hub with a non-assigned one. In mathematical terms, we replace $h_{ia} \in H_i$ with $h'_{ia} \in H \setminus H_i$ obtaining $H'_i = \{h_{i1}, \ldots, h'_{ia}, \ldots, h_{ir}\}$.

In the first neighborhood (N_H), the candidate element h_i to be removed from H is chosen by an evaluation cost that determines the most expensive hub in S in terms of its contribution to the objective function. For a pair of nodes (i, j), we route its traffic t_{ij} in S through the path $i \to h_i \to h_j \to j$ with cost $c_{ij}(h_i, h_j) = t_{ij}\left(\chi d_{ih_i} + \alpha d_{h_ih_j} + \delta d_{h_jj}\right)$. We split this cost into two parts:

$$c^1_{ij}(h_i) = t_{ij}\left(\chi d_{ih_i} + \alpha\frac{1}{2}d_{h_ih_j}\right) \quad \text{and} \quad c^2_{ij}(h_j) = t_{ij}\left(\alpha\frac{1}{2}d_{h_ih_j} + \delta d_{h_jj}\right).$$

We consider that $c^1_{ij}(h_i)$ reflects the cost associated with the use of h_i in the path $i \to h_i \to h_j \to j$. Similarly, $c^2_{ij}(h_j)$ provides an evaluation of the cost of using h_j in this path. In order to measure the total cost associated to a given hub, we sum these values up for all pairs of nodes:

$$c^1_h = \sum_{i,j \in V} c^1_{ij}(h) \quad , \quad c^2_h = \sum_{i,j \in V} c^2_{ij}(h).$$

Hence, we express the evaluation of the cost associated with each hub h as $cost_h = c^1_h + c^2_h$. Now, h^* is selected as the hub such that $cost_{h^*} = \max_{h \in H}\{cost_h\}$. In other words, h^* is, according to this estimation, the most expensive hub in the solution. Therefore, it can be considered as a good candidate to be replaced.

The local search LS_H performs moves in N_H as long as the objective function improves. At each iteration, it selects the hub with the largest contribution to the objective function (according to the estimation above) and searches for a non-hub node to reduce the solution value. We implement here the so-called *first improvement strategy*, in which we perform the first improving move in the neighborhood (instead of scanning the entire neighborhood to determine the best one). Starting from a random element, we examine in increasing order the non-hub nodes searching for the first improving exchange. LS_H terminates when no improving move is found, and the current solution is returned as the output of the procedure.

The evaluation of a move in LS_H is quite time consuming. Given a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, any change in H affects the rest of components in S. Specifically, when node h_i is replaced by node h'_i in H, it is needed to re-evaluate the hub assignment of all the vertices assigned to h_i, since they cannot use this hub anymore and could use any other hub to route their traffic, not necessarily h'_i. In mathematical terms, for any vertex v such that $h_i \in H_v$, the best hub $h \in H \setminus \{h_i\} \bigcup \{h'_i\}$ replacing h_i in H_v has to be selected. Therefore, we have to scan all the vertices assigned to h_i to compute the value associated with this exchange and re-compute matrix A. Moreover, to evaluate these assignments we need to compute and evaluate the routes (i.e., to re-compute $\mathcal{H}^1$ and $\mathcal{H}^2$). Note that even those routes not using h_i have to be re-computed since the new hub in H_v could provide a better route than the current one. Since the update of $A, \mathcal{H}^1$ and $\mathcal{H}^2$ requires a significant computational effort, an alternative would be just

to replace h_i with h'_i in H_v, remaining all the other elements in S the same. We explore this alternative in the next neighborhood, but we can anticipate that, as expected, it produces lower quality solutions in lower running times (as compared with the entire exploration). The complete evaluation of any trial move has been implemented in LS_H.

The second neighborhood N_A only considers moves on A without changing H by exploring the possibility of exchanging a hub h_{ia} to which node i is assigned, with one of the hub nodes $h'_{ia} \in H \setminus H_i$ to which i is not assigned. Based on this neighborhood, we propose two local search procedures, LS_{A1} and LS_{A2}. LS_{A1} implements a simple exploration consisting of replacing h_{ia} with h'_{ia} in all the routes from/to node i. As mentioned above, this could lead to sub-optimal solutions since we are not exploring all the assignments. Consider, for example, the update of the route $i \to h_{ia} \to h_j \to j$ when we replace h_{ia} with h'_{ia} in H_i. With LS_{A1} we would obtain the path $i \to h'_{ia} \to h_j \to j$. Specifically, given a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, when LS_{A1} performs a move changing h_{ia} by h'_{ia}, a new solution is obtained in which H remains unchanged, A only changes in one element (h_{ia} with h'_{ia} in row i), $\mathcal{H}^1$ changes h_{ia} with h'_{ia} in row i (in all its appearances), and similarly $\mathcal{H}^2$ in column i. Note that this move can be done very quickly but it does not consider whether any other hub in H_i can provide a better route from i to j. The local search procedure LS_{A1} performs moves in N_A as long as the objective function improves, exploring the assignments in increasing order, and performing the first improving move found. This local search terminates when no improving move is found, and the current solution is returned as the output of the method.

Method LS_{A2} considers the exchanges of hub assignments of node i (as LS_{A1}) but also explores the other hubs in $H_i \setminus \{h_{ia}\}$ to determine the best one for each particular route starting and finishing at i. Given a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, when LS_{A2} performs a move and changes h_{ia} with h'_{ia}, a new solution is obtained in which H remains unchanged, the matrix A only changes one element (h_{ia} with h'_{ia} in row i), but now $\mathcal{H}^1$ and $\mathcal{H}^2$ are completely recomputed. Since we cannot assure that any hub in a route remains unchanged, routes are computed from scratch like in Step 3 of the constructive method. It is clear that this move is computationally more expensive than the one implemented in LS_{A1}. However, as it will be shown in the comparison of both methods in Section 5, the extra running time is justified in those cases in which we want to match the optimal solution, since LS_{A1} is not able to reach it although it obtains very good results. In order to reduce the computational effort of the algorithm, a filtering mechanism to discard low-quality constructions is proposed. It is described in the next section.

3.3.4 Filtering mechanism

After a number of iterations, it is possible to estimate the fractional improvement achieved by the application of the improvement phase and use this information to increase the efficiency of the search [58]. Let us define the fractional improvement in the iteration t as:

$$P(t) = \frac{c(S_t) - c(S_t^*)}{c(S_t)}$$

where S_t is the solution constructed at iteration t, $c(S_t)$ is its value, and S_t^* is the improved solution obtained applying an improvement method to S_t (and $c(S_t^*)$ its value). This improvement method can be any of the three described in Section 3.3.3, LS_H, LS_{A1} or LS_{A2}, or any combination of them.

After $tmax$ iterations, the mean $\widehat{\mu_P}$ and standard deviation $\widehat{\sigma_P}$ of the improvement P can be estimated as:

$$\widehat{\mu_P} = \frac{\sum_{t=1}^{tmax} P(t)}{tmax}, \quad \widehat{\sigma_P} = \sqrt{\frac{\sum_{t=1}^{tmax} \left(P(t) - \widehat{\mu_P}\right)^2}{tmax - 1}}.$$

Then, these estimates can be used to determine, at a given iteration $q > tmax$, whether it is "likely" that the improvement phase will be able to improve enough the current construction to produce a better solution than the current best one, S_{best}. If this is not the case, we could discard the constructed solution and skip its improvement, saving its associated running time. In particular, we calculate the minimum fractional improvement $\Delta c(q)$ that is necessary for a construction S_q to be better than S_{best}, as:

$$\Delta c(q) = \frac{c(S_q) - c(S_{best})}{c(S_q)}.$$

If $\Delta c(q)$ is close to $\widehat{\mu_P}$, applying the improvement method to the current solution S_q would probably produce a solution S_q^* better than S_{best}. Therefore, in order to save computing time, the improvement method is only applied to the promising solutions S_q, according to this estimation. We can formulate this filtering mechanism as:

$$\Delta c(q) \begin{cases} < \widehat{\mu_P} + \lambda\widehat{\sigma_P}, & \text{apply the improvement method to } S_q\ , \\ \geq \widehat{\mu_P} + \lambda\widehat{\sigma_P}, & \text{discard } S_q. \end{cases}$$

where λ is a search parameter representing a threshold on the number of standard deviations away from the estimated mean percentage improvement. Preliminary experiments to test the effect of different λ values have been performed and are reported in Section 3.3.5.2.

3.3.5 Computational experiments

In this section we describe the computational experiments performed to test the efficiency of the GRASP heuristic. The procedures in our method have been implemented in C and the mixed integer linear programming formulation described in Section 3.2 have been solved using CPLEX 12.4, the most recent version of CPLEX when the experiments were carried out. The results reported in this section have been obtained with an Intel i7 @ 2.7 GHz and 4GB of RAM computer running Windows 7.

The metrics that we use to measure the performance of the algorithms are:

Value Average objective value of the best solutions obtained with the algorithm on the instances considered in the experiment.

Dev Average percentage deviation from the best-known solution (or from the optimal solution, if available).

Best Number of instances in a set for which a procedure is able to find the best-known solution.

CPU Average computing time in seconds employed by the algorithm.

3.3.5.1 Test problems

We have tested our algorithms on three sets of instances:

1. The **CAB** data set. From this original file, 75 instances with 25 nodes and $p \in \{1, \ldots, 5\}$ and $r \in \{1, \ldots, p\}$ have been generated by several authors. The following parameter values have been widely used: $\chi = 1$, $\delta = 1$, and $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$.

2. The **AP** data set. The size of the original data file is 200 nodes. Smaller instances can be obtained using a code from ORLIB. As with CAB, many authors have generated different instances from the original file. We have extended this set of instances by generating 360 instances with $n = 40$, 50, 70, 75, 80, 85, 90, 95, 100, 150, and 200 nodes. For those instances with $40 \leq n \leq 50$, p ranges from 1 to 5. For those with $70 \leq n \leq 95$, p ranges from 1 to 8, and for those with $100 \leq n \leq 200$, p takes values between 1 and 20. In all these cases, $r \in \{1, \ldots, p\}$. According with previous articles, cost parameter values are $\chi = 3.00$, $\alpha = 0.75$, and $\delta = 2$. Regarding the flows between nodes, these instances do not have symmetric flows (i.e., for a given pair of nodes i and j, t_{ij} is not necessarily equal to t_{ji}). Moreover, flows from one node to itself can be positive (i.e., t_{ii} can be strictly positive for a given i).

3. The **USA423** data set. From the original data, 30 instances have been generated with $p \in \{3, 4, 5, 6, 7\}$ and $2 \leq r \leq p - 1$. For each combination of parameters p and r, two different values for χ, α, δ have been used: 0.10, 0.07, 0.09, and 0.09, 0.075, 0.08, respectively.

3.3.5.2 Scientific testing

From the set of 465 instances derived from the CAB, AP, and USA423 data sets described before, we have used a subset of 45, with different sizes and values of p and r, to calibrate the parameters in our method. Specifically, we have considered 15 instances in the CAB set with $n = 25$ and $p \in \{3, 4, 5\}$, and 30 AP instances with $50 \leq n \leq 200$ and $p \in \{3, 4, 5, 6, 7\}$. For these experiments, the instances used have been classified as small ($25 \leq n \leq 45$), medium ($50 \leq n \leq 95$), and large ($100 \leq n \leq 200$).

The constructive method In our first experiment we study the constructive method described in Section 3.3.1 in terms of solution quality and diversification power. Clearly, the performance of this solution generator depends on the value of its two parameters, β, defining the size of the RCL, and k, determining the number of elements in which the evaluation is based on. In order to determine the most effective values for these parameters, we have created a measure of diversity for a set of solutions. We believe that, to perform an effective exploration of the solution space, the constructive method has to be able to generate solutions of a different structure which, in our specific problem, can be interpreted in terms of using different hubs. Therefore, we compute the number of different hubs used in a set of constructed solutions. Specifically, given a set of solutions $P = \{S_1, S_2, \ldots, S_q\}$, where $H_1, H_2, \ldots, H_q$ are their corresponding sets of hubs, the diversity measure, $div\,(P)$, is the number of elements in the set obtained as the union of these sets of hubs. In mathematical terms,

$$div\,(P) = \left| \bigcup_{i=1}^{q} H_i \right| .$$

This diversity metric can be easily interpreted as the number of different hubs in the set of solutions. The larger this value is, the more diversity the algorithm is able to produce in terms of hubs. In the first experiment we have generated $q = 100$ solutions with the constructive method, $k = 1.0$, and different values of α.

Table 3.1. Constructive method with different β values

β	**Dev**	**Best**	*div*	**CPU**
0.1	11.0%	12	27	0.483
0.2	6.7%	12	41	0.486
0.3	6.7%	3	51	0.485
0.4	5.3%	6	56	0.484
0.5	5.4%	2	58	0.485
0.6	4.7%	3	61	0.489
0.7	4.6%	2	62	0.493
0.8	3.3%	13	64	0.485
0.9	3.3%	11	64	0.487
Random	3.5%	10	62	0.468

Table 3.1 shows the metrics described above: Dev, Best and CPU, as well as the average diversity measure, *div*, on the 45 instances of the training set. This table indicates that the best solutions in terms of quality are obtained with $\beta = 0.8$, since the algorithm is able to obtain an average percentage deviation of 3.3% and 13 best known solutions, which compares favorably with the other results. Moreover, with $\beta = 0.8$, we also obtain the best results in terms of diversity (div= 64), which is larger than or equal to the rest of *div* values in this table. We therefore set $\beta = 0.8$ in the rest of the experiments.

Table 3.2. Constructive method with different k values

μ	**Dev**	**Best**	*div*	**CPU**
0.8	1.4%	30	63	0.480
0.9	1.6%	28	63	0.462
1.0	1.1%	33	64	0.462
1.1	1.2%	28	63	0.462
1.2	0.9%	31	64	0.462
1.3	1.8%	27	63	0.461
1.4	2.0%	25	63	0.462
1.5	1.7%	26	63	0.461
1.6	1.9%	26	63	0.462
1.7	1.8%	27	63	0.462
1.8	1.6%	29	63	0.462
1.9	1.4%	26	63	0.462
2.0	1.5%	25	63	0.463

Table 3.2 shows the results of the second experiment in which we run the constructive method with different values of the k parameter on the training set instances. We compute the value of k as a function of the instance size, giving values to μ in the expression $k = \left\lfloor \mu \frac{n}{p} \right\rfloor$. Results in this table show that $\mu = 1.0$ obtains the best values in terms of quality (column Best) and diversity (column *div*). We therefore set $\beta = 0.8$ and $\mu = 1.0$ in the rest of the experiments.

With the goal of supporting our conclusions about the performance of the proposed procedures, we have performed the non-parametric Friedman test for multiple correlated samples to the best solutions obtained by the proposed constructive method with each parameter value in Tables 3.1 and 3.2. This test computes, for each instance, the rank value of each method according to solution quality. Then, it calculates the average rank value for each method across all instances. If the averages differ greatly, the associated p-value or level of significance is small. The resulting p-values of 0.001 and 0.034 obtained with the individual best values in Tables 3.1 and 3.2, respectively, indicate that there are statistically significant differences among the variants tested. The ranks values produced by these tests confirm the selection of $\beta = 0.8$ and $\mu = 1.0$.

Algorithm designs With the search parameters set as indicated above, we proceed to compare the relative merit of the GRASP variants. In particular, we explore the contribution of the three local search algorithms proposed in Section 3.3.3, LS_H, LS_{A1} and LS_{A2}, when applied separately or in combination. Table 3.3 reports the results of five different methods when solving the 45 instances in the training set by generating 100 solutions for each instance. The first one, C, is simply the constructive method with no local search and it is considered as a baseline in this experiment. The next two are GRASP algorithms formed with the constructive method plus either LS_H or

LS_{A2}, denoted $C + LS_H$ and $C + LS_{A2}$, respectively. Finally, the last two GRASP methods combine in their local search phase LS_H with LS_{A1} or with LS_{A2}. In particular, in $C + LS_H + LS_{A1}$, each constructed solution is improved first with LS_H, and the resulting local optimum is then submitted to LS_{A1}, which provides the output of the entire method. Similarly, $C + LS_H + LS_{A2}$ applies LS_{A2} to the solutions obtained with LS_H. Table 3.3 shows the average results obtained according to the size of the instances, classified as small ($1 \leq n \leq 45$), medium ($50 \leq n \leq 95$) , and large ($100 \leq n \leq 200$).

Table 3.3. Comparison of GRASP variants

Algorithm	Size	Dev	CPU
C	small	6.32%	0.01
	medium	10.81%	0.09
	large	8.59%	0.21
	Total	8.11%	0.09
$C + LS_H$	small	0.46%	0.55
	medium	0.06%	34.87
	large	0.10%	179.16
	Total	0.25%	61.90
$C + LS_{A2}$	small	6.00%	0.33
	medium	10.59%	18.99
	large	8.38%	169.66
	Total	7.85%	55.03
$C + LS_H + LS_{A1}$	small	0.35%	0.56
	medium	0.06%	34.94
	large	0.09%	181.17
	Total	0.20%	62.52
$C + LS_H + LS_{A2}$	small	0.00%	0.81
	medium	0.00%	45.48
	large	0.00%	260.83
	Total	0.00%	88.80

Results in Table 3.3 clearly show that $C + LS_H + LS_{A2}$ obtains the best solutions overall (0.00% deviation from best), although it requires the longest running times of the five methods (88.80 seconds on average). Comparing the 0.25% average deviation achieved by $C + LS_H$ on 61.90 seconds with the 7.85% achieved by $C + LS_{A2}$ on 55.03 seconds, we can confirm that LS_H performs a more efficient exploration of the search space than LS_{A2}. However, LS_H is not able to reach the best known solutions by itself, and it needs the post-processing of LS_{A2} to match the best values (as shown with $C + LS_H + LS_{A2}$).

We have applied the Friedman test for paired samples to the data used to generate

Table 3.3. The resulting probability value of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the five methods tested. A typical post-test analysis consists of ranking the methods under comparison according to the average rank values computed with this test. According to this, we obtain that the $C + LS_H + LS_{A2}$ method is the best overall with an average rank of 1.36, followed by $C + LS_H + LS_{A1}$ with an average rank of 2.24 and $C + LS_H$ with 2.40. Finally, we obtain the two methods with larger rank values (as compared with the previous methods): $C + LS_{A2}$ (4.09) and C (4.91).

We compare now the results of $C + LS_H$ and $C + LS_{A2}$ shown in Table 3.3 with two well-known non-parametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The former one answers the question: Do the two samples (solutions obtained with both methods in our case) represent two different populations? The resulting probability value of 0.000 indicates that the values compared come from different methods. On the other hand, the Sign test computes the number of instances on which an algorithm supersedes another one. The resulting probability value of 0.000 indicates that $C + LS_H$ is the clear winner between both methods.

We study now the search profile of the most interesting combinations of local searches. Figure 3.3 shows the progression of the average deviation found by three methods for the training set of instances during 120 iterations of search time. The figure shows how most improvements on the best solution obtained with the $C + LS_H + LS_{A2}$ and $C + LS_H$ methods are achieved early in the search (i.e., within 10% of the number of iterations). After that point, both methods stagnate, and only exhibit a marginal improvement in the next iterations. On the other hand, $C + LS_{A2}$ performs worse, with an average percentage deviation of several orders of magnitude larger than the other methods.

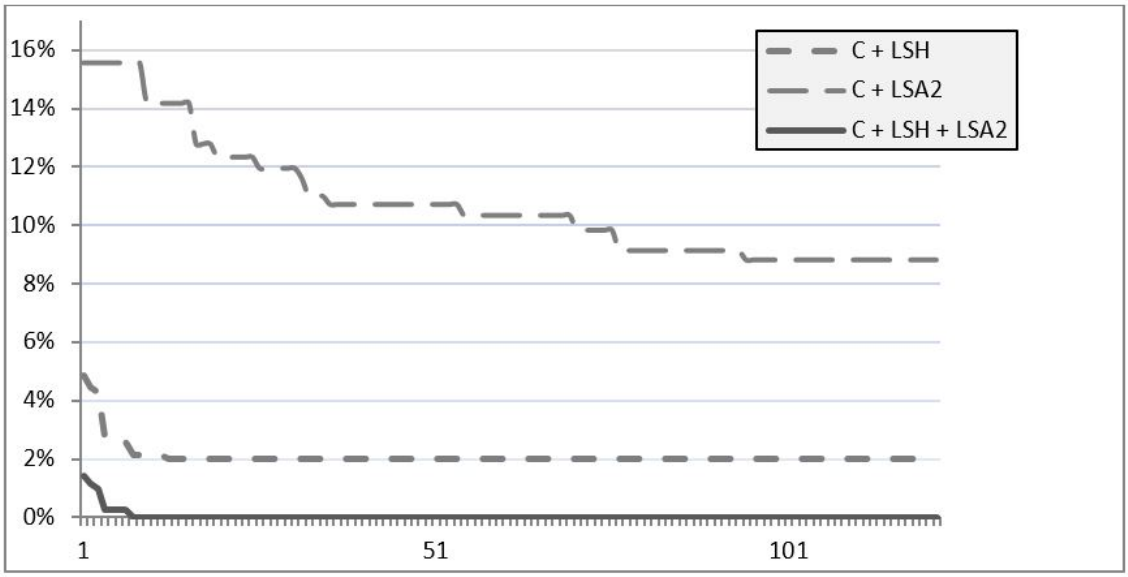

Figure 3.3. Search profile.

The filtering mechanism In our next experiment we test the efficiency of the filtering mechanism described in Section 3.3.4. Table 3.4 reports the results obtained with the $C + LS_H + LS_{A2}$ method when running with different values of the two filter parameters: $tmax$, the number of initial iterations for which we compute the mean $\widehat{\mu_P}$ and standard deviation $\widehat{\sigma_P}$ of the improvement achieved with the local search, and λ, the value to compute the filtering threshold $\widehat{\mu_P} + \lambda\widehat{\sigma_P}$. Specifically, it reports the average Dev and CPU, as in the previous experiments, and the average number of solutions discarded for improvement, # of skip, out of the 100 solutions constructed.

Table 3.4. Filtering GRASP constructions

tmax	λ	**Dev**	**CPU**	**# of skip**
	-1	0.152%	15.0	76.0
	0	0.135%	34.4	44.5
10	1	0.000%	53.9	16.3
	2	0.000%	60.7	4.2
	3	0.000%	62.4	1.1
	-1	0.151%	21.3	67.8
	0	0.151%	39.1	39.9
20	1	0.072%	53.9	14.8
	2	0.000%	59.4	2.9
	3	0.000%	60.3	0.6

Table 3.4 shows that, as expected, the larger the λ value, the longer the CPU time. In other words, with low λ values the method discards more constructed solutions and therefore requires less CPU time than with larger values (given that it saves the computation of the local search associated with the discarded solutions). For example, with $\lambda = 0$ and $tmax = 20$, an average of 39.9 out of the 100 solutions constructed are discarded and the method only applies the local search to the remaining solutions. This has two consequences: the first one is that the CPU time is 39.1, which is lower than the 88.80 reported in Table 3.3 in which no filtering was applied and the 100 constructed solutions were submitted to the local search. The second one is that the average percentage deviation is 0.151% instead of the 0.00% of the unfiltered GRASP. On the other hand, this table shows that there are small differences when $tmax = 10$ or 20 with a slightly improvement in the former case. Therefore, in the following experiments we set $tmax = 10$ and denote the method as GRASP(λ) selecting the λ value in each experiment according to this trade-off between computing time and solution quality.

3.3.5.3 Competitive testing

Comparison with optimal solutions In Section 3.2 we have described the mixed integer formulation proposed by Yaman [103] for the UrApHMP. We have used CPLEX to solve 105 small instances with n ranging from 25 to 50, and $p, r \in \{1, 2, 3, 4, 5\}$.

Table 3.5 reports the average percentage deviation with respect to the optimal solution

Table 3.5. GRASP deviations from the optimal value

			CPLEX	**GRASP(0)**			**GRASP(3)**		
n	p	**# ins**	**CPU**	**Opt**	**Dev**	**CPU**	**Opt**	**Dev**	**CPU**
25	1	5	37.92	5	0.00%	0.04	5	0.00%	0.05
	2	10	25.42	10	0.00%	0.12	10	0.00%	0.21
	3	15	21.62	15	0.00%	0.22	15	0.00%	0.49
	4	20	18.85	15	0.17%	0.42	15	0.17%	1.17
	5	25	17.30	24	0.03%	0.92	24	0.03%	2.14
	Total	75	21.04	69	0.06%	0.48	69	0.06%	1.15
40	1	1	682.36	1	0.00%	0.16	1	0.00%	0.27
	2	2	226.71	2	0.00%	0.56	2	0.00%	1.04
	3	3	283.05	3	0.00%	0.79	3	0.00%	2.18
	4	4	198.89	4	0.00%	2.48	4	0.00%	5.75
	5	5	166.31	4	0.04%	4.18	4	0.04%	9.10
	Total	15	240.80	14	0.01%	2.30	14	0.01%	5.16
50	1	1	3730.80	1	0.00%	0.27	1	0.00%	0.50
	2	2	843.34	2	0.00%	1.09	2	0.00%	1.85
	3	3	818.61	3	0.00%	3.29	3	0.00%	5.22
	4	4	786.44	4	0.00%	7.16	4	0.00%	11.46
	5	5	596.66	5	0.00%	11.01	5	0.00%	19.77
	Total	15	933.49	15	0.00%	6.40	15	0.00%	10.97

of GRASP(λ) with $\lambda = 0$, which filters many constructions as shown above, and $\lambda = 3$, in which the filter is basically not applied. Note that in this experiment, instead of reporting the number of best solutions found, we report the number of instances in which the method is able to match the optimal solution (Opt). This table also includes the average running time in seconds employed by CPLEX to obtain the optimal solutions.

Table 3.5 shows that the GRASP method is able to obtain the optimal solution in most cases. Both the filtered variant, GRASP(0), and the unfiltered one, GRASP(3), obtain 98 optimal values out of the 105 instances considered. As a matter of fact, the only difference between the results obtained with the two methods is the running time. Since the values reported in this table are average values, we complement its information with the range of the deviation and running times for the 7 instances in which the heuristic is not able to match the optimum value. Specifically, the CPU of GRASP(0) ranges in these instances from 0.16 to 2.92 seconds, while its percentage deviation ranges from 0.21% to 0.95%. It is worth mentioning that CPLEX requires up to 2 hours of computing time to obtain the optimal value on some of the instances with $n = 50$, and that larger instances cannot be solved due to memory requirements. Therefore, we cannot provide the optimal values for larger instances.

Comparison with optimal assignments As it has been described before, to obtain a solution for this problem we face a three step process:

1. Selecting the p hubs,
2. Determining the assignments of each node to r of the p hubs, and
3. Identifying, for each pair of nodes, the optimal route through the appropriated hubs.

In the previous section, we have considered the linear integer formulation in [103] to obtain the optimal solution for the small instances of this problem. In order to measure the quality of the results obtained with our method, and considering that we cannot solve larger instances using this formulation, we have adapted this formulation to optimally solve the assignment and routing subproblems (steps 2 and 3 above). In particular, we first use our heuristic to select the set H of p hubs. Then, to assign the nodes to hubs and compute the traffics among nodes, we solve the following integer problem:

$$\min \sum_{i\in V}\sum_{j\in V}\sum_{k\in H}\sum_{l\in H} t_{ij}\left(\chi d_{ik} + \alpha d_{kl} + \delta d_{lj}\right) x_{ijkl} \tag{3.10}$$

Subject to

$$\sum_{k\in H} z_{ik} \leq r, \qquad \forall i \in V \tag{3.11}$$

$$\sum_{k\in H}\sum_{l\in H} x_{ijkl} = 1, \qquad \forall i,j \in V \tag{3.12}$$

$$\sum_{l\in H} x_{ijkl} \leq z_{ik}, \qquad \forall i,j \in V, \quad \forall k \in H \tag{3.13}$$

$$\sum_{k\in H} x_{ijkl} \leq z_{jl}, \qquad \forall i,j \in V, \quad \forall l \in H \tag{3.14}$$

$$\sum_{k\in H} x_{ijkl} \leq z_{jl}, \qquad \forall i,j \in V, \quad \forall l \in H \tag{3.15}$$

$$x_{ijkl} \geq 0, \qquad \forall i,j \in V, \quad \forall k,l \in H \tag{3.16}$$

$$z_{ik} \in \{0,1\}, \qquad \forall i \in V, \quad \forall k \in H. \tag{3.17}$$

Table 3.6. GRASP deviations from the assignment and routing optimal values on AP instances

Size	n	# instances	Opt	Dev
Medium	70	35	35	0.00%
	75	35	35	0.00%
	80	35	35	0.00%
	85	35	35	0.00%
	90	35	35	0.00%
	95	35	35	0.00%
	Total	210	210	0.00%
Large	100	36	36	0.00%
	150	36	36	0.00%
	200	36	36	0.00%
	Total	108	108	0.00%

This formulation, in which the set of hubs is fixed, is a special case of the one described in Section 3.2. We use it with CPLEX to obtain the optimal solution of the assignment and routing subproblems for a specific set of hubs. It is clear that, since H has been obtained heuristically, the optimality of the resulting solution cannot be guaranteed. However, with this experiment, we are able to test whether our algorithm is obtaining or not the optimal solution in steps 2 and 3 for a given solution of step 1.

Table 3.6 shows, for the 318 medium and large instances in the AP set, its number of nodes (n), the number of instances (# instances) tested for each value of n, the number of instances in which the GRASP obtains the optimal solution in steps 2 and 3 (Opt), and the average percentage deviation with respect to the assignment and routing optimal values (Dev). The results in this table clearly show that our GRASP algorithm is able to match the optimal assignment and routing values in all the AP instances tested.

In order to study the behavior of our algorithm on larger instances, we have repeated the above experiment on the set of 30 instances generated from the USA423 set. As it has been mentioned, all the 30 instances have 423 nodes. Two different sets of values for the cost parameters χ, α, δ have been used for each instance: 0.10, 0.07, 0.09 (denoted by A), and 0.09, 0.075, 0.08 (denoted by B). The results are shown in Table 3.7. CPU_{GRASP} column denotes the maximum time allowed to the GRASP, while CPU_{CPLEX} shows the time needed by CPLEX to get the optimal allocation and routing values for the set of p hubs obtained with the GRASP. All times are shown in **minutes**. Although in these instances the GRASP algorithm cannot match the optimal assignment and routing values, in our opinion the results shown in Table 3.7 are very good. They show a deviation from the optimal values that never exceeds a 0.82% on average, and only in two out of the 30 instances slightly exceeds 1%. CPLEX needs more than 20 hours of computing time on average to solve these instances.

Table 3.7. GRASP deviations from the assignment and routing optimal values on the USA423 instances

Cost parameters	p	# inst	Dev	$\mathbf{CPU}_{GRASP}$	$\mathbf{CPU}_{CPLEX}$
A	3	1	0.00%	30	21.5
	4	2	0.01%	30	224.5
	5	3	0.12%	30	197.7
	6	4	0.21%	60	148.1
	7	5	0.82%	60	1232.1
B	3	1	0.00%	30	0.5
	4	2	0.01%	30	52.6
	5	3	0.20%	30	38.7
	6	4	0.19%	60	627.5
	7	5	0.78%	60	280.4

Comparison with previous heuristics As far as we know, there is no previous heuristic for the UrApHMP, in which each node can be connected to at most r of the p hubs. However, two particular cases of this general problem, the single and the multiple allocation problems, have been extensively studied, so we can compare our method with previously proposed heuristics for these two cases. In particular, in the uncapacitated *multiple* allocation p-hub median problem, each node can send and receive traffic through any of the p hubs, while in the uncapacitated *single* allocation p-hub median problem, each node is assigned to one of the p hubs, only allowing sending and receiving traffic through this single hub. Although we have designed our GRASP algorithm to solve the general uncapacitated r-allocation p-hub median problem and it does not take advantage of the specific characteristics of these two special cases, we can apply it to solve them and to ascertain if the GRASP is able to compete with recently published methods specially proposed for these multiple and single allocation versions.

Table 3.8. GRASP vs. evolutionary method with $r = p$

		$\mathbf{GRASP}_1$			$\mathbf{GRASP}_{10}$			Evolutionary		
n	# inst	Best	Dev	CPU	Best	Dev	CPU	Best	Dev	CPU
10	7	5	0.77%	0.00	7	0.00%	0.01	7	0.00%	0.05
20	7	6	0.27%	0.02	7	0.00%	0.10	7	0.00%	0.16
25	7	5	0.20%	0.04	7	0.00%	0.21	7	0.00%	0.24
40	9	7	0.28%	0.41	9	0.00%	1.94	9	0.00%	1.19
50	9	7	0.08%	0.83	8	0.00%	4.87	9	0.00%	7.36
100	11	6	0.22%	40.91	10	0.00%	321.00	10	0.01%	67.93
200	11	5	0.10%	780.42	9	0.00%	1879.25	9	0.08%	346.62
Total	61	41	0.25%	148.30	57	0.00%	373.12	58	0.01%	76.07

Table 3.8 shows the results of our GRASP when solving the multiple allocation version. This table shows the results obtained with the GRASP for a single iteration (one construction plus the local search), denoted $GRASP_1$, and for 10 iterations, $GRASP_{10}$. The table also reports the results of the evolutionary approach recently proposed by Milanovic [71] for the multiple allocation version. The three algorithms are applied on the 61 AP instances reported in that paper. Specifically, Table 3.8 shows the size and number of instances in each row, and, for each method, the number of instances in which it obtains the best known solution, the average percentage deviation with respect to this best, and the CPU in seconds. Note that in $GRASP_{10}$ the time reported corresponds to the total CPU required for the 10 iterations. Results for the evolutionary method are directly taken from Milanovic [71], and therefore running times are only indicative and cannot be directly compared with GRASP computing times.

Results in Table 3.8 clearly show that the GRASP algorithm is able to obtain state–of–the–art results for the multiple allocation problem. Even $GRASP_1$, the version in which only one solution is generated, performs remarkably well, obtaining results of similar quality to those reported with the evolutionary method by Milanovic [71], specifically designed for this problem. As expected, the $GRASP_{10}$ variant, in which the method is applied for 10 iterations, is more time consuming than $GRASP_1$, and is able to match, and in some cases surpass, the evolutionary method. In particular, it obtains a new best known value of 92646.38 in the AP instance with n=200 and p=15 for which the previously best known value was 92669.64.

Table 3.9. GRASP vs. GA with $r = 1$

		$GRASP_1$		**$GRASP_{10}$**		**GA**	
n	p	**Dev**	**CPU**	**Dev**	**CPU**	**Dev**	**CPU**
	2	0.04%	0.17	0.04%	0.98	0.00%	13.29
	3	0.00%	0.41	0.00%	4.38	0.00%	16.64
	4	0.03%	0.41	0.03%	4.40	0.00%	17.76
100	5	0.04%	0.91	0.04%	5.29	0.00%	22.11
	10	0.00%	3.47	0.00%	34.80	0.00%	40.73
	15	1.43%	8.42	0.26%	111.51	0.00%	57.54
	20	0.04%	16.08	0.04%	67.89	0.00%	79.20
	2	0.00%	1.82	0.00%	15.88	0.00%	100.72
	3	0.07%	3.61	0.07%	43.48	0.00%	111.77
	4	0.03%	5.78	0.03%	30.48	0.00%	131.16
200	5	0.45%	10.08	0.26%	54.43	0.08%	169.73
	10	0.39%	39.63	0.39%	454.83	0.00%	259.14
	15	1.07%	142.34	0.56%	1213.85	0.04%	313.02
	20	1.88%	331.82	0.11%	3432.02	0.20%	374.75
Average		0.39%	40.35	0.13%	391.02	0.02%	121.97

Table 3.9 shows the results for the single allocation problem. As in the previous experiment, we report the results with $GRASP_1$ and $GRASP_{10}$. We also report in this table the results of the Genetic Algorithm (GA) described in Kratica et al. [55]. This table shows the Dev and CPU values for each method on the 14 AP instances reported in that paper. To complement this comparison, Table 3.10 reports the results of our two GRASP variants and the VNS approach presented by Ilic et al. [50] on the 8 AP instances reported in that paper. These are the hardest AP instances for this problem.

Tables 3.9 and 3.10 show that the GRASP variants are able to obtain good results on the single allocation version, although they behave slightly worse than the specialized algorithms for this problem. In particular, the GA by Kratica et al. [55] shows an average percentage deviation of 0.02% obtained in 121.97 seconds, while $GRASP_1$ and $GRASP_{10}$ obtain an average percentage deviation of 0.39% and 0.13% in 69.06 and 671.83 seconds, respectively. The VNS by Ilic et al. [50] performs remarkably well since it is able to achieve an average percentage deviation of 0.00% in 6.12 seconds. However, the GRASP algorithm is not designed to exploit the particular characteristics of the single allocation version of this problem, as it does the VNS, and the objective of this comparison is to show that it performs relatively well across different types of p-hub location problems.

Table 3.10. GRASP vs. VNS with $r = 1$

		$GRASP_1$		**$GRASP_{10}$**		**VNS**	
n	p	**Dev**	**CPU**	**Dev**	**CPU**	**Dev**	**CPU**
100	5	0.04%	0.91	0.04%	5.29	0.00%	0.08
	10	0.00%	3.47	0.00%	34.80	0.00%	0.67
	15	1.43%	8.42	0.26%	111.51	0.00%	3.22
	20	0.04%	16.08	0.04%	67.89	0.00%	3.57
200	5	0.45%	10.08	0.26%	54.43	0.00%	5.16
	10	0.39%	39.63	0.39%	454.83	0.00%	5.60
	15	1.07%	142.34	0.56%	1213.85	0.00%	17.66
	20	1.88%	331.82	0.11%	3432.02	0.00%	12.98
Average		0.66%	69.09	0.21%	671.83	0.00%	6.12

3.3.5.4 Run time distribution

Aiex, Resende, and Ribeiro observed [3] that the variable *time-to-target-value* in GRASP usually has an exponential distribution. Time-to-target (TTT) plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. TTT plots are used to characterize the running times of stochastic algorithms for combinatorial optimization. Specifically, for each instance/target pair, the running times are sorted in

increasing order. We associate with the i-th sorted running time t_i a probability

$$p_i = \frac{(\frac{i-1}{2})}{n}$$

and plot the points (t_i, p_i). The resulting diagram shows the cumulative probability distribution plot and permits to check whether a given algorithm has or not an exponential distribution.

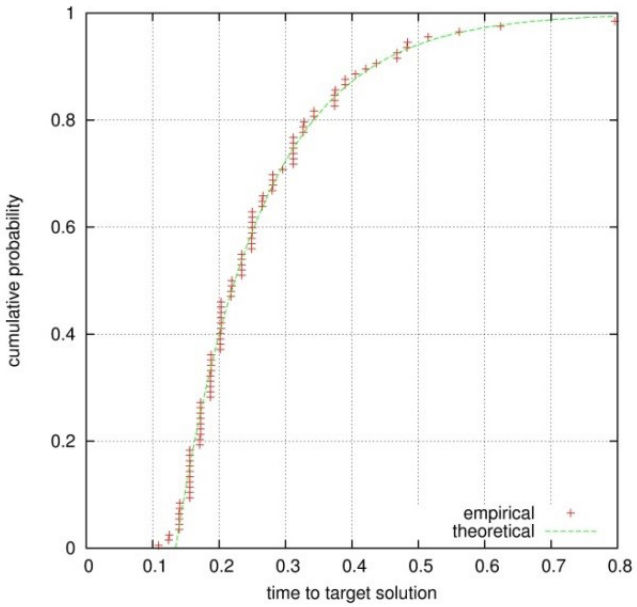

Figure 3.4. Time to target plot.

We ran 100 times our GRASP for the UrApHMP on a representative instance, stopping when a solution is found with objective value equal to the best known for this instance. For each run we recorded the running time. Each run is independent of the other by using a different initial seed for the random number generator. With these 100 running times, we plot the time-to-target plot (run time distributions) shown in Figure 3.4, in which we add the theoretical exponential distribution. This experiment confirms the expected exponential runtime distribution for our GRASP. Therefore, linear speed is expected if the algorithm is implemented in parallel.

3.3.6 Concluding remarks

We have developed in this section a heuristic procedure based on the GRASP methodology that provides high quality solutions for the uncapacitated r-allocation p-hub median problem. We have explored the critical issue of which solution-generation-method proves effective to obtain a good set of solutions in terms of quality and diversity. We have defined three neighbourhoods in the local search and a filtering mechanism to selectively apply it. Overall experiments with 465 instances have been performed to assess the merit of the procedures developed here. Our implementation has shown to be competitive in a set of instances previously reported in the Literature. Moreover, the procedure has been shown to be robust in terms of solution quality within a reasonable computational effort. The proposed method has been also compared with the formulation implemented in CPLEX and with previous heuristics for different p-hubs variants (single and multiple allocation problems). The experimentation shows that the GRASP method is able to obtain high quality solutions across different p-hub median problems.

3.4 A scatter search algorithm

Scatter Search (SS) may be considered, from the standpoint of a metaheuristic classification, as a population-based algorithm [59] that constructs solutions by combining others to efficiently solve $\mathcal{NP}$-hard optimization problems. It derives its foundations from strategies originally proposed for combining decision rules and constraints in the context of integer programming [43, 45, 67]. Path-relinking (PR) was originally proposed in the context of tabu search [42, 83, 88] as a method to find high quality solutions in the path between two good solutions.

If solutions of a combinatorial optimization problem are seen as points in a space, the exploration of this space is done in scatter search by evolving a set of reference points. These points define a set, known as *Reference Set* (*RefSet*), and typically consist of good solutions obtained by prior problem solving efforts. A given iteration of the SS method generally consists of three main steps: to combine the solutions of *RefSet*, to improve the solutions obtained, and to update *RefSet* with the resulting solutions that are better than those currently in *RefSet*. In the next sections we explain how these steps are adapted to solve the UrApHMP.

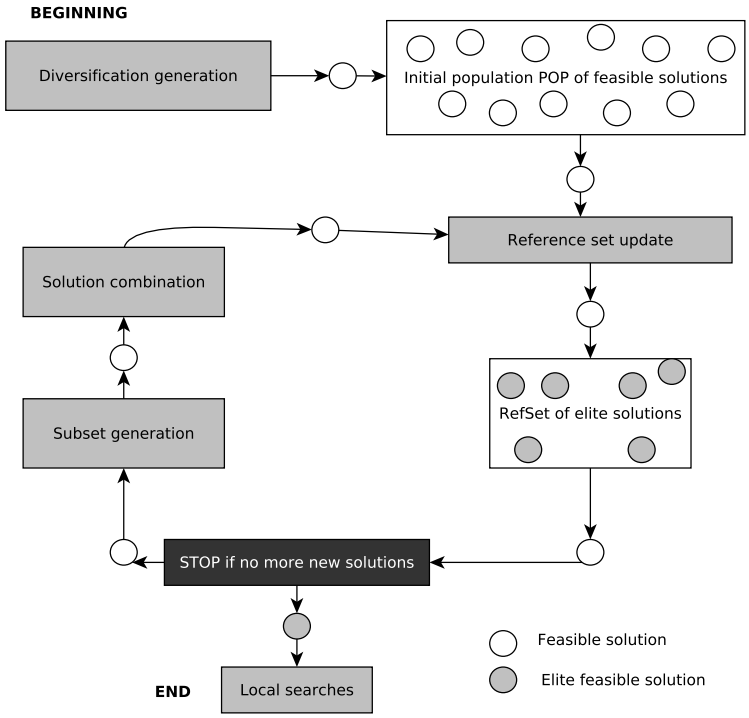

Figure 3.5. Scheme of the proposed scatter search algorithm

As shown in Figure 3.5, scatter search starts by generating a set *Pop* of diverse solu-

tions (further details will be given in Section 3.4.1). Then, it selects β of them to create *RefSet*. The standard design selects the best $\frac{\beta}{2}$ solutions in *Pop*, and then the most diverse $\frac{\beta}{2}$ solutions w.r.t. the solutions already in *RefSet* (for more details, see Section 3.4.2). The main loop of the method consists of applying the combination method to all the pairs of solutions in *RefSet*. As it will be described in Section 3.4.4, path-relinking is the method we will use to create new solutions from each pair of solutions selected from *RefSet*. Path-relinking is an intensification strategy to explore trajectories connecting good (also known as *elite*) solutions obtained by other heuristic methods. When hybridizing both methodologies, SS and PR, it seems natural to consider path-relinking as the method for combining the solutions in *RefSet*, generating paths between and beyond these reference points. Once a new solution has been obtained, the SS mechanism decides whether it is included or not in *RefSet*. We have implement a standard *RefSet* update method that only includes a solution in *RefSet* if it is better than the worst solution and if it also provides sufficient diversity according to a "distance" between the solution and *RefSet*. Finally, the improvement method we propose to apply to the solutions found is described in Subsection 3.4.6. In what follows we describe the application of the methods that conform of the scatter search methodology to solve the UrApHMP.

3.4.1 The diversification generator method

The diversification generator method (DGM) yields a population *Pop* of π initial feasible solutions for the problem. It seeks to balance solutions' quality, in terms of cost, and diversity, in terms of attributes, and can be seen as the mechanism that creates the first generation of solutions. If the attributes of this first generation of solutions are good, there is some confidence of getting better solutions in the following iterations after strategically combining them.

To create *Pop*, we have developed and tested seven different DGM for the UrApHMP. Six of them are based on GRASP constructions that differ among them in the implementation designs and in the different expressions for the evaluations of hubs that will be used in each of the solutions. The seventh is simply a random construction to provide diversity to *Pop*.

As it is well known, in the *semi-greedy* implementation of GRASP, each element of a solution is iteratively selected by evaluating all candidate elements with respect to a greedy function g that measures their attractiveness. Only the q elements with best g values are placed in a restricted candidate list (RCL), where q is a search parameter. Then, an element in the RCL is randomly selected, according to a uniform distribution, to become part of the solution. The values of g are updated at each iteration to reflect the changes brought on by the selection of the previous element. An interesting variant of the classic GRASP design has been recently proposed in [84]. It is based on a *sampled greedy* that first builds a RCL by uniformly sampling q elements at random. Then, g is evaluated over these q elements. The RCL element with the best g value is added to the solution under construction. In both GRASP implementation, the value of q controls the trade-off between greediness and randomness. In the first design, lower q values favor

greedy selection (w.r.t. randomization), while this is obtained with large q values in the second design.

In our implementation, we have based the g evaluation for the selection of the p hubs on a cost criterion. Let $h \in V$ be a candidate node to be used as hub. If h were a hub, it would be used for the transportation of the traffics among some terminals, possibly the φ terminals $i_1, \ldots, i_\varphi$ with lower assignment cost to h, where φ is a given parameter. We then compute $g(h)$ as:

$$g(h) = \sum_{s=1}^{s=\varphi} cost(i_s, h), \quad \forall h \in V,$$

where $cost(i, h)$ represents the assignment cost of terminal i to hub h. We propose several ways for computing $cost(i, h)$. In all of them, let $\overrightarrow{T_i} = \sum_{j \in V} t_{ij}$ be the sum of all the traffics from i to all nodes j. Similarly, let $\overleftarrow{T_i} = \sum_{j \in V} t_{ji}$ be the sum of all the traffics from all nodes j to node i. The different alternatives for computing the *cost* functions are:

Type 1 Here, $cost(i, h)$ is computed as the cost of sending $\overrightarrow{T_i}$ from i to h, i.e., $cost(i, h) = d_{ih}\overrightarrow{T_i}$. This evaluation cost is the one proposed in Section 3.3.1 and produces good quality solutions.

Type 2 Now, we also consider the cost of receiving the incoming traffic $\overleftarrow{T_i}$ for i from h, hence $cost(i, h) = d_{ih}\overrightarrow{T_i} + d_{hi}\overleftarrow{T_i}$.

Type 3 In addition to the costs of transporting the incoming an outgoing traffics, we consider here some of the discounting factors that are usually present in the UrApHMP. To do this, we enrich the *cost* expression as $cost(i, h) = \chi d_{ih}\overrightarrow{T_i} + \frac{\alpha + \delta}{2} d_{hi}\overleftarrow{T_i}$, where χ, α and δ are the unit rates for collection (origin-hub), transfer (hub-hub), and distribution (hub-destination), respectively (see, for instance, [103]).

The three types of *cost* functions above, combined with the two GRASP designs (the semi-greedy and the sampled greedy), are applied to populate *Pop*. As it will be shown later, they have proven to be effective to obtain a balanced *Pop* set of good quality and diverse solutions. We call these methods DGM1 to DGM6, as shown in Table 3.11. A last method, called DGM7, is based on the notion of constructing solutions at random by simply generating random sets of p hubs, helping to create a limited amount of solutions not guided by an evaluation function to just bring diversity to *Pop* for avoiding premature convergence of the algorithm.

Once the p hubs are selected for a solution, the following step is to allocate r of the p hubs to each terminal. Let $H^i \subseteq H$ be the set of the r hubs assigned to terminal i in a solution. For any terminal i we compute the following estimation of the assignment cost of i to a hub h as:

$$assignment(i, h) = d_{ih}\overrightarrow{T_i} + \sum_{j \in V} d_{hj} t_{ij}.$$

Table 3.11. Classification of the different diversification generation methods.

	Semi-greedy	Sampled greedy
Type 1	DGM1	DGM4
Type 2	DGM2	DGM5
Type 3	DGM3	DGM6

Then, we assign to i the hub h_a with the lowest *assignment* cost. Then, $h_a \in H^i$. For the remaining assignments to i, the above expression is updated to reflect the previous assignments:

$$assignment(i,h) = d_{ih}\overrightarrow{T_i} + \sum_{j \in V \setminus H^i} d_{hj}t_{ij} - \sum_{u \in H^i} d_{iu}t_{iu}, \quad \forall h \in H \setminus H^i.$$

This process is done in a greedy way, selecting at each iteration the lowest assignment cost. Note that the *assignment* expressions are an estimation, because they assume that there is only one hub h in the path between any pair of nodes i and j, which is not necessarily true. Finally, we route all the traffics at their minimum cost, like we have previously explained in Section 3.3.1.

Once the p hubs have been located, r hubs have been assigned to each node and all the traffics have been routed, we have a feasible solution for the UrApHMP. A solution is denoted, based on Section 3.3.2, by $s = (H, A)$ and its cost by $f(s)$, where $H = \{h_1, \ldots, h_p\} \subseteq V$ is the set of hubs in the solution and $A = [a_{ij}]_{i=1,\ldots,n;j=1,\ldots,r}$, $a_{ij} \in H^i$, is a matrix whose rows contain the r hubs assigned to each node.

3.4.2 The reference set construction method

Using methods DGM1 to DGM7, we generate π feasible solutions, but only β of them, the *best* ones, will become part of *RefSet*. The notion of *best* solution is not limited here to its quality, as a measure given by the objective function, but also to the diversity that each solution brings to *RefSet* in terms of its attributes.

The process to select the β solutions of *Pop* that will define *RefSet* is done as follows: As the SS methodology specifies [59], we want to choose $\frac{\beta}{2}$ solutions attending their quality. To do this, we order all solutions in *Pop* by ascending order of their costs and introduce them, one by one, only if there is no other solution already introduced in *RefSet* with the same cost. We stop this selection process after examining the 50% of the solutions in *Pop*, even if they are less than $\frac{\beta}{2}$ solutions. The rest of the solutions of *RefSet* will be selected from *Pop* by a diversity criterion process that tries to choose those solutions of *Pop* that differ most from *RefSet*.

Given $s \notin RefSet$ and $t \in RefSet$, let $C = \{h : h \in H_s \cap H_t\}$ be the set of common hubs in solutions s and t. We define $d_H(s,t) = p - |C|$ as the number of hubs in s not present in t (or viceversa). We can consider the lower the value of $d_H(s,t)$ is, the closer s and t are. To select the solution $s \in Pop$ to be included in *RefSet*, we define

$dist(s, RefSet) = \min_{t \in RefSet} d_H(s,t)$. This distance is computed for all solutions in *Pop*, and the solution s^* with maximum value of $dist(s, RefSet)$ is introduced in *RefSet*. Another distance function, d_A, has been defined to break a possible tie when two or more solutions $s_1, \ldots, s_z \in Pop$ take the same *dist* value ($dist(s_1, RefSet) = \ldots = dist(s_z, RefSet)$). Distance d_A is based on the assignments of terminals to the hubs in C. Given $s \notin RefSet$, $t \in RefSet$, and $h \in C$, let T_s^h be the set of terminals assigned to hub h in solution s. Then $|T_s^h \cap T_t^h|$ gives the number of common assignments to h in both solutions. We define $d_A(s,t) = \min_{h \in C} |T_s^h \cap T_t^h|$. If a tie occurs, we select the solution with the maximum d_A value.

3.4.3 The subset generation method

Once the *RefSet* has been created, the subset generation method (SGM) consists of producing, at any iteration, different sets $X \subset RefSet$ to serve as a basis for the application of the combination method later on. The SGM we propose works by ordering the solutions of *RefSet* in ascending order of their cost (from best to worst) and then generating subsets defined by any two different solutions. To avoid the repetition of previously generated subsets at earlier iterations, each subset is generated only if at least one of its solutions was introduced in *RefSet* in the preceding iteration. Suppose that m subsets were not considered for this reason, then the number of resulting subsets at each iteration for which the combination method will be applied is $\frac{\beta^2-\beta}{2} - m$.

3.4.4 The solution combination method

The solution combination method (SCM) is an element of scatter search that is context-dependent. Although it is possible to design "generic" combination procedures, we thought that it would be more effective to design the SCM based on the characteristics of the UrApHMP. Our proposal for the SCM is a path-relinking implementation that is applied to each subset generated in the previous step. As it has been mentioned, path-relinking is an intensification procedure that explores *paths* (also called *trajectories*) in the neighborhood space of two good solutions, generating intermediate solutions that can eventually be better than the two being connected.

More formally, let $\mathcal{G} = (F, M)$ be the search space graph, where node set F is the set of feasible solutions of the problem, and M is the set of edges associated with the moves in the neighborhood structure. Given two solutions $s, t \in F$, a move is defined as $(s,t) \in M$ if and only if $s \in \mathcal{N}(t)$ and $t \in \mathcal{N}(s)$, where $\mathcal{N}$ is a given neighborhood structure. The path-relinking operator explores a path $\mathcal{P}(s,t)$ that connects s and t with the objective of finding solutions $s^* \in \mathcal{P}(s,t)$ for which $f(s^*) < \min\{f(s), f(t)\}$. Let s be the "initial" solution of the path and t its "guiding" solution. This path is generically accomplished by swapping out elements selected in s with elements in t, generating a set of *intermediate* solutions. To obtain a first intermediate solution s' in the path $\mathcal{P}(s,t)$, we remove a hub v from H_s and replace it by a hub u in H_t, thus obtaining $H_{s'} = H_s \setminus \{v\} \cup \{u\}$. Let $\Delta(s,t)$ be the set of attributes present in t but not in s. In the UrApHMP, we define $\Delta(s,t)$ as the set of nodes that are hubs in t but not

in s. In mathematical terms, $\Delta(s,t) = \{u : u \in H_t, u \notin H_s\}$. Note that one or more first intermediate solutions can be obtained when $|\Delta(s,t)| > 1$.

To combine the solutions in the subsets X generated from *RefSet*, the procedure we have designed works as follows: For each subset $X = \{s,t\}$, we explore the path between its two solutions. To do this, we consider as initial solution the one with worse value, s. Then, we calculate $\Delta(s,t)$. If $|\Delta(s,t)| \geq 2$, it means that at least two hubs of H_t are not in H_s. If $|\Delta(s,t)| \leq 1$, no path-relinking is necessary as there are no intermediate solutions between s and t. Our PR generates $m = |\Delta(s,t)|^2$ different solutions, and each terminal node of these m solutions needs to be assigned to r of the hubs as a previous step to know its value. The best of these m solutions, s_δ, is saved and the remaining $m-1$ solutions are discarded as possible steps of this path. Replace s^* by s_δ if $f(s_\delta) < f(s^*)$, where s^* is the best solution found in $\mathcal{P}(s,t)$ so far. This procedure is repeated while $|\Delta(s,t)| \geq 2$, and returns s^* as the output. Note that $f(s^*)$ is not necessarily better than $f(s)$ or $f(t)$. All the generated solutions s^* are stored in a set called *Pool*.

3.4.5 The reference set update method

The reference set update method (RSUM) is associated with each application of the SGM. The update operation consists of maintaining a record of the β best solutions found so far by the procedure. The issues related to this updating function are straightforward: All the solutions in *RefSet* that are worse than those in the current *Pool* will be replaced by these ones, with the aim of keeping in *RefSet* the β best and most different solutions found so far. Let $s \in Pool$ and $w \in RefSet$ such that $f(s) < f(w)$. In this case, s will replace w if s is different from all other solutions in *RefSet*.

3.4.6 The improvement method

As mentioned in Section 3.4, three optimization subproblems arise when solving the UrApHMP. Since we solve the routing subproblem optimally, we propose two improvement procedures based on local search strategies for the other two subproblems: LS_H for the hub selection, and LS_A for the terminal allocations. Both are based on the local search procedures proposed in Section 3.3.3.

LS_H implements a classical exchange procedure in which a hub h_i is removed from H, and a non-hub node $h'_i \in N \setminus H$ replaces h_i, thus obtaining $H' = \{h_1, h_2, \ldots, h'_i, \ldots, h_p\}$. We have described in Section 3.3.3 a mechanism to determine the order of exploration of the hubs in this procedure, but we have empirically checked that LS_H performs better in the scatter search scheme without this mechanism. Note that any change in H affects the other components in s. Specifically, when hub h_i is replaced by h'_i, we need to re-evaluate the hub assignment of at least all the vertices assigned to h_i. Moreover, the routes for the traffics are not necessarily optimal for the new set of hubs H'. Hence, since the solution structure changes that much, we evaluate all the routes from scratch. As a classical local search procedure, LS_H performs moves as long as the cost improves. We have implemented here the so-called *first strategy*, in which the first improving move is performed, instead of scanning the entire neighborhood to determine the best move.

The local search procedure LS_A is similar to LS_H, but it considers changes in the assignment of terminals to hubs. In particular, for a node i with $H^i = \{h_{i1}, \ldots, h_{ia}, \ldots, h_{ir}\}$, this procedure exchanges an assigned hub with a non-assigned one. In mathematical terms, we replace h_{ia} with $\bar{h}_{ia} \in H \setminus H^i$, thus obtaining $\bar{H}^i = \{h_{i1}, \ldots, \bar{h}_{ia}, \ldots, h_{ir}\}$.

3.4.7 Computational experiments

This section describes the computational experiments performed to test the efficiency of the scatter search with path-relinking method we propose to solve the UrApHMP. The procedure has been implemented in C and the results reported in this section have been obtained with an Intel core i7–3770 at 3.40GHz and 16GB of RAM, under Ubuntu 14.04 GNU/Linux – 64 bits operating system. The metrics that we use to measure the performance of the algorithms in a particular experiment are:

- Dev: Average percentage deviation with respect to the best solution found (or from the optimal solution, if available).
- # Best: Number of best solutions found.
- CPU: Average computing time in seconds.

We use the same benchmark of instances that has been proposed for the GRASP method of Section 3.3.5 from three well-known data sets: CAB, AP, and USA423.

The experiments are divided into two main blocks. The first block, described in Section 3.4.7.1, is devoted to study the behavior of the components of the solution procedure, as well as to determine the best values for the search parameters. The second block of experiments, in Section 3.4.7.2, has the goal of comparing our procedure with the best published methods.

3.4.7.1 Scientific testing

The first set of experiments to calibrate our method is performed on a subset of 47 instances: five instances from the CAB set with $n = 25$, and 42 instances with $40 \leq n \leq 200$ from the AP set. We refer to these 47 instances as the *training set* and to the remaining instances as the *testing set*.

The values of π, q and φ: We first study the values for the parameters used in the diversification generator method: π (that determines how many solutions will be constructed in *Pop*), q (that defines the size of the RCL in the DGM1–DGM6 methods), and φ (that determines the number of elements in the evaluation for the selection of hubs).

First, we have given parameter π two possible values: 100 and 150. For each instance of the training set, we have constructed 0.15π solutions with each of the methods DGM1 to DGM6, to obtain a 90% of the initial solutions generated. The remaining solutions (up to π) are obtained with DGM7, the random generator. In order to evaluate the

capacity of the DGM methods without taking into account the effect of the combination and improvement methods, this experiment only considers the constructive phase, not performing any of the subsequent elements of the scatter search procedure. The results on the training set are shown in Table 3.12.

As expected (see Table 3.12), the best solutions in terms of quality are obtained with $\pi = 150$. In this case, the algorithm obtains an average percentage deviation of 1.4% and 31 best known solutions. The CPU time for $\pi = 150$ is still reasonable, with virtually no difference in small and medium instances. Although for the large instances, the CPU values are slightly larger, we consider that the enhancement of the results worth the CPU effort, so we set $\pi = 150$ in the rest of the experiments.

Table 3.12. Calibration of π for the DGM of SS.

		Dev (%)		**# Best**		**CPU**	
Size	**# inst**	**100**	**150**	**100**	**150**	**100**	**150**
s (small)	8	2.1	0.0	2	8	0.0	0.0
m (medium)	27	2.2	0.6	8	19	0.2	0.3
l (large)	12	2.0	4.1	8	4	1.2	1.8
Summary	47	2.1	1.4	18	31	0.4	0.6

In the second experiment, we study the value of the parameter q that defines the RCL size. To do this we have considered $q = \min\{\frac{n}{2}, \omega p\}$, where ω is another parameter whose impact is studied next. For each instance of the training set, we have constructed $\frac{1}{6}\pi$ solutions with each DGM1 to DGM6 method but not with DGM7, because it is a totally random procedure. As in the previous experiment, we have only considered the constructive phase. The results for $\omega \in \{2, 3, 4, 5\}$ are presented in Table 3.13. Note that the best solutions in terms of quality are obtained with $\omega = 5$. With this value, we have obtained an average percentage deviation of 1.4% and 26 best known solutions. In order to compare the results, we have performed the well-known non-parametric Friedman test. The resulting probability value of 0.000006981 indicates that the compared values come from different methods. Moreover, we do not appreciate differences in the CPU times among the different values of ω, so we set $\omega = 5$ from now on.

Now we study the value of φ, the number of terminals appearing in the computation of $g(h)$. This number is computed as $\varphi = \lfloor \lambda \frac{n}{p} \rfloor$, where $\lambda \in \{1.0, 1.2, 1.5, 1.7, 2.0\}$. Again, we have constructed $\frac{1}{6}\pi$ solutions with methods DGM1 to DGM6 for each instance of the training set, using only the constructive phase of SS. The results for the different values of λ are shown in Table 3.14. It seems there is not a clear value of λ outperforming the others, hence, we have compared the results using the Friedman test. The resulting probability value of 0.1461 confirms that the compared values do not present significant differences. Considering that the best average deviation was found for $\lambda = 1$, $\varphi = \lfloor \frac{n}{p} \rfloor$ is the value we have finally chosen for the rest of the experiments.

Table 3.13. Calibration of ω for the DGM of SS.

		Dev (%)				# Best				CPU			
Size	**# inst**	**2**	**3**	**4**	**5**	**2**	**3**	**4**	**5**	**2**	**3**	**4**	**5**
s	8	2.1	1.7	1.8	0.6	5	1	3	4	0.0	0.0	0.0	0.0
m	27	7.4	5.0	3.6	1.3	3	4	4	16	0.2	0.2	0.2	0.2
l	12	8.9	3.3	3.4	2.4	1	2	3	6	1.1	1.1	1.2	1.2
Summary	47	6.9	4.0	3.2	1.4	9	7	10	26	0.4	0.4	0.4	0.4

Table 3.14. Calibration of φ through λ for the DGM of SS.

		Dev (%)					# Best					CPU				
Size	**# inst**	**1**	**1.2**	**1.5**	**1.7**	**2**	**1**	**1.2**	**1.5**	**1.7**	**2**	**1**	**1.2**	**1.5**	**1.7**	**2**
s	8	1.4	3.1	3.1	3.1	3.0	4	1	0	3	1	0.0	0.0	0.0	0.0	0.0
m	27	3.1	4.3	3.8	2.9	2.6	7	3	3	9	7	0.2	0.2	0.2	0.2	0.2
l	12	3.0	1.5	5.0	4.5	3.0	3	3	1	4	4	1.3	1.3	1.3	1.3	1.3
Summary	47	2.7	3.4	4.0	3.3	2.8	14	7	4	16	12	0.5	0.4	0.4	0.4	0.4

The value of β: In order to study the size of *RefSet* (β), we have constructed 0.15π solutions with DGM1 to DGM6 for each instance of the training set. The remaining solutions up to π are obtained with DGM7. In this experiment, we include in the algorithm all elements of scatter search except the local searches, to evaluate the power of the combination method (SCM) without the effects of the improvement procedures. As it can be seen in Table 3.15, the higher the value of β, the better are the results. However, this obviously implies higher computing times. We have compared the results using the Friedman test for all the values of β tested. The resulting probability value of 0.0000 indicates that the results obtained for the different values of β are significantly different. An important issue is to know if a similar deviation to the one obtained with $\beta = 10$ can be obtained in shorter times with a smaller size of β by using the improvement methods. This issue is the subject of the next experiment.

Table 3.15. Calibration of β for the DGM of SS.

		Dev (%)						# Best						CPU					
Size	**# inst**	**5**	**6**	**7**	**8**	**9**	**10**	**5**	**6**	**7**	**8**	**9**	**10**	**5**	**6**	**7**	**8**	**9**	**10**
s	8	1.0	0.0	0.0	0.0	0.0	0.0	5	8	8	8	8	8	0.1	0.1	0.2	0.2	0.4	0.4
m	27	3.7	2.0	1.3	0.5	0.4	0.1	3	6	7	8	13	20	1.2	2.1	3.1	4.5	6.2	8.1
l	12	4.4	1.9	1.9	0.3	0.2	0.1	0	2	2	4	8	7	4.9	8.0	12.1	15.7	23.3	29.0
Summary	47	3.4	1.6	1.2	0.4	0.3	0.1	8	16	17	20	29	35	2.0	3.3	4.9	6.6	9.6	12.2

The effect of local searches: Now we study the effect of the improvement procedures described in Section 3.4.6. Recall that two local searches have been proposed, one for

the hub selection (LS_H) and another for the terminal allocations (LS_A).

The standard SS design specifies to apply the improvement method to all the solutions resulting from the combination method. Some previous works have proposed a selective implementation of the local search procedures, reducing their application to only the best solutions obtained from the combination method in each global iteration. Since, due to the combinatorial nature of this problem, the local searches we propose are quite time consuming,we go a step further and limit the application of the improvement method to only the best solutions across all the global iterations. Specifically we have designed our procedure in such a way it applies LS_H and/or LS_A only to the solutions of *RefSet* just before the end of the algorithm. To evaluate the effect of the proposed local searches, we have studied the following four variants:

A: LS_H and LS_A are applied to all the solutions.

B: LS_H and LS_A are applied to the best solution only.

C: LS_A is applied to all the solutions.

D: LS_A is applied to the best solution only.

In what follows we compare each variant above for each value of β (from 5 to 10). Table 3.16 shows the results obtained on the 47 instances of the training set. Variants C and D exhibit the worst results in terms of the number of best solutions found and the deviation with respect to the best solutions found in this experiment. The results

Table 3.16. Computational results obtained with the four variants for the local search procedures.

			Dev (%)						# Best						CPU					
Var	**Size**	**# inst**	**5**	**6**	**7**	**8**	**9**	**10**	**5**	**6**	**7**	**8**	**9**	**10**	**5**	**6**	**7**	**8**	**9**	**10**
A	*s*	8	0.0	0.0	0.0	0.0	0.0	0.0	7	8	8	8	8	8	0.2	0.2	0.3	0.4	0.5	0.6
	m	27	0.1	0.0	0.0	0.1	0.1	0.0	21	22	24	18	21	24	4.6	6.0	7.4	9.1	11.5	14.0
	l	12	0.0	0.0	0.2	0.0	0.0	0.1	8	10	10	11	10	7	33.3	43.7	51.8	64.8	81.4	91.3
	Summary	47	0.1	0.0	0.1	0.1	0.1	0.0	36	40	42	37	39	39	11.2	14.6	17.5	21.8	27.5	31.4
B	*s*	8	0.1	0.1	0.1	0.1	0.1	0.1	7	7	7	7	7	7	0.1	0.1	0.2	0.2	0.3	0.4
	m	27	0.2	0.2	0.2	0.3	0.2	0.0	14	16	18	12	13	19	2.0	2.7	3.9	5.1	6.7	8.3
	l	12	0.1	0.1	0.4	0.2	0.1	0.2	6	5	5	3	6	5	10.8	13.8	17.7	25.5	34.4	40.7
	Summary	47	0.2	0.2	0.2	0.2	0.1	0.1	27	28	30	22	26	31	3.9	5.1	6.8	9.4	12.7	15.2
C	*s*	8	3.0	2.7	2.7	2.4	2.4	2.4	0	0	0	0	0	0	0.1	0.1	0.2	0.3	0.4	0.5
	m	27	3.6	2.6	2.3	1.7	1.5	1.6	0	0	0	2	3	2	2.5	3.3	4.7	6.1	8.0	9.8
	l	12	7.6	6.7	5.7	5.2	4.9	4.6	0	0	0	0	0	0	16.1	22.9	27.3	37.8	52.1	61.7
	Summary	47	4.5	3.7	3.3	2.7	2.5	2.5	0	0	0	2	3	2	5.6	7.8	9.7	13.2	18.0	21.4
D	*s*	8	3.0	2.7	2.7	2.4	2.4	2.4	0	0	0	0	0	0	0.1	0.1	0.2	0.2	0.3	0.4
	m	27	3.6	2.6	2.3	1.7	1.5	1.6	0	0	0	2	3	2	1.6	2.3	3.4	4.7	6.4	8.0
	l	12	7.6	6.7	5.7	5.2	4.9	4.6	0	0	0	0	0	0	7.6	11.0	14.5	21.7	30.9	37.7
	Summary	47	4.5	3.7	3.3	2.7	2.5	2.5	0	0	0	2	3	2	2.9	4.1	5.7	8.3	11.6	14.3

clearly indicate that applying LS_H to the solutions in *RefSet* substantially improves their value (from 2.5% to 4.5% on average). Variant A exhibits a larger number of best solutions found compared to variant B, what confirms that applying LS_H and LS_A to

all the solutions of *RefSet* is useful to match the best known solutions. Nevertheless, note that this exhaustive application of the local search procedures to all the solutions in *RefSet* results in much higher CPU times (from 11.15 to 31.45 seconds in variant A versus 3.94 to 15.24 in variant B). Despite the fact that variant B is not able to get some of the best known solutions, the average deviation is very small (from 0.1% to 0.2%). This indicates that, for short computing times, variant B obtains very good solutions in terms of average deviation of the cost and, hence, is also a very good option. Both variants, A and B, are the ones chosen to compare in Section 3.4.7.2 the SS procedure with the previously proposed GRASP method of Section 3.3 for the UrApHMP.

Regarding the β parameter, and considering only variants A and B, it can be observed that the best results in terms of Dev are obtained with values $\beta = 6$ and $\beta = 10$. In order to compare both sets of results, we have performed two well-known non-parametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The Wilcoxon test answers the question: Do the two samples (in our case, the solutions obtained with $\beta = 6$ and $\beta = 10$) represent two different populations? The resulting probability value of 0.50 indicates that there are not statistical differences, meaning that the compared values do not come from different methods. The Sign test computes the number of instances on which an algorithm beats the other one. The resulting probability value of 1.0 corroborates the previous result. As the CPU effort is clearly lower with $\beta = 6$ than with $\beta = 10$, we select $\beta = 6$ from now on.

3.4.7.2 Competitive testing

After the calibration process described before, we now compare the performance of our SS algorithm versus the GRASP procedure proposed in Section 3.3, which as far as we know is the best heuristic algorithm for the UrApHMP. All the methods under comparison are run in the same computer.

As a summary, our SS with PR procedure is set as follows: we generate 150 initial solutions; the size of the RCL when constructing solutions with methods DGM1 to DGM6 is defined by $\min\{\frac{n}{2}, 5p\}$; $\lfloor \frac{n}{p} \rfloor$ terminals are considered for the evaluation of each hub candidate; the size of *RefSet* is set to six solutions; and both local search procedures LS_H and LS_A are applied to the solutions of *RefSet*, using variants A and B, just before the end of the algorithm.

The results with the two versions of the algorithm corresponding to variants A and B, denoted SS^A and SS^B respectively, are summarized in Table 3.17. This table clearly shows that, except in the small size instances, our SS procedures outperform the previously proposed GRASP method. In particular, although SS^A and SS^B match 16 out of 30 small size instances while GRASP is able to match 25, and their deviation is 0.10% versus 0.06% in GRASP, the CPU time used by the SS methods is much shorter. Regarding the medium size instances, we can observe that the behavior of SS^B and GRASP are similar, although the former uses 1/5 of the GRASP computing time. In these instances, SS^A performs better than GRASP (108 best solutions found and Dev 0.02% versus 79 best and Dev 0.18%) using less than 1/2 of the GRASP computing time. Detailed results for each instance can be found in Tables 3.18, 3.19, 3.20, and 3.21. Finally, in what

refers to the large size instances, both SS versions clearly outperform GRASP in terms of Dev (0.01% and 0.07% versus 0.23% on average). Moreover, SS^A is able to find a larger number of best solutions (40 against 26 of SS^B and 29 of GRASP, respectively), although at a bigger computing time. Detailed results for each instance can be found in Tables 3.22 and 3.23.

Table 3.17. Computational results on the CAB and AP instances

			Dev (%)			# Best			CPU		
Size	n	**# inst**	**SS^A**	**SS^B**	**GRASP**	**SS^A**	**SS^B**	**GRASP**	**SS^A**	**SS^B**	**GRASP**
	25	18	0.1	0.1	0.1	11	11	15	0.1	0.1	0.4
s	40	6	0.1	0.1	0.0	1	1	5	0.3	0.2	3.2
	50	6	0.0	0.0	0.0	4	4	5	0.6	0.3	9.4
	summary	30	0.1	0.1	0.1	16	16	25	0.2	0.1	2.8
	60	21	0.0	0.2	0.4	19	11	5	2.7	1.4	4.6
	65	19	0.0	0.5	0.3	12	7	10	3.3	1.6	6.4
	70	19	0.0	0.2	0.3	14	10	15	4.2	2.0	11.0
	75	18	0.0	0.2	0.0	9	9	15	4.5	2.1	9.2
m	80	18	0.0	0.3	0.1	13	8	8	6.5	3.0	14.9
	85	18	0.0	0.1	0.1	13	12	9	6.7	3.0	15.6
	90	21	0.0	0.1	0.1	15	12	8	8.0	3.6	22.9
	95	21	0.0	0.0	0.1	13	10	9	9.8	4.1	24.3
	summary	155	0.0	0.2	0.2	108	79	79	5.8	2.6	13.8
	100	21	0.0	0.0	0.1	13	10	9	11.7	4.9	4.8
l	150	20	0.0	0.0	0.3	13	10	12	40.5	13.7	20.6
	200	21	0.0	0.2	0.2	14	6	8	86.1	27.5	58.9
	summary	62	0.0	0.1	0.2	40	26	29	46.2	15.4	28.2

In order to compare the results obtained with SS^A, SS^B, and GRASP from a statistical point of view, we have performed first the non-parametric Friedman test. The resulting probability value of 0.000 indicates that the results are significantly different. The ranks values produced by this test are 1.77 for SS^A, 2.09 for GRASP, and 2.15 for SS^B. Then, we have performed the non-parametric Wilcoxon and Sign tests for pairwise comparisons. When comparing SS^A with SS^B and SS^A with GRASP, the resulting probability values of 0.000 indicate that the compared results come from different methods. When comparing SS^B with GRASP, the resulting probability of 0.53 indicates that there are no significant differences between the two methods regarding the results obtained.

To complement this analysis, we have carried out a final comparison on very large size instances. Table 3.24 shows the result of SS^A, SS^B, and GRASP on the USA423 instances described in Section 3.3.5.1. Procedures SS^A and SS^B have been run with the parameters specified at the beginning of this section, while GRASP parameters' values are specified in Section 3.3.5.2. As shown in this table, SS^A and GRASP are run for the same running time. This table confirms the superiority of SS^A compared to GRASP, since the former exhibits an average percent deviation of 0.3% and GRASP only achieves a 4.5%. The resulting probability of 0.01 of the non-parametric pairwise Wilcoxon test confirms this conclusion. Additionally, SS^B is very competitive since it obtains slightly

Table 3.18. Computational results on medium-sized hard instances

						DEV (%)			CPU			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP60	3	2	3	0.75	2	0.0	0.0	1.3	0.41	0.17	0.30	157493.38
AP60	4	2	3	0.75	2	0.0	0.3	1.8	0.62	0.27	0.48	142412.35
AP60	4	3	3	0.75	2	0.0	1.7	0.1	0.87	0.39	1.24	142268.41
AP60	5	2	3	0.75	2	0.0	0.2	0.0	0.90	0.44	1.06	130186.83
AP60	5	3	3	0.75	2	0.0	0.0	0.2	1.19	0.57	2.18	129680.76
AP60	5	4	3	0.75	2	0.0	0.0	0.1	1.52	0.72	2.86	129652.93
AP60	6	2	3	0.75	2	0.0	0.0	0.0	1.38	0.70	1.52	122365.30
AP60	6	3	3	0.75	2	0.0	0.3	0.3	1.93	0.96	1.63	122066.63
AP60	6	4	3	0.75	2	0.0	0.3	0.3	2.54	1.22	4.51	121979.40
AP60	6	5	3	0.75	2	0.0	0.3	0.0	3.11	1.51	4.55	121979.40
AP60	7	2	3	0.75	2	0.0	0.0	0.1	1.57	0.96	2.76	116380.78
AP60	7	3	3	0.75	2	0.0	0.2	0.5	2.45	1.24	3.22	116003.39
AP60	7	4	3	0.75	2	0.0	0.1	0.1	3.21	1.43	5.09	115959.96
AP60	7	5	3	0.75	2	0.0	0.1	0.1	4.20	1.89	7.31	115951.78
AP60	7	6	3	0.75	2	0.0	0.0	0.1	4.64	2.10	7.75	115951.78
AP60	8	2	3	0.75	2	0.0	0.0	1.1	2.05	1.30	2.98	110041.02
AP60	8	3	3	0.75	2	0.0	0.0	0.4	2.96	1.49	6.40	109888.11
AP60	8	4	3	0.75	2	0.0	0.0	0.5	3.65	2.06	6.82	109668.92
AP60	8	5	3	0.75	2	0.0	0.0	1.2	4.61	2.53	6.53	109651.38
AP60	8	6	3	0.75	2	0.0	0.0	0.0	6.73	3.62	11.39	109651.38
AP60	8	7	3	0.75	2	0.0	0.0	0.0	6.71	3.71	15.80	109651.38
AP65	3	2	3	0.75	2	0.0	2.1	0.0	0.43	0.21	0.69	157509.77
AP65	4	2	3	0.75	2	0.1	0.7	0.0	0.82	0.40	0.82	142702.65
AP65	4	3	3	0.75	2	0.0	0.0	0.3	0.94	0.43	1.18	142632.42
AP65	5	2	3	0.75	2	0.0	0.0	0.0	1.09	0.46	1.51	130848.81
AP65	5	3	3	0.75	2	0.0	0.0	0.0	1.47	0.57	4.16	130127.46
AP65	5	4	3	0.75	2	0.0	0.0	0.0	1.88	0.73	3.03	130094.52
AP65	6	2	3	0.75	2	0.0	0.0	1.7	1.75	0.92	1.81	123292.08
AP65	6	3	3	0.75	2	0.0	1.5	0.0	2.14	1.02	4.09	122959.75
AP65	6	4	3	0.75	2	0.0	1.4	0.1	3.09	1.31	4.40	122871.81
AP65	6	5	3	0.75	2	0.0	1.4	0.1	3.79	1.64	6.62	122871.81
AP65	7	2	3	0.75	2	0.1	0.1	0.0	2.58	1.18	2.76	116951.08
AP65	7	3	3	0.75	2	0.2	0.2	0.0	3.17	1.55	6.52	116665.46
AP65	7	4	3	0.75	2	0.0	0.0	0.8	4.21	1.96	8.41	116601.77
AP65	7	5	3	0.75	2	0.0	0.0	0.9	5.53	2.45	9.19	116590.58
AP65	7	6	3	0.75	2	0.0	0.0	0.9	6.48	2.99	14.49	116590.58
AP65	8	2	3	0.75	2	0.0	0.0	0.0	2.83	1.51	3.85	111622.72
AP65	8	4	3	0.75	2	0.0	1.0	0.0	4.43	2.41	12.57	111239.47
AP65	8	6	3	0.75	2	0.0	0.0	0.1	7.59	4.08	13.18	111231.25
AP65	8	7	3	0.75	2	0.0	1.0	0.1	9.41	4.30	22.31	111231.25

Table 3.19. Computational results on medium-sized hard instances (continuation)

						DEV (%)			CPU			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP70	3	2	3	0.75	2	0.0	0.0	2.0	0.62	0.26	0.56	158038.30
AP70	4	2	3	0.75	2	0.1	0.1	0.0	0.99	0.43	1.95	142720.05
AP70	4	3	3	0.75	2	0.0	0.0	1.0	1.22	0.53	1.70	142626.15
AP70	5	2	3	0.75	2	0.2	0.2	0.0	1.58	0.65	2.51	132562.81
AP70	5	3	3	0.75	2	0.0	0.0	0.0	1.96	0.76	4.56	132100.10
AP70	5	4	3	0.75	2	0.0	0.0	0.0	2.58	0.97	6.15	132055.96
AP70	5	3	3	0.75	2	0.0	0.0	0.0	3.40	1.59	6.29	123645.87
AP70	5	4	3	0.75	2	0.0	0.0	0.0	3.93	2.03	5.49	123601.74
AP70	5	5	3	0.75	2	0.0	0.0	0.0	4.82	2.54	7.18	123601.74
AP70	7	2	3	0.75	2	0.2	0.2	0.0	2.42	1.35	6.07	117996.74
AP70	7	3	3	0.75	2	0.0	0.0	0.0	3.32	1.75	11.29	117525.65
AP70	7	4	3	0.75	2	0.0	0.0	0.0	4.54	2.06	11.29	117485.79
AP70	7	5	3	0.75	2	0.0	0.0	0.0	5.65	2.71	12.06	117485.26
AP70	7	6	3	0.75	2	0.0	0.0	0.0	6.36	3.25	15.83	117485.26
AP70	7	3	3	0.75	2	0.0	0.7	0.0	4.19	1.94	8.58	112134.88
AP70	7	4	3	0.75	2	0.0	0.6	0.6	5.78	2.63	16.02	112098.09
AP70	7	5	3	0.75	2	0.0	0.6	0.0	7.23	3.11	16.82	112082.12
AP70	7	6	3	0.75	2	0.0	0.6	0.0	8.96	4.23	24.68	112082.12
AP70	7	7	3	0.75	2	0.0	0.6	1.1	10.24	5.04	49.23	112082.12
AP75	3	2	3	0.75	2	0.1	0.1	0.0	0.76	0.31	0.58	158171.28
AP75	4	2	3	0.75	2	0.0	0.0	0.0	1.45	0.58	2.14	142854.97
AP75	4	3	3	0.75	2	0.1	1.1	0.0	1.75	0.66	2.14	142668.41
AP75	5	2	3	0.75	2	0.0	0.0	0.0	2.00	0.79	3.37	132822.87
AP75	5	3	3	0.75	2	0.0	0.0	0.0	2.39	0.92	4.66	132387.75
AP75	5	4	3	0.75	2	0.0	0.0	0.0	3.10	1.20	7.07	132365.64
AP75	6	2	3	0.75	2	0.1	0.1	0.0	2.43	1.30	3.95	125657.15
AP75	6	3	3	0.75	2	0.0	0.0	0.0	3.32	1.59	7.01	125224.59
AP75	6	4	3	0.75	2	0.0	0.0	0.5	4.66	2.04	7.63	125184.65
AP75	6	5	3	0.75	2	0.0	0.0	0.1	5.82	2.55	14.15	125184.65
AP75	7	2	3	0.75	2	0.0	0.8	0.0	3.47	1.67	7.39	119237.88
AP75	7	3	3	0.75	2	0.0	0.0	0.0	5.08	2.28	10.08	118808.16
AP75	7	4	3	0.75	2	0.0	0.0	0.0	6.25	2.91	15.12	118786.38
AP75	7	5	3	0.75	2	0.0	0.0	0.0	7.95	3.65	13.32	118786.38
AP75	7	6	3	0.75	2	0.0	0.0	0.0	9.32	4.47	15.73	118786.38
AP75	8	2	3	0.75	2	0.0	0.0	0.2	3.68	2.19	4.18	114690.98
AP75	8	4	3	0.75	2	0.0	0.6	0.0	6.60	3.28	22.15	113400.50
AP75	8	7	3	0.75	2	0.0	0.0	0.0	11.16	5.56	24.54	114086.67

Table 3.20. Computational results on medium-sized hard instances (continuation)

						DEV (%)			CPU			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP80	3	2	3	0.75	2	0.1	0.1	0.0	0.92	0.37	1.44	158202.73
AP80	3	3	3	0.75	2	0.0	0.0	0.0	1.95	0.70	2.12	143102.38
AP80	5	2	3	0.75	2	0.0	0.0	0.0	2.08	0.89	3.17	132915.50
AP80	5	3	3	0.75	2	0.0	0.0	0.0	2.82	1.14	4.85	132446.27
AP80	5	4	3	0.75	2	0.0	0.0	0.0	3.63	1.44	6.71	132424.08
AP80	6	2	3	0.75	2	0.1	0.1	0.0	3.25	1.60	5.83	125743.83
AP80	6	3	3	0.75	2	0.0	0.0	0.0	4.27	2.03	8.36	125284.10
AP80	6	4	3	0.75	2	0.0	0.0	0.1	5.66	2.64	11.12	125258.38
AP80	6	5	3	0.75	2	0.0	0.0	0.1	7.18	3.31	9.75	125258.38
AP80	7	2	3	0.75	2	0.0	0.6	0.6	4.02	2.06	4.35	119597.86
AP80	7	3	3	0.75	2	0.0	0.0	0.0	5.87	2.61	12.48	119132.73
AP80	7	4	3	0.75	2	0.0	0.0	0.8	7.61	3.52	17.05	119112.52
AP80	7	5	3	0.75	2	0.0	0.0	0.1	9.90	4.55	25.55	119105.55
AP80	7	6	3	0.75	2	0.0	0.0	0.1	11.76	5.43	21.25	119105.55
AP80	7	3	3	0.75	2	0.0	1.9	0.6	6.27	3.35	19.47	113787.42
AP80	7	5	3	0.75	2	0.0	1.2	0.1	10.58	5.48	26.02	114404.95
AP80	7	6	3	0.75	2	0.0	1.2	0.4	13.59	6.03	32.28	114404.95
AP80	7	7	3	0.75	2	0.0	1.2	0.0	15.65	7.34	55.93	114404.95
AP85	3	2	3	0.75	2	0.0	0.0	0.0	1.35	0.45	1.62	158274.33
AP85	4	2	3	0.75	2	0.0	0.1	0.0	1.76	0.73	3.11	142919.25
AP85	4	3	3	0.75	2	0.0	0.0	0.0	2.36	0.95	3.96	142822.33
AP85	5	2	3	0.75	2	0.2	0.2	0.0	2.77	1.34	6.08	133552.92
AP85	5	3	3	0.75	2	0.0	0.0	0.0	3.71	1.67	6.13	133110.33
AP85	5	4	3	0.75	2	0.0	0.0	0.0	4.63	2.16	10.08	133081.65
AP85	6	2	3	0.75	2	0.0	0.0	0.0	4.86	1.72	5.82	126462.13
AP85	6	3	3	0.75	2	0.0	0.0	0.4	5.18	2.33	7.05	125925.59
AP85	6	4	3	0.75	2	0.0	0.0	0.1	6.20	2.22	18.55	125849.01
AP85	6	5	3	0.75	2	0.0	0.0	0.1	8.59	3.74	18.91	125849.01
AP85	7	2	3	0.75	2	0.0	0.3	0.0	4.55	2.32	6.68	120735.00
AP85	7	3	3	0.75	2	0.1	0.1	0.0	6.02	2.80	15.98	119872.79
AP85	7	4	3	0.75	2	0.0	0.0	0.6	8.04	3.56	15.41	119852.39
AP85	7	5	3	0.75	2	0.0	0.0	0.0	10.17	4.47	31.81	119837.13
AP85	7	6	3	0.75	2	0.0	0.0	0.1	12.01	5.40	38.39	119837.13
AP85	8	2	3	0.75	2	0.2	0.7	0.0	5.77	3.02	12.15	114508.00
AP85	8	5	3	0.75	2	0.0	0.0	0.1	14.49	6.68	26.24	114966.55
AP85	8	6	3	0.75	2	0.0	0.0	0.1	17.65	8.30	53.54	114966.55

Table 3.21. Computational results on medium-sized hard instances (continuation)

						DEV (%)			CPU			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP90	3	2	3	0.75	2	0.0	0.0	0.0	1.28	0.44	1.94	157612.16
AP90	4	2	3	0.75	2	0.1	0.1	0.0	1.83	0.76	2.92	142465.46
AP90	4	3	3	0.75	2	0.0	0.0	0.0	2.65	1.07	7.77	142342.85
AP90	5	2	3	0.75	2	0.1	0.1	0.0	3.79	1.47	5.43	132771.23
AP90	5	3	3	0.75	2	0.0	0.0	0.0	4.35	1.79	8.05	132486.47
AP90	5	4	3	0.75	2	0.0	0.0	0.0	5.62	2.28	11.28	132422.07
AP90	6	2	3	0.75	2	0.1	0.1	0.0	3.86	1.74	9.31	125465.09
AP90	6	3	3	0.75	2	0.0	0.0	0.1	5.47	2.25	11.34	125176.85
AP90	6	4	3	0.75	2	0.0	0.0	0.1	6.51	2.65	18.00	125055.28
AP90	6	5	3	0.75	2	0.0	0.0	0.1	7.90	3.27	17.64	125055.28
AP90	7	2	3	0.75	2	0.1	0.1	0.0	5.72	2.72	10.30	119666.91
AP90	7	3	3	0.75	2	0.0	0.0	0.4	7.55	3.51	16.32	119379.94
AP90	7	4	3	0.75	2	0.0	0.0	0.2	10.04	4.48	41.87	119198.08
AP90	7	5	3	0.75	2	0.0	0.0	0.1	12.39	5.28	36.33	119190.81
AP90	7	6	3	0.75	2	0.0	0.0	0.1	14.94	6.84	58.77	119190.81
AP90	8	2	3	0.75	2	0.0	0.0	0.0	6.07	3.44	10.55	114099.35
AP90	8	3	3	0.75	2	0.0	0.0	1.1	8.03	4.17	27.21	113872.57
AP90	8	4	3	0.75	2	0.0	0.0	0.2	11.06	5.07	19.20	113776.78
AP90	8	5	3	0.75	2	0.0	0.0	0.4	13.60	6.36	38.81	113732.39
AP90	8	6	3	0.75	2	0.0	0.0	0.1	16.91	7.47	47.03	113732.39
AP90	8	7	3	0.75	2	0.0	0.8	0.1	18.50	9.25	81.84	113732.39
AP95	3	2	3	0.75	2	0.0	0.0	0.0	1.67	0.53	1.50	157684.46
AP95	4	2	3	0.75	2	0.1	0.1	0.0	2.17	0.86	3.60	142538.31
AP95	4	3	3	0.75	2	0.1	0.1	0.0	2.55	1.06	5.49	142457.05
AP95	5	2	3	0.75	2	0.1	0.1	0.0	3.51	1.38	6.49	133014.42
AP95	5	3	3	0.75	2	0.0	0.0	0.0	4.79	1.76	14.01	132763.09
AP95	5	4	3	0.75	2	0.0	0.0	0.0	6.19	2.27	12.57	132686.94
AP95	6	2	3	0.75	2	0.0	0.0	0.0	5.28	1.99	6.35	125700.19
AP95	6	3	3	0.75	2	0.0	0.0	0.1	6.59	2.58	11.73	125435.39
AP95	6	4	3	0.75	2	0.0	0.0	0.1	8.63	3.22	24.37	125311.86
AP95	6	5	3	0.75	2	0.0	0.0	0.1	10.54	3.98	17.80	125311.86
AP95	7	2	3	0.75	2	0.0	0.1	0.0	6.38	3.23	11.90	119897.57
AP95	7	3	3	0.75	2	0.0	0.0	0.1	8.06	3.81	20.17	119628.97
AP95	7	4	3	0.75	2	0.0	0.2	0.0	10.96	4.70	30.11	119422.38
AP95	7	5	3	0.75	2	0.0	0.1	0.1	13.88	5.88	37.85	119422.38
AP95	7	6	3	0.75	2	0.0	0.1	0.1	16.18	7.11	36.05	119422.38
AP95	8	2	3	0.75	2	0.0	0.0	0.4	8.32	4.22	15.76	114236.69
AP95	8	3	3	0.75	2	0.0	0.0	0.0	10.01	4.52	25.82	114351.51
AP95	8	4	3	0.75	2	0.0	0.0	0.4	14.33	5.87	37.18	113900.72
AP95	8	5	3	0.75	2	0.0	0.0	0.0	18.42	7.34	53.85	113890.35
AP95	8	6	3	0.75	2	0.0	0.0	1.0	21.56	8.83	85.85	113868.30
AP95	8	7	3	0.75	2	0.0	0.0	0.0	25.07	10.85	51.21	113868.30

Table 3.22. Computational results on large-sized hard instances

						DEV (%)			CPU			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP100	3	2	3	0.75	2	0.0	0.0	0.0	1.99	0.60	0.45	158043.08
AP100	4	2	3	0.75	2	0.0	0.0	0.0	2.78	1.01	1.12	143208.40
AP100	4	3	3	0.75	2	0.0	0.0	0.0	4.00	1.41	1.21	143086.43
AP100	5	2	3	0.75	2	0.1	0.1	0.0	4.24	1.74	2.11	133815.35
AP100	5	3	3	0.75	2	0.0	0.0	0.0	5.73	2.21	1.53	133569.22
AP100	5	4	3	0.75	2	0.0	0.0	0.0	6.91	2.83	1.57	133483.00
AP100	6	2	3	0.75	2	0.1	0.1	0.0	6.30	2.45	3.36	126523.14
AP100	6	3	3	0.75	2	0.0	0.0	0.0	8.60	3.18	2.46	126228.60
AP100	6	4	3	0.75	2	0.0	0.0	0.1	11.44	4.11	5.26	126107.94
AP100	6	5	3	0.75	2	0.0	0.0	0.1	14.13	5.17	3.59	126107.94
AP100	7	2	3	0.75	2	0.1	0.1	0.0	7.60	3.40	4.14	120697.19
AP100	7	3	3	0.75	2	0.0	0.0	0.0	10.37	4.40	6.02	120471.47
AP100	7	4	3	0.75	2	0.0	0.2	0.2	13.67	5.55	6.62	120187.66
AP100	7	5	3	0.75	2	0.0	0.0	0.2	18.03	7.21	5.34	120164.59
AP100	7	6	3	0.75	2	0.0	0.1	0.1	19.49	9.52	8.97	120234.75
AP100	8	2	3	0.75	2	0.0	0.1	0.0	9.73	4.63	3.18	114709.50
AP100	8	3	3	0.75	2	0.1	0.1	0.0	11.67	5.23	6.27	114439.93
AP100	8	4	3	0.75	2	0.0	0.0	0.1	15.68	6.68	9.50	114315.28
AP100	8	5	3	0.75	2	0.0	0.0	1.2	20.19	8.42	7.40	114298.12
AP100	8	6	3	0.75	2	0.0	0.0	1.1	24.68	10.28	7.98	114296.13
AP100	8	7	3	0.75	2	0.0	0.0	0.0	27.87	12.15	12.93	114296.13
AP150	3	2	3	0.75	2	0.0	0.0	0.0	4.95	1.79	1.68	158742.05
AP150	4	2	3	0.75	2	0.1	0.1	0.0	8.66	3.33	3.57	143811.40
AP150	4	3	3	0.75	2	0.0	0.0	0.0	10.70	3.82	2.94	143696.46
AP150	5	2	3	0.75	2	0.0	0.0	1.6	13.06	5.02	6.93	134590.93
AP150	5	3	3	0.75	2	0.0	0.1	1.8	17.36	5.93	9.85	134053.76
AP150	5	4	3	0.75	2	0.0	0.0	0.0	20.77	6.89	9.59	134022.43
AP150	6	2	3	0.75	2	0.1	0.1	0.0	19.62	6.95	8.39	127219.49
AP150	6	3	3	0.75	2	0.0	0.0	0.0	25.01	8.73	14.03	126935.58
AP150	6	4	3	0.75	2	0.0	0.0	0.0	29.01	10.38	11.20	126871.13
AP150	6	5	3	0.75	2	0.0	0.0	0.0	39.97	13.43	16.45	126871.13
AP150	7	2	3	0.75	2	0.1	0.1	0.0	27.73	9.20	18.00	121297.48
AP150	7	3	3	0.75	2	0.0	0.0	0.2	33.58	11.11	26.64	121101.93
AP150	7	4	3	0.75	2	0.0	0.0	0.0	48.84	15.48	19.35	120922.63
AP150	7	5	3	0.75	2	0.0	0.0	0.3	58.01	17.32	21.78	120965.44
AP150	7	6	3	0.75	2	0.0	0.0	0.0	68.40	20.69	27.71	120965.44
AP150	8	2	3	0.75	2	0.0	0.0	1.0	33.95	13.09	20.70	115486.83
AP150	8	3	3	0.75	2	0.0	0.0	0.6	45.65	16.29	33.58	115609.81
AP150	8	5	3	0.75	2	0.0	0.0	0.0	83.73	28.37	43.13	115108.06
AP150	8	6	3	0.75	2	0.0	0.0	0.0	102.16	34.79	61.19	115105.52
AP150	8	7	3	0.75	2	0.0	0.0	0.7	118.18	41.23	55.18	115105.52

Table 3.23. Computational results on large-sized hard instances (continuation)

						DEV (%)			CPU			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP200	3	2	3	0.75	2	0.0	0.0	0.0	10.93	3.96	5.67	159987.41
AP200	4	2	3	0.75	2	0.0	0.0	0.0	24.46	7.55	11.99	144755.16
AP200	4	3	3	0.75	2	0.0	0.0	0.0	33.80	9.03	12.00	144611.12
AP200	5	2	3	0.75	2	0.0	0.0	0.0	36.78	10.49	21.50	137408.43
AP200	5	3	3	0.75	2	0.0	0.4	0.3	43.53	11.50	22.56	136914.54
AP200	5	4	3	0.75	2	0.0	0.2	0.2	64.65	16.60	33.38	136777.91
AP200	6	2	3	0.75	2	0.0	0.0	0.0	38.48	14.40	37.03	130235.76
AP200	6	3	3	0.75	2	0.0	1.2	0.0	50.76	16.11	44.05	129883.62
AP200	6	4	3	0.75	2	0.0	0.0	0.0	76.71	23.27	54.33	129817.47
AP200	6	5	3	0.75	2	0.0	0.0	1.2	93.25	28.60	30.80	129817.47
AP200	7	2	3	0.75	2	0.0	0.2	0.6	57.63	18.31	49.65	123989.21
AP200	7	3	3	0.75	2	0.1	0.2	0.0	78.94	23.21	56.82	123670.80
AP200	7	4	3	0.75	2	0.0	0.2	0.2	110.69	30.88	73.03	123661.35
AP200	7	5	3	0.75	2	0.0	0.2	0.2	139.14	38.47	89.25	123658.33
AP200	7	6	3	0.75	2	0.0	0.2	0.0	163.59	45.90	63.66	123658.33
AP200	8	2	3	0.75	2	0.0	0.1	0.5	89.36	33.12	61.56	118125.17
AP200	8	3	3	0.75	2	0.0	0.2	0.5	106.03	41.38	79.85	117828.62
AP200	8	4	3	0.75	2	0.0	0.0	0.0	104.03	36.42	136.11	117719.51
AP200	8	5	3	0.75	2	0.0	0.0	0.4	142.68	46.31	121.85	117709.98
AP200	8	6	3	0.75	2	0.0	0.0	0.6	162.28	56.19	118.79	117709.98
AP200	8	7	3	0.75	2	0.0	0.0	0.4	180.24	65.34	112.22	117709.98

better results than GRASP in shorter computing times.

As a final test, we compare the performance of SS^A and SS^B with the results of the evolutionary approach recently proposed by Milanovic [71] for the multiple allocation version of the p-hub median problem. The proposed SS procedures are applied on 40 AP instances of different sizes, where $r = p$. Table 3.25 shows, in each row, the size and the value of p on each instance, and, for each method, the value of the objective function, the average percentage deviation with respect to the best known solution, and the CPU time. Results for the evolutionary method are directly taken from [71], and therefore running times are only indicative and cannot be directly compared with the SS computing times.

Results in Table 3.25 show that SS obtains high-quality solutions for the multiple allocation version, although they are slightly worse than those obtained by the evolutionary approach. In particular, SS^A and SS^B exhibit an average percent deviation of 0.1%, while the evolutionary approach is able to obtain a 0.0% in similar running times. Additionally, SS^A matches 35 best known solutions, and SS^B matches 30 out of 40 instances. The evolutionary method is able to match all the best known solutions. It must be noted however, that SS is designed to obtain good solutions in all the variants of this problem (i.e., for the different values of r), while the evolutionary method is suited for one specific variant.

3.4.8 Concluding remarks

In this section, we have proposed a metaheuristic algorithm based on scatter search for the uncapacitated r-allocation p-hub median problem. This problem was introduced by Yaman [103] as a generalization of the classical single allocation and multiple allocation p-hub median problem. The proposed scatter search procedure incorporates several designs for the diversification generator method, a path-relinking procedure for combining solutions, and two local search procedures as the improvement method. The computational experiments on a large set of instances from the literature show that our algorithm is able to find high-quality solutions in short computing times, and outperforms the previously introduces GRASP procedure of Section 3.3.

It is worth mentioning that our scatter search design only applies the improvement method at the end of the search. This selective application reduces the CPU time without sacrifying the final solution quality, making our method competitive in both quality of solutions and speed.

Table 3.24. Computational results on the USA423 instances

					Solution value			Dev (%)			CPU		
p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**
3	2	0.1	0.07	0.09	33727412959	40576260062	43867093585	0.0	20.3	30.1	99.0	30.3	100
4	2	0.1	0.07	0.09	31555933763	31555933763	31540232352	0.0	0.0	0.0	169.1	47.0	170
4	3	0.1	0.07	0.09	31378516085	31378516085	41500538133	0.0	0.0	32.3	259.3	63.3	260
5	2	0.1	0.07	0.09	29361690398	29361690398	29428603263	0.0	0.0	0.2	227.4	73.9	230
5	3	0.1	0.07	0.09	29012500636	29012500636	29005134361	0.0	0.0	0.0	337.1	98.3	340
5	4	0.1	0.07	0.09	28993057993	28993057993	35768287695	0.0	0.0	23.4	443.5	145.5	445
6	2	0.1	0.07	0.09	28761333285	29648009936	29813468007	0.0	3.1	3.7	348.0	116.0	350
6	3	0.1	0.07	0.09	28140175764	28636803830	27999028069	0.5	2.3	0.0	474.8	118.9	475
6	4	0.1	0.07	0.09	27952573758	28449216973	27690657697	0.9	2.7	0.0	545.6	144.3	550
6	5	0.1	0.07	0.09	27937297018	27937297018	27688845102	0.9	0.9	0.0	660.6	212.5	665
7	2	0.1	0.07	0.09	28076360663	28076360663	30799812597	0.0	0.0	9.7	416.6	122.9	420
7	3	0.1	0.07	0.09	26692507210	27724315371	26751472180	0.0	3.9	0.2	561.4	179.4	565
7	4	0.1	0.07	0.09	26258768165	26258768165	27537936675	0.0	0.0	4.9	717.9	207.6	720
7	5	0.1	0.07	0.09	26255012774	26255012774	26989369645	0.0	0.0	2.8	944.1	247.9	945
7	6	0.1	0.07	0.09	26255010934	26255010934	26255010934	0.0	0.0	0.0	1115.1	309.1	1120
3	2	0.09	0.075	0.08	30176127832	36338055076	30176127832	0.0	20.4	0.0	112.1	34.2	115
4	2	0.09	0.075	0.08	28373652481	28373652481	30530137285	0.0	0.0	7.6	173.5	61.3	175
4	3	0.09	0.075	0.08	28125164296	28125164296	28125164296	0.0	0.0	0.0	297.7	70.3	300
5	2	0.09	0.075	0.08	26451114267	26451114267	26492913729	0.0	0.0	0.2	232.3	82.8	235
5	3	0.09	0.075	0.08	26088080203	26088080203	26079665626	0.0	0.0	0.0	360.6	111.8	365
5	4	0.09	0.075	0.08	26061806930	26061806930	27249791197	0.0	0.0	4.6	471.3	138.6	475
6	2	0.09	0.075	0.08	26525421601	26525421601	26016295685	2.0	2.0	0.0	318.7	118.7	320
6	3	0.09	0.075	0.08	24971043351	25449149342	25788838779	0.0	1.9	3.3	497.9	155.3	500
6	4	0.09	0.075	0.08	25265904571	25265904571	24828738291	1.8	1.8	0.0	631.6	203.9	635
6	5	0.09	0.075	0.08	25243965930	25243965930	25385565293	0.0	0.0	0.6	853.8	264.2	855
7	2	0.09	0.075	0.08	25906507439	29264746888	26592882079	0.0	13.0	2.6	413.9	130.5	415
7	3	0.09	0.075	0.08	24784471473	25578317735	24066130754	3.0	6.3	0.0	636.6	192.6	640
7	4	0.09	0.075	0.08	23586398612	24971605186	23619010159	0.0	5.9	0.1	765.5	220.9	770
7	5	0.09	0.075	0.08	23577554532	24905755621	24220406570	0.0	5.6	2.7	1085.47	326.35	1090
7	6	0.09	0.075	0.08	23577543900	24887707334	25017677542	0.0	5.6	6.1	1356.71	500.71	1360
								0.3	3.2	4.5	517.6	157.6	520.1

Table 3.25. Computational results on AP instances for the multiple allocation version

		Solution value			Dev (%)			CPU		
n	p	$\mathbf{SS}^A$	$\mathbf{SS}^B$	**Evo**	$\mathbf{SS}^A$	$\mathbf{SS}^B$	**Evo**	$\mathbf{SS}^A$	$\mathbf{SS}^B$	**Evo**
	2	173415.5	173415.5	173415.5	0.0	0.0	0.0	0.1	0.0	0.1
	3	155458.1	155458.1	155458.1	0.0	0.0	0.0	0.2	0.1	0.0
	4	140682.2	140682.2	140682.2	0.0	0.0	0.0	0.4	0.2	0.4
	5	130384.1	130384.1	130384.1	0.0	0.0	0.0	0.6	0.4	0.5
40	6	122170.2	122170.2	122170.2	0.0	0.0	0.0	1.1	0.6	5.9
	7	116035.9	116035.9	116035.9	0.0	0.0	0.0	1.5	0.9	7.1
	8	109971.1	109971.1	109971.1	0.0	0.0	0.0	2.1	1.5	8.4
	9	104211.5	104211.5	104211.5	0.0	0.0	0.0	2.6	1.6	9.6
	10	99451.8	99451.8	99451.8	0.0	0.0	0.0	4.2	2.6	12.5
	2	174390.6	174390.6	174390.6	0.0	0.0	0.0	0.1	0.1	0.1
	3	156014.6	156014.6	156014.5	0.0	0.0	0.0	0.3	0.1	0.3
	4	141154.3	141154.3	141154.3	0.0	0.0	0.0	0.7	0.3	0.6
	5	129414.2	129507.4	129414.2	0.0	0.1	0.0	1.0	0.5	0.8
50	6	121673.6	121673.6	121673.6	0.0	0.0	0.0	2.2	1.3	8.5
	7	115913.4	115913.4	115911.6	0.0	0.0	0.0	3.0	1.5	10.3
	8	111139.4	111139.4	109927.6	1.1	1.1	0.0	3.9	2.4	12.5
	9	104968.8	104968.8	104968.8	0.0	0.0	0.0	6.7	3.9	15.2
	10	100509.3	100645.4	100509.2	0.0	0.1	0.0	9.7	6.3	18.0
	2	176246.8	176246.8	176246.8	0.0	0.0	0.0	0.7	0.3	5.1
	3	157870.9	157870.9	157870.9	0.0	0.0	0.0	2.2	0.8	13.3
	4	143004.4	143086.4	143004.3	0.0	0.1	0.0	3.8	1.3	18.5
	5	133483.0	133483.0	133483.0	0.0	0.0	0.0	7.6	2.6	23.8
	6	126107.9	126107.9	126107.9	0.0	0.0	0.0	16.0	6.3	31.3
100	7	120164.6	120164.6	120164.6	0.0	0.0	0.0	24.5	10.9	41.3
	8	115144.7	115144.7	114295.9	0.7	0.7	0.0	25.7	14.4	57.0
	9	109449.1	109449.1	109449.1	0.0	0.0	0.0	38.7	20.2	68.7
	10	104800.8	104800.8	104794.3	0.0	0.0	0.0	71.6	45.3	87.2
	15	88882.5	89273.6	88882.5	0.0	0.4	0.0	226.9	148.0	167.1
	20	79453.7	79453.7	79191.6	0.3	0.3	0.0	564.9	385.1	233.9
	2	178094.0	178094.0	178094.0	0.0	0.0	0.0	5.0	1.9	44.2
	3	159725.1	159725.1	159725.1	0.0	0.0	0.0	13.8	5.0	73.9
	4	144508.2	144508.2	144508.2	0.0	0.0	0.0	36.1	11.9	95.5
	5	136761.8	136761.8	136761.8	0.0	0.0	0.0	56.2	16.8	156.2
	6	129556.5	130739.0	129556.5	0.0	0.9	0.0	124.7	24.6	185.7
200	7	123608.9	124132.2	123608.9	0.0	0.4	0.0	153.6	51.0	226.3
	8	117879.5	117879.5	117710.0	0.1	0.1	0.0	231.8	85.9	285.0
	9	112374.5	112374.5	112374.5	0.0	0.0	0.0	389.1	123.0	366.9
	10	107846.8	108913.6	107846.8	0.0	1.0	0.0	485.7	201.3	432.7
	15	92806.9	92920.7	92669.6	0.1	0.3	0.0	1566.4	965.0	816.4
	20	83385.9	83385.9	83385.9	0.0	0.0	0.0	3857.5	2422.7	1130.0
					0.1	0.1	0.0	198.6	114.2	116.8

Chapter 4

Models and solution methods for the uncapacitated r-allocation p-hub equitable center problem

Summary

In this chapter we study the uncapacitated r-allocation p-hub center problem (with $1 < r < p$) and explore alternative models and solution procedures for this problem. In particular, we are interested in studying a version of the uncapacitated r-allocation p-hub center problem that applies to situations for which an element of equity or fairness is important. Given a network with link costs and a set of o-d demands, it is possible to calculate (or know as an input) an ideal minimum cost for each pairwise demand. We will assume that it is desired to find customer-oriented solutions for which the actual cost for each demand remains relatively close to their ideal minimum cost. As we will see, objective function of the UrApHMP (Equation (3.1)) is clearly not equipped to produce customer-oriented solutions since it focuses on minimizing the total cost of the company. Other objective functions that we will see may produce some customer-oriented solutions but do not directly measure deviations from the ideal minimum costs. Hence, we introduce a model that seeks to minimize large deviations between actual and ideal cost for all demands. Since cost basis may be significantly different from one demand to another, we focus on the following model based on relative deviations. Decision makers facing hub network design problems may be interested in comparing solutions that trade off cost and customer service. In this context, a bi-objective problem formulation is the appropriate approach to search for solutions that simultaneously consider cost and service. We propose a variant that we have called the Uncapacitated r-Allocation p-Hub Median and Equitable Center Problem with a bi-objective formulation with the goal to construct the set E of efficient solutions to this bi-objective problem.

4.1 Introduction

As shown in Yaman's literature review [103], published work has focused on the p-hub median problem. Both the single and multiple allocation versions of these problems have been studied extensively, see for example the papers by Ilic et al. [50] and Milanovic [71]. The cost-minimizing solutions found with p-hub median models, however, might include routes for origin and destination (o-d) demand pairs that are unreasonably long, calling into question the quality of the service that the network design will provide to some of its customers. The p-hub center problem addresses this issue by minimizing a function of the cost of meeting individual o-d demands. The typical cost structure for a given $i \to k \to l \to j$ path in a hub network is represented in Figure 4.1, where i is the origin, j is the destination, and k and l are hubs.

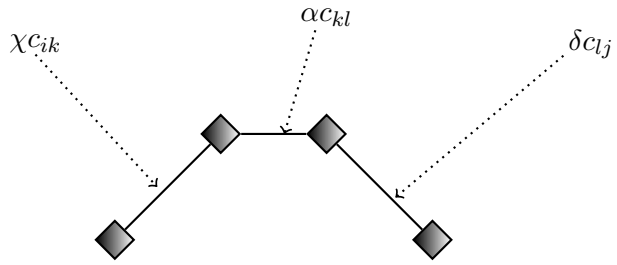

Figure 4.1. Cost structure of demand from origin i to destination j.

Figure 4.1 shows the collection cost c_{ik} from a terminal i (origin) to a hub k, the transfer cost c_{kl} from hub k to hub l, and the distribution cost c_{lj} from hub l to terminal j (destination). Discount factors, if applicable, are associated with each cost: χ for the origin to hub cost, α for the hub to hub cost, and δ for the hub to destination cost. The total cost of sending one unit of traffic from origin i to destination j via hubs k and l is given by $C_{ijkl} = \chi c_{ik} + \alpha c_{kl} + \delta c_{lj}$. Since x_{ijkl} is the fraction of the traffic t_{ij} from i to j routed via hubs k and l, then the objective function for a basic model of the p-hub median problem has the following form:

$$\min \sum_{i \in V} \sum_{j \in V} \sum_{k \in V} \sum_{l \in V} t_{ij} C_{ijkl} x_{ijkl}. \tag{4.1}$$

Note that (4.1) is a more compact representation of (3.1) but both objective functions are the same, i.e., they attempt to minimize the total cost. In contrast, in the p-hub center problem, x_{ijkl} is equal to 1 if the demand t_{ij} from origin i to destination j is routed via hubs k and l, and it is equal to 0 otherwise. The objective function for a basic model of the p-hub center problem has the following mathematical form:

$$\min \max_{i,j,k,l \in V} C_{ijkl} x_{ijkl}. \tag{4.2}$$

This objective function is relevant in hub networks where the flow involves time-sensitive commodities, e.g. people, live animals, and perishable items. In this context, cost refers

to time, and α may be interpreted as a time discount factor due to higher speed on the inter-hub links [18]. An alternative objective function for the p-hub center problem has the following form:

$$\min \max_{i,j,k,l \in V} \{\max \{\chi c_{ik}, \alpha c_{kl}, \delta c_{lj}\} x_{ijkl}\}. \tag{4.3}$$

In (4.3) the cost also refers to time and the function is of interest in situations where the maximum travel time in any link is important. For instance, the model would be applicable to networks where items require some special processing that is only available at the hubs.

We are interested here in studying a version of the uncapacitated r-allocation p-hub center problem that applies to situations for which an element of equity or fairness is important. In particular, given a network with link costs and a set of o-d demands, it is possible to calculate an ideal minimum cost for each pairwise demand. We assume that it is desired to find customer-oriented solutions for which the actual cost for each demand remains relatively close to its ideal minimum cost. Objective function (4.1) is clearly not equipped to produce customer-oriented solutions since it focuses on minimizing the total cost of the company. Objective functions (4.2) and (4.3) may produce some customer-oriented solutions but do not directly measure deviations from the ideal minimum costs. Table 4.1 shows the four cases associated with the calculation of the cost to fulfill the demand from origin i to destination j. Each case depends on whether or not the origin or destination is chosen as a hub. The cost equations assume that when the origin is a terminal then the collection occurs via hub k. Likewise, when the destination is a terminal then the distribution occurs via hub l.

Table 4.1. Cost functions for all terminal-hub combinations

Case	Origin (i)	Destination (j)	Cost
1st	Terminal	Terminal	$\chi c_{ik} + \alpha c_{kl} + \delta c_{lj}$
2nd	Hub	Terminal	$\alpha c_{il} + \delta c_{lj}$
3rd	Terminal	Hub	$\chi c_{ik} + \alpha c_{kj}$
4th	Hub	Hub	αc_{ij}

For our model, we first must determine the minimum cost to fulfill the demand from an origin i to a destination j via hubs k and l. This minimum cost, denoted by $\hat{C}_{ij}$, is calculated as follows:

$$\hat{C}_{ij} = \min_{k,l \in V, k \neq i, l \neq j} (\chi c_{ik} + \alpha c_{kl} + \delta c_{lj}), \qquad \forall i, j \in V. \tag{4.4}$$

A customer-oriented solution is defined as one that seeks to minimize large deviations between actual and ideal cost for all demands. Since cost basis may be significantly different from one demand to another, we focus on the following model based on relative deviations:

$$\min \max_{i,j \in V, i \neq j, t_{ij} > 0} \left\{ \frac{(\sum_{k \in V} \sum_{l \in V} C_{ijkl} x_{ijkl}) - \hat{C}_{ij}}{\hat{C}_{ij}} \right\} \tag{4.5}$$

Subject to:

(3.2) – (3.7), (3.9), and

$$x_{ijkl} \in \{0,1\}, \quad \forall i,j,k,l \in V. \tag{4.6}$$

We refer to this problem as the Uncapacitated r-Allocation p-Hub Equitable Center Problem, which we abbreviate as UrApHECP. This problem arises in competitive environments, such as in the airline industry. For instance, an airline interested in minimizing total cost, might design a network for which some itineraries could offer low value to their customers. Value in this context could be interpreted as the relationship between cost and trip duration. That is, a long itinerary (with several stops) with a high cost may be perceived as offering low value. We assume that, all things being equal (e.g., airfare), travelers would typically prefer a short trip duration (e.g., a direct flight).

To illustrate this point, consider the network with 55 nodes in the Australian Post (AP) instance introduced by Ernst and Krishnamoorthy [31]. The network and demands correspond to the Australian post services that operate from five hubs (i.e., $p = 5$) and in which terminals may be assigned to no more than two hubs (i.e., $r = 2$). The solution procedures developed in Sections 3.3 and 3.4 are designed to minimize the total cost $f(s)$ of a solution s, calculated as given by the objective function (3.1) of UrApHMP. The best solutions obtained by the application of both procedures include o-d pairs for which their cost is significantly larger than their minimum cost. Table 4.2 shows five o-d pairs, their associated hubs, as well as the relative difference D,

$$D_{ij} = 100\,\frac{C_{ijkl} - \hat{C}_{ij}}{\hat{C}_{ij}} \tag{4.7}$$

between the minimum cost and the solution cost obtained with the procedure of Section 3.4. As the costs are related to distance, the customers associated with the five

Table 4.2. Deviation values obtained by solving the UrApHMP.

Origin (i)	Destination (j)	Hub (k)	Hub (l)	D_{ij}
30	29	36	40	493.90%
30	20	36	40	467.40%
30	19	36	16	449.60%
33	32	40	40	433.30%
33	44	40	40	417.90%

o-d demands in Table 4.2 will most likely find their assigned routing unacceptable and might choose to hire a different service. Table 4.3 shows a different set of assignments for the o-d pairs in Table 4.2. The relative difference between the actual costs and the minimum costs has decreased across all o-d pairs.

The change of hub assignments from Table 4.2 to Table 4.3 causes a 0.82% increase in total cost in terms of the objective function of the UrApHMP (Equation (3.1)). Clearly, AP must consider the tradeoff between total cost increase and customer satisfaction and retention. With that in mind, we now develop a method to solve the UrApHECP.

Table 4.3. Smaller deviation values for an alternative solution.

Origin (i)	Destination (j)	Hub (k)	Hub (l)	D_{ij}
30	29	20	20	114.60%
30	20	20	20	5.20%
30	19	20	20	34.16%
33	32	20	20	240.33%
33	44	20	20	269.30%

4.2 A GRASP algorithm

As mentioned in Section 3.3, the GRASP methodology was developed in the late 1980s by Feo and Resende [36]. Algorithm 5 shows the pseudocode of the GRASP framework that we will use to minimize the objective function $g(s)$ associated with a solution s to the UrApHECP. Such a solution is characterized by the hub selection, the hub assignment, and the traffic routing. Each GRASP iteration consists of constructing a trial solution with a greedy randomized procedure (line 4 in Algorithm 5) and then applying local search to the constructed solution (line 5 in Algorithm 5). This two-phase process is repeated until some stopping condition is satisfied (lines 3 to 7 in Algorithm 5). The best solution (s^*) found over all constructions and local searches is returned as the GRASP solution.

```
   Input: stopping criterion
1  s* ← ∅
2  g(s*) ← +∞
3  while stopping criterion is not satisfied do
4  |   s ← ConstructSolution()
5  |   s ← ImproveSolution(s)
6  |   if g(s) ≤ g(s*) then
7  |   |   s* ← s
   Output: s*
```

Algorithm 5: GRASP template

As indicated in Algorithm 5, the design of GRASP entails the customization of the ConstructSolution and the ImproveSolution functions. Constructing solutions in the context of the UrApHECP includes three major steps: the selection of the p hubs, the allocation of each terminal to at most r hubs, and the routing of all the traffic. In this chapter we focus on GRASP designs where the hub selection and the initial hub allocation and routing occur in ConstructSolution and where a search for improved hub allocations within the same hub selection occurs in ImproveSolution. The next sections describe our efforts to create effective construction and improvement functions within the GRASP framework.

4.2.1 Construction methods

Our construction procedures require the ideal (minimum) costs for each o-d pair. These calculations are somewhat onerous but they are performed only once and therefore they can be done off-line before the search procedure is executed. Note that for a given o-d pair, all combinations of hubs must be considered in order to find the minimum cost. As part of this process, we record the number of times $q(h)$ that a node h is selected as the ideal hub for any of the terminals. For instance, if the ideal path for a demand from a terminal i to a terminal j traverses hubs $\hat{k}$ and $\hat{l}$ (i.e., $\hat{C}_{ij} = \chi c_{i\hat{k}} + \alpha c_{\hat{k}\hat{l}} + \delta c_{\hat{l}j}$) then both counts $q(\hat{k})$ and $q(\hat{l})$ are increased by one. If the origin or destination is a hub then only the count for the ideal hub connected to the terminal is incremented. That is, the q count does not include the instances when a node is both a hub and a terminal in the minimum cost. Counting those instances does not add information to the attractiveness of a node to be selected as a hub from the point of view of being used by the terminal nodes in the network.

1 $CL \leftarrow V$
2 // Stage 1: Select p hubs
3 **while** $|H| < p$ **do**
4 $\quad q_{max} \leftarrow \max_{h \in CL} q(h)$
5 $\quad$ Randomly select $h^* \in RCL = \{h : h \in CL, q(h) \geq \beta_1 q_{max}\}$
6 $\quad H \leftarrow H \cup \{h^*\}$
7 $\quad CL \leftarrow CL \setminus \{h^*\}$
8 // Stage 2: Assign terminals to r hubs
9 Calculate $q_i(h), \forall i \in V \setminus H, h \in H$
10 Let $H_i \forall i \in V \setminus H$ be the set of r hubs with the largest $q_i(h)$ values
11 // Stage 3: Route traffic through the network
12 For all (i,j) with $t_{ij} > 0$, let $r_{(i,j)}$ be the path $i \rightarrow k \in H_i \rightarrow l \in H_j \rightarrow j$ that minimizes C_{ijkl}. Let $\Pi = \{\pi_{(i,j)}, \forall (i,j) : t_{ij} > 0\}$
Output: $s = \{H, H_i \forall i, \Pi\}$

Algorithm 6: Construction procedure C1

Algorithm 6 shows the pseudo-code for C1, the first of the two solution construction procedures that we have developed for this application. The hub selection is based on q counts. That is, the greedy function is the maximization of q, which is employed as a measure of the node attractiveness. In the first stage, the procedure calculates q_{max} as the maximum value of q for all the nodes in the candidate list (CL). The candidate list contains all nodes in the graph that have not been chosen as hubs in any of the previous iterations of the construction procedure and therefore $CL = V$ at the beginning of the construction. A restricted candidate list (RCL) is then constructed with those candidate nodes whose q values are "close to" q_{max}, where proximity is determined by the value of the parameter β_1. The next hub is chosen randomly among those nodes in RCL. The selection steps (lines 3 to 7 in Algorithm 6) are repeated p times to produce a set H of

hubs with the specified cardinality.

The second stage of the construction consists of assigning terminals to the chosen set of hubs. Once the set H of hubs has been determined, the ideal costs for all the terminal nodes are recalculated taking into account the hubs defining H. Then, values $q_i(h)$, representing the number of times that hub $h \in H$ appears in the ideal cost for terminal i, when i is either an origin or a destination, are computed. The r hubs with the largest $q_i(h)$ values define the set H_i.

In the third and last stage of the construction the traffic is routed through the network configured by the choices made in the first two stages. Since the links in the network have no capacity limits, a greedy routing is optimal for the chosen hubs and hub allocations. Therefore, each t_{ij} is routed through $k \in H$ and $l \in H$ in order to minimize $C_{ijkl} = \chi c_{ik} + \alpha c_{kl} + \delta c_{lj}$. Finding the set Π of min-cost paths for all o-d pairs requires a computational effort of $\mathcal{O}(n^2 r^2)$. The hub selection H, the hub allocation H_i for all $i \in V \backslash H$, and the set of paths Π for all o-d pairs represent a solution s to the problem (see line 12 in Algorithm 6). i.e., $s = \{H, H_i \forall i, \Pi\}$. The objective function value is $g(s) = D_{max} = \max_{(i,j):t_{ij}>0} D_{ij}$. Note that the calculation of D_{ij}, as shown in (4.7), depends on s.

A variant of C1 (referred to as C2) is obtained by modifying the process in which the q values are calculated. The goal of C2, in contrast to C1, is to avoid the selection of inferior hubs. That is, C2 tries to select hubs that are not going to produce large costs. For this purpose, we define $\check{C}_{ij}$ as the largest cost associated with the traffic pair (i, j). The $\check{C}_{ij}$ values are obtained by simply changing min to max in expression (4.4).

Hubs k and l are "acceptable" as hubs for terminals i and j ($t_{ij} > 0$), respectively, if:

$$C_{ijkl} \leq \hat{C}_{ij} + \beta_2 \left(\check{C}_{ij} - \hat{C}_{ij} \right),$$

where $\beta_2 \in [0, 1]$. In this case, $q(h)$ counts the number of times a node h is selected as "acceptable" for any of the terminals. As in C1, the hub selection step (see line 5 in Algorithm 6) gives preference to candidate nodes with large q values. Note that if $\beta_2 = 0$, C2 coincides with C1, while $\beta_2 = 1$ makes that all p hubs have the same q values (turning the hub selection into a totally random process).

4.2.2 Improvement methods

We designed and tested two solution-improvement procedures, LS1 and LS2. Both of these procedures are local searches, meaning that they stop upon reaching the first local optimal point. Also, both procedures are based on changing the allocation of terminals to hubs, while maintaining the same set of hubs. That is, the hub selection is not changed from the one given by the starting solution generated by one of the construction methods in Section 4.2.1. LS1 attempts to find an improved solution with respect to objective function (4.5) by identifying the (i, j) pair for which $D_{ij} = D_{max}$ (see line 4 in Algorithm 3). This is the traffic pair that determines the value of the objective function. Let k and l be the hubs that terminals i and j are currently using to route t_{ij}. Then, a neighborhood search is launched to identify at least one allocation change for i or j that would reduce

D_{ij} without increasing the D value for the traffic in o-d pairs $(i,*)$, $(*,i)$, $(*,j)$, and $(j,*)$, where $*$ represents any terminal different from i and j. The complete procedure is shown in Algorithm 7.

```
   Input: s
1  Compute D_ij for all (i,j) pairs such that t_ij > 0
2  repeat
3      improve ← FALSE
4      Identify the path π_(i,j) for which D_ij = D_max
5      Find a new path π'_(i,j) through k' ∉ H_i or l' ∉ H_j such that D_i*, D_*i, D_*j,
       D_j* < D_max, where * is any terminal node different from i and j
6      if new path found then
7          Update H_i and/or H_j and Π
8          Update D_i*, D_*i, D_*j, D_j* and D_max
9          improve ← TRUE
10 until improve is FALSE;
   Output: s
```

Algorithm 7: Solution improvement procedure LS1

Line 5 in Algorithm 7 performs the neighborhood search consisting of evaluating different paths for (i,j). If a new path is found, then the relevant sets and values are updated (lines 7 and 8 in Algorithm 7), and the *improve* flag is set to TRUE. If no new path is found, then the procedure terminates.

The second improvement procedure (LS2), summarized in Algorithm 8, does not tackle D_{max} directly as it focuses on minimizing the total cost of the objective function (3.1). The neighborhood search consists of evaluating the exchanges $k \in H_i \leftrightarrow k' \notin H_i$ for all i such that $t_{ij} > 0$ or $t_{ji} > 0$, $i,j \in V$, (lines 4 to 13 in Algorithm 8). This is done in order to minimize the cost of routing the traffic between i and j (lines 8 and 9 in Algorithm 8). For each exchange, all paths using hub k are recomputed by redirecting the corresponding traffic through the least expensive available route (line 8 in Algorithm 8).

A hub allocation for the terminal under consideration (i.e., node i) is changed to improve the routing cost (line 11 in Algorithm 8). The procedure ends when no change in the hub allocation for any of the terminal nodes is able to reduce the current cost of at least one traffic in the network.

4.2.3 The Uncapacitated r-Allocation p-Hub Median and Equitable Center Problem

Decision makers facing hub network design problems are interested in comparing solutions that trade off cost and customer service. In this context, a bi-objective problem formulation is the appropriate approach to search for solutions that simultaneously consider cost and service. We propose the Uncapacitated r-Allocation p-Hub Median and

Input: s
Compute D_{ij} for all (i, j) pairs such that $t_{ij} > 0$
repeat
  improve $\leftarrow$ FALSE
  foreach $i \in V$ **do**
    foreach $k \in H_i$ **do**
      foreach $k' \in H \setminus H_i$ **do**
        Replace k with k'
        Find new paths Π' for all (i, j) and (j, i) for which $k \in \mathfrak{p}_{(i,j)}$ or $\mathfrak{p}_{(j,i)}$
        Compute the cost of routing all (i, j) and (j, i) traffic through Π'
        if cost has improved **then**
          $H_i \leftarrow H_i \setminus \{k\} \cup \{k'\}$
          Update Π, D and D_{max}
          improve $\leftarrow$ TRUE
until improve is FALSE;
Output: s

Algorithm 8: Solution improvement procedure LS2

Equitable Center Problem (UrApHMECP) with the following formulation:

Objective 1: (3.1)

Objective 2: (4.5)

Subject to: (3.2) – (3.7), (3.9), and (4.6).

The goal is to construct the set E of efficient solutions to this bi-objective problem. Efficient solutions are defined in the traditional sense, that is, as those for which it is not possible to decrease the value of one of the objectives without increasing the value of the other objective. (Note that both objectives attempt to minimize a function of the solution.) The outcome of a heuristic approach is an approximation $\hat{E}$ of the efficient frontier E. We propose a BGRASP solution method (see Algorithm 9) that builds upon the strategies introduced earlier. The procedure alternates between C1 and C2 to construct solutions and then sequentially applies LS1, LS2, and again LS1 (lines 3 to 6 in Algorithm 9). The starting solution for each local search is the best solution found in the previous step. Since LS1 uses $g(s)$ to measure solution quality and LS2 uses $f(s)$, the application of both methods is expected to result in a set of solutions that, taken as a whole, produces a reasonable approximation $\hat{E}$ for the UrApHMECP.

Although the updating of $\hat{E}$ is shown in line 7 of Algorithm 9, this set of non-dominated solutions is maintained and updated throughout the application of all procedures, that is, after the completion of C1 or C2 (which produces s) as well as during all the neighborhood searches associated with LS1 and LS2. Maintaining and updating $\hat{E}$ includes the clean-up processes needed to eliminate solutions that become dominated by the inclusion of new solutions in the set. The second call to LS1 (see line 6 in Algorithm 9) has been added after preliminary experiments that have showed that additional

Input: stopping criterion
$\hat{E} \leftarrow \emptyset$
while stopping criterion is not satisfied **do**
  $s \leftarrow$ C1() or C2()
  $s' \leftarrow$ LS1(s)
  $s'' \leftarrow$ LS2(s')
  $s''' \leftarrow$ LS1(s'')
  Update $\hat{E}$
Output: $\hat{E}$

Algorithm 9: Bi-objective GRASP (BGRASP)

non-dominated solutions can be found in the trajectory from s'' to s'''.

4.2.4 Computational experiments

This section describes both the scientific and the competitive testing of our proposed solution method. Scientific testing refers to experiments designed to fine-tune the procedure and also to identify the contribution of the various components of the procedure. The goal of these experiments is to determine what makes the procedure perform well (or not) and why. We then engage in the traditional competitive testing to compare performance against alternative methods to search for solutions to the uncapacitated r-allocation p-hub center problem. The procedures in our method have been implemented in C and the integer linear programming formulations have been solved using CPLEX 12.6.1. All the results reported in this section have been obtained by running our codes on an Intel Core i5-4200U PCU @ 1.6 GHz and 8GB of RAM laptop computer with the Ubuntu Linux 14.04.02–64bits operating system. For the single-objective problems, we use the following metrics to measure performance:

- $Value$: Average objective value of the best solutions obtained by the procedure on the instances considered in the experiment.
- Dev: Average percentage deviation from a reference solution, where the reference solution depends on the testing (i.e., scientific or competitive).
- $Best$: Number of instances in a set for which a procedure is able to match the reference solution, where the reference solution depends on the testing (i.e., scientific or competitive).
- CPU: Average computing time in seconds employed by the algorithm.

4.2.4.1 Problem instances

The following two sets of instances have been used:

1. The **CAB** (Civil Aviation Board) data set. As in other chapters, we have used this network to generate 91 instances with $p \in \{3, \ldots, 5\}$, and $r \in \{2,\ldots, p-1\}$. The following discount factors have been widely used: $\chi = 1$, $\delta = 1$, and $\alpha = 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9,$ and 1.0.

2. The **AP** (Australian Post) data set. As with CAB, many authors have generated various instances from the original network data. We have extended this set of instances by generating 56 more with $n = 30$, 40, 50, 60, 75, 80, 90, and 100 nodes. For these instances, $p \in \{3,\ldots,7\}$ and $r \in \{2,\ldots, p-1\}$. Discount factors are set to χ= 3, α= 0.75 and δ= 2. These instances have asymmetric flows between the nodes (i.e., for a given pair of nodes i and j, t_{ij} is not necessarily equal to t_{ji}). Moreover, flows from one node to itself can be positive (i.e., t_{ii} can be strictly positive for a given i).

4.2.4.2 Scientific testing

From the complete set of 147 instances corresponding to the CAB and AP data sets described above, we have created a training set of 20 instances of various sizes and values of p and r. Specifically, the training set contains 10 instances from the CAB set, with $n = 20$ $p \in \{3, 4, 5\}$ and $r \in \{2,\ 3, 4\}$, and 10 AP instances with $30 \leq n \leq 90$, $p \in \{5, 6, 7\}$, and $r \in \{2, 3, 4, 5\}$. The goal of scientific testing is to assess the merit of the various elements included in a solution procedure. Since the tests isolate these elements, it is not expected that the quality of the solutions obtained by these partial procedures will rival those of the best-known (or optimal) solutions that were found with complete search processes. Therefore, for the purpose of scientific testing, it is customary to use as reference solutions in the calculation of *Dev* the best solutions that the elements being tested are able to produce. This enables the detection of statistical differences between the performance of specific configurations of the elements under study.

In the first experiment, we study the construction methods described in Section 4.2.1 to assess their merit in terms of solution quality in the context of the UrApHECP. The performance of this solution generators depends on the values of their corresponding parameters, β_1 for C1 and β_2 for C2, which determine the size of the RCL in each construction method. Table 4.4 shows the *Dev* values corresponding to C1 and C2 for β_1 and β_2 values varying from 0.1 to 1.0.

Each *Dev* value is calculated over the 20 best solutions constructed by C1 and C2, one for each instance in the training set. The best solution for a problem instance is selected among 100 solutions generated by each method and parameter setting. For example, C1 with $\beta_1 = 0.1$ is used to generate 100 solutions for the first instance in the training set. The best solution out of these 100 is selected. The process is repeated 19 more times for all instances in the training set. Then, $Dev = 22.9\%$ is the result of averaging the deviation against the reference solution of the 20 best solutions found (one for each problem instance).

With the goal of detecting differences among the various parameter values, we applied the non-parametric Friedman test for multiple correlated samples. The test has been

Table 4.4. *Dev* values for C1 and C2 solutions of the training set instances

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
β_1	22.90%	27.90%	18.90%	20.00%	33.10%	42.90%	61.70%	74.30%	87.70%	94.90%
β_2	17.70%	28.40%	26.20%	32.60%	33.20%	33.10%	32.60%	17.80%	22.60%	21.50%

performed on the set of best solutions obtained by each construction method and each parameter value in Table 4.4. This test computes, for each instance, the rank of each method according to solution quality. Then, it calculates the average rank for each method across all instances. If the averages differ greatly, the associated p-value or level of significance is small. For C1, the Friedman test results in a p-value of 0.000, indicating that there are statistically significant differences among the average solution quality obtained by employing different β_1 values. We then have performed a paired sample test (Wilcoxon) for the best solutions obtained by C1 with $\beta_1 = 0.3$ and $\beta_1 = 0.4$, which are the settings that result in the smallest *Dev* values. The resulting p-value of 0.53 indicates that there is no significant difference between setting β_1 to 0.3 or 0.4. We chose $\beta_1 = 0.3$ for the rest of experiments.

The Friedman test for the best solutions obtained by C2 with the 10 settings of β_2 shown in Table 4.4 yielded a p-value of 0.13. This result does not provide evidence that there is a significant difference in performance of C2 among all the β_2 values that were tested. Although not supported from a statistical point of view, the values $\beta_2 = 0.1$ and $\beta_2 = 0.8$ resulted in the smallest deviations, and therefore we decided to use $\beta_2 = 0.1$.

We now search for the most effective combination of construction and improvement within GRASP in Algorithm 5. To that end, we test all combinations of construction and improvement procedures, i.e., C1 ($\beta_1 = 0.3$) with LS1 and LS2 as well as C2 ($\beta_2 = 0.1$) with LS1 and LS2. The stopping criterion is set to 100 GRASP iterations for each instance in our training set. Table 4.5 shows the performance measures associated with each combination. The *Dev* values in Table 4.5 may be compared with the *Dev* values

Table 4.5. Performance of various GRASP (Algorithm 5) configurations.

Combination	Value	Dev	Best	CPU
C1 ($\beta_1 = 0.3$) + LS1	183.0	0.0679	10	0.8
C1 ($\beta_1 = 0.3$) + LS2	183.6	0.0862	8	13.7
C2 ($\beta_2 = 0.1$) + LS1	186.6	0.1119	11	3.7
C2 ($\beta_2 = 0.1$) + LS2	190.7	0.1541	8	16.6

in Table 4.4 for the corresponding β_1 and β_2 settings to measure the contribution of the local search. On average, LS1 and LS2 improve the initial C1 solutions by 12.11% (18.9% - 6.79%) and 10.28% (18.9% - 8.62%), respectively. Likewise, LS1 and LS2 improve the initial C2 solutions by 6.51% (17.7% - 11.19%) and 2.29% (17.7% - 15.41%) respectively. The results in Table 4.5 indicate that the most effective combination consists of pairing C1 with LS1.

We are also interested in identifying the contribution of the local search procedures within BGRASP. Of particular interest is the marginal improvement obtained by the order in which the local search procedures are applied (as shown in Algorithm 9). Using a termination criterion of 100 iterations, for each instance in the training set we calculate $\hat{E}$ after the construction step (line 3 in Algorithm 9), then after the completion of LS1 and LS2 (lines 4 and 5 in Algorithm 9), and finally after the completion of the additional call to LS1 (line 6 in Algorithm 9). Figure 4.2 illustrates the outcome of this experiment on one of the training instances.

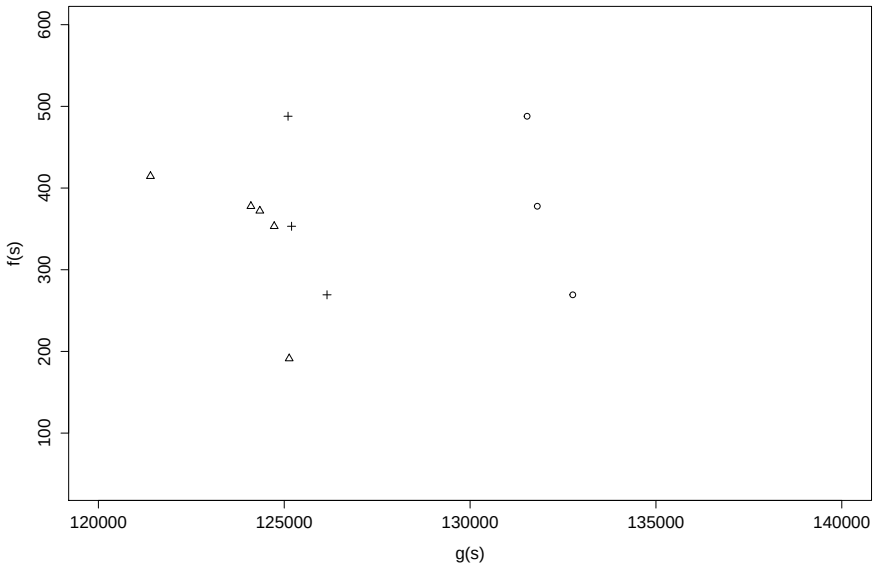

Figure 4.2. Efficient frontier approximations for an instance in the training set.

Figure 4.2 shows that each of the local search stages contribute to finding a better approximation $\hat{E}$ (by moving the points toward the left and the bottom of the graph). While this depiction of the local search contributions is only for one of the problem instances in the training set, we have observed a similar behavior in the remaining instances in the set.

4.2.4.3 Competitive testing

As mentioned in Section 4.1, we have delevoped in Chapter 3 a solution procedure, based on the scatter search methodology, for the UrApHMP. We argued, through an illustrative example, that a solution procedure designed for the UrApHMP is not capable of producing high quality solutions for the UrApHECP. We now provide experimental evidence to support this argument. The experiment consists of applying the scatter search procedure (labeled as SS) of Section 3.4 in Chapter 3 and our GRASP (Algorithm 5 with C1 and LS1) to all problems in our test set. The stopping criterion, for both

procedures, is set to a maximum number of 1000 iterations for each problem instance. Table 4.6 shows the average objective function value (i.e., $Value$) and the Dev values calculated against the best-known solutions as the reference points. The set of instances has been divided in six subsets according to the number of nodes.

Table 4.6. Performance comparison between SS and GRASP on UrApHMP and UrApHECP

			UrApHMP		UrApHECP		CPU seconds	
Set	n	#inst	Value (SS)	Dev (GRASP)	Value (GRASP)	Dev (SS)	SS	GRASP
CAB	15	27	15059746	7.60%	48.4	84.00%	0.1	0.6
CAB	25	54	6.826E+09	14.20%	128.1	179.70%	0.9	13.5
AP	40	6	138422.4	6.20%	121.5	37.80%	2.8	46.8
AP	60	10	132012.9	15.30%	271.5	28.70%	11.2	305.3
AP	80	15	129413.6	18.60%	392.1	141.40%	49.9	837.5
AP	100	15	130146.4	23.40%	674.3	56.60%	102.3	2757.5
Total/Average		127	2.905E+09	14.10%	217.8	121.70%	19.4	660.2

The procedures perform as expected. SS finds the best-known solutions for all instances when solving the UrApHMP. That is, a Dev column for SS under UrApHMP would consists of all zeros. The third column of Table 4.6 shows the average objective values of the best-known solutions for UrApHMP. The GRASP solutions to these problems are, on average, 14.1% away from the best-known solutions, as indicated by the values in the Dev (GRASP) column in Table 4.6. When solving the UrApHECP on the same instances, the performance of SS and GRASP is exactly the opposite to the previous case. That is, GRASP finds all the best-known solutions and the SS solutions are, on average, 121.7% away from these reference points. These results provide empirical evidence that UrApHMP and UrApHECP are significantly different problems, requiring specialized procedures and hence justifying our current work. We point out that the computational effort of GRASP is about an order of magnitude larger than the effort employed by SS. This is due to the complexity of the calculations associated with the optimization of the UrApHECP, which we illustrated in Section 4.2.1.

In our next competitive experiment, we have attempted to obtain solutions for the UrApHECP by solving the MIP formulation in Section 4.1. The well-known commercial MIP solver CPLEX 12.6.1 was unable to provide any feasible integer solutions to problems with $n > 20$. Therefore, we eliminated the possibility of comparing the performance of our method against this optimization package. We then have attempted a comparison of our GRASP for the UrApHECP with two general-purpose metaheuristic optimizers, LocalSolver and OptQuest[1]. As a preliminary experiment, both of these optimizers have been tried on a small AP instance with $n = 20$, $p = 4$, $r = 2$, $\chi = 3$, $\alpha = 0.75$ and $\delta = 2$. The best-known solutions for UrApHMP and UrApHECP have objective function values of $f(s) = 132263$ and $g(s) = 50.65\%$, respectively. These solutions have been found

[1] LocalSolver is a product of Innovation 24 (localsolver.com) and OptQuest is a product of OptTek Systems (opttek.com).

in a fraction of a CPU second by SS (for UrApHMP) and GRASP (for UrApHECP). LocalSolver was able to match these solutions within 20-minute runs. However, OptQuest was only able to find a solution for UrApHMP with an objective function value of $f(s) = 136704.72$ and a solution for UrApHECP with an objective function value of $g(s) = 84.44\%$. Similar results were found with additional small problems and therefore we have decided to continue our competitive testing only with LocalSolver.

Given the general nature of LocalSolver, we have allowed it to run for 20 minutes for each problem instance, which represents about twice the computational effort of our GRASP. LocalSolver has run into memory issues on problems with $n > 60$, therefore our experiments are limited to the 35 instances summarized in Table 4.7.

Table 4.7. Comparison between LocalSolver and GRASP on the UrApHECP

			LocalSolver			GRASP		
Set	n	# inst	Value	Dev	Best	Value	Dev	Best
CAB	15	9	60.8	0.00%	9	61.1	0.58%	8
CAB	25	10	235.5	14.40%	2	207.2	0.56%	9
AP	40	6	132.3	15.50%	1	113.5	0.31%	5
AP	60	10	402.5	47.70%	0	271.5	0.00%	10
Total/Average		35	220.6	20.40%	12	172	0.36%	32

The quality of the solutions that LocalSolver finds decreases with the size of the problem. For the CAB instances, LocalSolver's performance is somewhat comparable to GRASP. However, the gap widens when tackling the larger AP instances.

To the best of our knowledge, the Literature does not include a procedure specifically developed for the UrApHMECP. Once again, we have started by executing both LocalSolver and OptQuest on a small AP problem with $n = 20$. The narrow multiobjective capabilities of LocalSolver limit our analysis. LocalSolver is not designed to search for an approximation of an efficient frontier. If more than one objective is defined within a LocalSolver model, the system treats them lexicographically. The objectives are considered in the order that they are declared. The execution of LocalSolver produces a single solution that is the best approximation of the lexicographic optimization of the objectives. For each problem instance, we attempt to produce four non-dominated solutions in the efficient frontier:

1. minimize $f(s)$ (modelNumber = 1).
2. minimize $g(s)$ (modelNumber = 2).
3. minimize $g(s)$ and then $f(s)$ (modelNumber = 3).
4. minimize $f(s)$ and then $g(s)$ (modelNumber = 4).

Given the heuristic nature of the LocalSolver, there is no guarantee that the four solutions will be non-dominated and therefore we remove those that are dominated.

OptQuest has a multiobjective setting in which the procedure searches for an approximation of the efficient frontier associated with the objective functions defined in the optimization model. Figure 4.3 shows the non-dominated solutions for the biobjective mode (3.1), (4.5), (3.2) – (3.7), (3.9), and (4.6) found by BGRASP, LocalSolver, and OptQuest on an AP problem with $n = 20$.

The BGRASP solutions in Figure 4.3 dominate two of the LocalSolver solutions, the ones found using the lexicographical multiobjective functionality of LocalSolver. The solution found using the multiobjective OptQuest search is also dominated by the GRASP solutions. In addition, one of the LocalSolver solutions dominates the OptQuest solution. Since similar results were found with other smaller instances, we focused our competitive testing on LocalSolver.

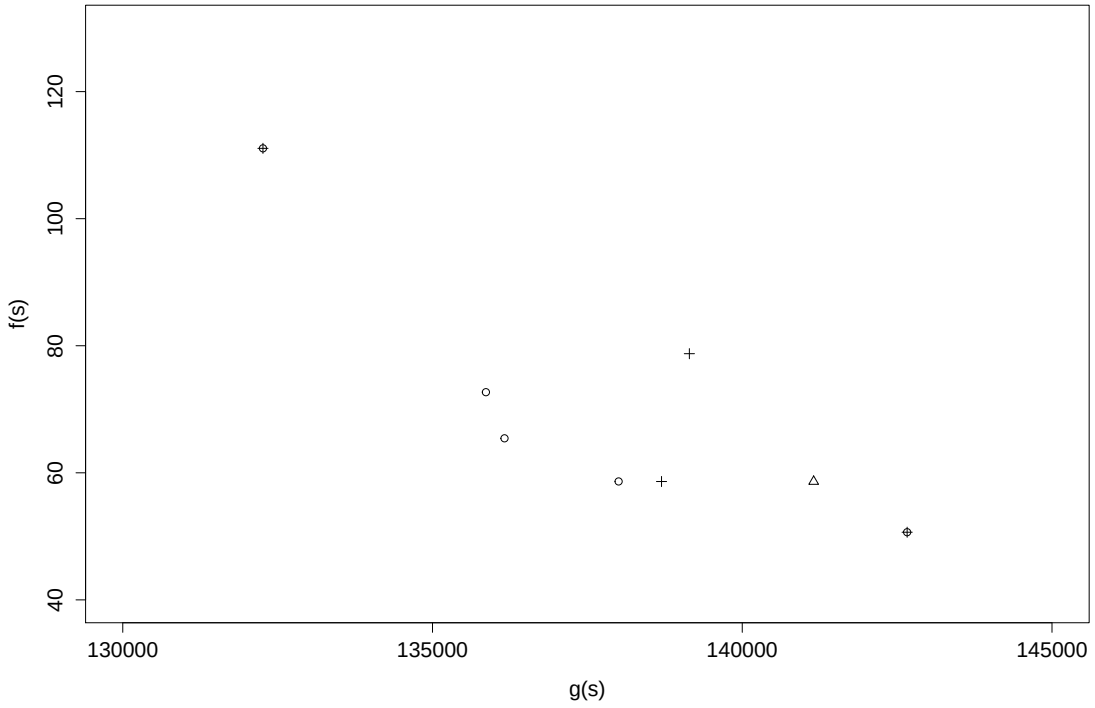

Figure 4.3. Bi-objective solutions to an AP instance with $n = 20$.

We employ the same 35 instances in Table 4.7 to run BGRASP for 1000 iterations and LocalSolver for 20 minutes. The comparison is done in terms of the hypervolume. This metric was developed by Zitzler and Thiele [106] and measures the size of the space covered, which approximates the volume where the dominated points reside. Hence, the larger the hypervolume the better. The number of points in $\hat{E}$ is another measure of interest in multiobjective optimization. Table 4.8 reports both the average hypervolume and the average number of points found by BGRASP and LocalSolver for each subset

of problems.

Table 4.8. Comparison between LocalSolver and BGRASP on the UrApHMECP

			Hypervolume		Number of points	
Set	n	Instances	BGRASP	LocalSolver	BGRASP	LocalSolver
CAB	15	9	0.29	0.11	1.89	1.78
CAB	25	10	0.84	0.15	6.7	2.5
AP	40	6	0.77	0.03	3.67	1.83
AP	60	10	0.71	0.2	4.4	1.8
Total/Average		35	0.65	0.13	4.29	2

The nondominated solutions that BGRASP finds result in a hypervolume that on average is about 5 times larger than the hypervolume corresponding to the LocalSolver solutions. Also on average, half of the solutions that LocaSolver finds for each problem instance are dominated (as indicated by the value of 2.00 in the last row of the LocalSolver column under the Number of points heading).

4.3 Concluding remarks

We have studied in this chapter two hub-network design problems that have not been addressed in the Literature, the Uncapacitated r-Allocation p-Hub Equitable Center Problem (UrApHECP) and the Uncapacitated r-Allocation p-Hub Median and Equitable Center Problem (UrApHMECP). Modeling equity as a relative deviation from an ideal value (e.g., cost) is applicable in contexts where solutions for which some of the demand is fulfilled using routes that are far from ideal are not desirable. We argue that the airline industry is such that if routes connecting two terminal nodes (i.e., an origin and a destination) are far from ideal (e.g., a direct flight) it could result in a loss of customers to the competition. In order to keep operational costs in perspective, we suggest to formulate the problem as a bi-objective optimization model that accounts for both cost-efficiency and service.

The UrApHECP is a so-called minimax model, since it seeks to minimize the maximum deviation from the ideal values. These problems create "flat" objective function spaces because many solutions share the same objective function value. Empirical evidence points to multistart methods as an effective way of searching these spaces. This is the reason the selection of GRASP as the underlying methodology for our solution procedure. A careful scientific testing has been performed to identify a high-performing configuration of our search method. This has been followed by a competitive testing designed to show the need for a specialized procedure for both the UrApHECP and the UrApHMECP. Although our tests are limited to problem sizes that LocalSolver is able to handle, the proposed GRASP and BGRASP are scalable and able to tackle problems of realistic size.

Chapter 5

Models and solution methods for the stochastic r-allocation p-hub median problem with non-stop services

Summary

In this chapter we extend the uncapacitated r-allocation p-hub median problem in two directions: (i) by considering uncertainty in the traffics and in the shipping costs, and (ii) by considering the possibility of shipping traffic directly between terminals (non-stop services) in case this renders savings in the overall cost. In particular, we associate uncertainty with the traffics to be shipped between nodes and with the transportation costs. If we assume that such uncertainty can be captured by a finite set of scenarios, each of which having some known occurrence probability, it is possible to develop a compact formulation for the deterministic equivalent problem. However, even for small instances of the problem, the model becomes too large to be easily tackled by a general purpose solver. This fact motivates the development of an approximate procedure, whose starting point is the determination of a feasible solution to every (deterministic) single-scenario problem. These solutions are then embedded into a process inspired by Path Relinking: gradually an initial solution to the overall problem is transformed by the incorporation of attributes from some guiding solutions. In our case, the guiding solutions are those found for the single-scenario problems.

We report and discuss the results of the computational experiments performed using instances randomly generated for the new problem using the well-known CAB data set.

5.1 Introduction

An important aspect concerning hub location is related with the shipment of the traffic originated at each node. Most of the literature assumes that all traffic must be routed via at least one hub which prevents direct shipments between terminals. However, in some applications (e.g., in Logistics) it may be possible (and even advantageous) to make direct shipments between terminals. This is the case, for instance, if the volume of traffic between the nodes is high and no specialized facility or equipment is required for processing that traffic. Some authors have noticed the practical relevance of considering these "non-stop" services. This is the case with Aykin [10, 11, 12], Nickel et al. [73], Sung and Jin [100], and Wagner [101]. Nevertheless, the literature capturing this feature is still much scarce.

Another aspect that is increasingly attracting the attention of academicians and practitioners concerns embedding uncertainty in optimization models. This is not a new topic. However, the technological developments we have observed in the past decades (e.g., the huge increase in computing power) made possible to give more relevance to that aspect. This has led to more comprehensive and, from a practical point of view, more relevant models. Hub location has not been ignored in this trend. In fact, several works can already be found in the literature capturing uncertainty in optimization models developed for hub location problems. To the best of our knowledge, the paper by Marianov and Serra [65] is the first work dealing with uncertainty within the context of hub location. Sim et al. [95] introduce the stochastic p-hub center problem. Often, the uncertain parameters are related with the amount of traffic to ship. This is the case, for instance, with the work by Bollapragada et al. [15] . However, other possibilities, as uncertainty in the costs, have been studied (the reader should refer to Contreras et al. [23], and Alumur et al. [7]).

Most of the "classical" hub location problems are $\mathcal{NP}$-hard. Accordingly, the same holds for many of their extensions and thus it is not surprising to find much literature presenting heuristic procedures in this field.

In this chapter we will extend the uncapacitated r-allocation p-hub median problem in two directions: (i) by considering uncertainty in the traffics and in the shipping costs, and (ii) by allowing non-stop services.

We assume that uncertainty can be represented by a stochastic random vector with some known probability distribution. This leads to the adoption of a two-stage stochastic modeling framework for the problem: the first-stage decisions refer to the network design (selection of hubs and allocation of terminals to the hubs); the second-stage decision is dependent on how uncertain is revealed and regards the transportation of the traffics through the network.

When the underlying random vector above mentioned has a finite support, it is possible to derive a compact mixed-integer linear programming formulation for the deterministic equivalent problem. Nevertheless, this is a large-scale optimization model even for small instances of the problem, which motivates the development of a heuristic algorithm for obtaining high-quality feasible solutions. The procedure we propose is in-

spired on the Path Relinking methodology ([44]). In particular, it progressively changes an initial solution to the problem by incorporating attributes from a set of guiding solutions. In our case, the guiding solutions are feasible solutions previously obtained to the single-scenario problems (one for each).

The development of metaheuristics for stochastic combinatorial optimization problems is not a new topic as we can observe in the survey paper by Bianchi et al. [14]. Nevertheless, we can also conclude that most of the work was developed on stochastic traveling salesman problems, on stochastic vehicle routing, and on stochastic scheduling. Within the context of facility location, no much work can be found. The paper by Albareda-Sambola et al. [4] is a good exception. The authors introduced a so-called fix-and-relax-coordination procedure for a multi-period location–allocation problem under uncertainty. This is a specialization of the fix-and-relax heuristic ([27], [32]) embedding a branch-and-fix coordination two-stage solution algorithm ([5]).

The new methodology we propose in this chapter represents a contribution to the development of heuristic approaches for stochastic hub location problems that can be easily extended to other stochastic discrete optimization problems.

This chapter is organized as follows: in Section 5.2 we present a mathematical model for the deterministic version of the problem we are investigating. Afterwards, we extend the model to a setting in which the traffics and shipping costs are stochastic. In Section 5.3 we introduce the new heuristic procedure. In particular, Section 5.3.1 is devoted to developing a heuristic to the deterministic (single-scenario) version of the problem. Finally, in Section 5.4, we present the computational tests performed in order to assess the quality of the new heuristic.

5.2 The uncapacitated r-allocation p-hub median problem with non-stop services

In this section we start by summarizing the most important results concerning the (deterministic) uncapacitated r-allocation p-hub median problem introduced by Yaman [103], as well as the special case in which non-stop services ([100]) are allowed. Then, we introduce a stochastic version, which, as far as we know, is the first time it has been considered in the Literature.

5.2.1 Deterministic model

Consider a network $G = (V, E)$ with a set of demand nodes V and a set of edges E, and let t_{ij} be the amount of traffic to be sent between each pair of nodes i and j. Recall that in the uncapacitated r-allocation p-hub median problem (UrApHMP), traffic t_{ij} has to be routed along a path $i \to k \to l \to j$, where nodes k and $l \in V$ are used as intermediate points for this transportation (see Chapter 3). The UrApHMP consists of choosing a set H of p nodes that are used as intermediate transfer points between any pair of nodes in G, allocating each terminal to at most r of the p hubs, and such that

the total transportation cost is minimized. The traffic to and from each terminal can be routed via one or several hubs among the ones to which the terminal is allocated.

Yaman [103] pointed out the possibility of considering non-stop services between origin–destination pairs. Non-stop services refers to the possibility of sending traffics between any pair of nodes using a direct edge, at a given cost, rather than going through their corresponding hubs. The direct transportation via such a direct edge is called a non-stop service. The problem we study in this chapter is the UrApHMP with non-stop services, where fixed assignment costs of terminals to hubs are also considered. We call this problem the Uncapacitated r-Allocation p-Hub Median Problem with non-stop services (UrApHMP-NSS).

In order to formulate the problem as a mixed-integer linear optimization problem, we define the following variables:

- Given a node $k \in V$, $z_{kk} = 1$ if node k is set to be a hub and $z_{kk} = 0$, and otherwise. Given a non-hub node $i \in V$, $z_{ik} = 1$ if node i is assigned to node k and $z_{ik} = 0$ otherwise.
- Moreover, we define x_{ijkl} as the proportion of the traffic t_{ij} from node i to node j that travels along the path $i \to k \to l \to j$, where k and l are the nodes that will be used as hubs.
- Finally, for two nodes $i, j \in V$, $y_{ij} = 1$ if t_{ij} is routed on a non-stop service, and $y_{ij} = 0$ otherwise.

We formulate the UrApHMP-NSS as follows.

$$\min \sum_{i,k\in V, i\neq k} a_{ik} z_{ik} + \sum_{i,j,k,l\in V} t_{ij}(\chi c_{ik} + \alpha c_{kl} + \delta c_{lj}) x_{ijkl} + \sum_{i,j\in V} (t_{ij} d_{ij} + b_{ij}) y_{ij} \tag{5.1}$$

$$\text{Subject to} \quad \sum_{k\in V} z_{kk} = p \tag{5.2}$$

$$\sum_{k\in V} z_{ik} \leq r, \qquad \forall i \in V \tag{5.3}$$

$$z_{ik} \leq z_{kk}, \qquad \forall i, k \in V \tag{5.4}$$

$$\sum_{l\in V} x_{ijkl} \leq z_{ik}, \qquad \forall i, j, k \in V \tag{5.5}$$

$$\sum_{k\in V} x_{ijkl} \leq z_{jl}, \qquad \forall i, j, l \in V \tag{5.6}$$

$$\sum_{k\in V} \sum_{l\in V} x_{ijkl} + y_{ij} = 1, \qquad \forall i, j \in V : t_{ij} > 0 \tag{5.7}$$

$$x_{ijkl} \geq 0, \qquad \forall i, j, k, l \in V \tag{5.8}$$

$$y_{ij} \in \{0, 1\}, \qquad \forall i, j \in V \tag{5.9}$$

$$z_{ik} \in \{0, 1\}, \qquad \forall i, k \in V. \tag{5.10}$$

The objective function (5.1) represents the total cost. It consists of the allocation cost of terminals to hubs and the transportation cost of the traffics. The first term refers to the assignment cost, a_{ik}, of each node $i \in V$ to a particular hub $k \in V$, regardless of the amount of traffic node i sends or receives through k. The second term represents the cost of transporting all traffics t_{ij} through the hubs, where χ, α, and δ are unit rates for collection (origin-hub), transfer (hub-hub) and distribution (hub-destination), respectively. In this term, c_{ik}, c_{kl}, and c_{lj} denote the cost of shipping all the traffic t_{ij} via the edges (i,k), (k,l), and (l,j), respectively. The last term of the objective function describes the cost of transporting traffics using non-stop services. In this term, d_{ij} denotes the cost of shipping the traffic t_{ij} using a non-stop service. Note that there is also a fixed cost, b_{ij}, associated with the use of this direct edge between i and j.

Constraint (5.2) imposes to use exactly p nodes as hubs, while constraints (5.3) restrict any node to be assigned to at most r of the p hubs. Constraints (5.4) assure that if node k is not a hub, node i cannot be assigned to it. Moreover, constraints (5.5) and (5.6) guarantee that if nodes i and j are not assigned to hubs k and l, no traffic can be sent between i and j through those hubs. Constraints (5.7) ensure that all traffic of the network is routed, either using the hub connections or non-stop services. Finally, the variables domains are stated in constraints (5.8) – (5.10).

5.2.2 A two-stage stochastic model

We introduce here a stochastic version of the above problem. In particular, we assume that demands and transportation costs are not known in advance but can be captured by a probability distribution. This is motivated by the following observations:

1. A hub location problem has embedded a network design problem (locating the hubs and allocating the terminals to the hubs) whose related decisions often have a long-lasting effect. Hence, these are intrinsically strategic decisions typically made before having perfect information about the future. On the other hand, the decisions about transportation of the traffics are often operational decisions that can be made just in time, when precise information is available, i.e., after uncertainty is revealed.

2. What is more, transportation costs are often related to the price of resources like electricity or oil, and therefore they are quite difficult to predict. However, the actual transportation costs incurred will depend on how the network design decisions were made since the latter condition the former.

These observations motivate the use of a two-stage stochastic programming optimization model in which the here-and-now decisions (first-stage) concern the network design and the recourse decisions (second-stage) regard the transportation of the traffics. The latter are called “recourse decisions” because they are made in such a way that the best response is given (depending on the occurring scenario) to the setting defined by the first-stage decisions. This modeling framework is not new within the context of hub location. In the works by Contreras et al. [23] and by Alumur et al. [7] we can observe stochastic hub

location problems with the network design decisions separated from the transportation decisions.

We call scenario a complete realization of all the uncertain parameters. The number of possible scenarios can be either finite or infinite, depending on the supports of the random variables involved in the problem. In fact, if, for every $i, j \in V$, the traffic t_{ij} is assumed to be a random variable, the same happening with the costs c_{ij} and d_{ij}, $(i, j) \in E$, then the random vector underlying the problem is

$$\boldsymbol{\xi} = \left[[t_{ij}]_{i,j \in V}, [c_{ij}]_{(i,j) \in E}, [d_{ij}]_{(i,j \in E)}, [b_{ij}]_{(i,j \in E)} \right].$$

Each realization of this random vector is a scenario. We assume that it is possible to compute or estimate accurately the probability associated with each scenario (we refer the reader to [7] for a deeper discussion on this issue).

We introduce a stochastic version of the UrApHMP-NSS as follows:

$$\begin{aligned} \min \quad & \sum_{i,k \in V, i \neq k} a_{ik} z_{ik} + \mathcal{Q}(\mathbf{z}) && (5.11) \\ \text{Subject to:} \quad & (5.2) - (5.4), (5.10), \end{aligned}$$

where $\mathcal{Q}(\mathbf{z}) = E_{\boldsymbol{\xi}}[Q(\mathbf{z}, \boldsymbol{\xi})]$ is the mathematical expectation with respect to ξ, and

$$\begin{aligned} Q(\mathbf{z}, \boldsymbol{\xi}) = \quad \min \quad & \sum_{i,j,k,l \in V} t_{ij}(\chi c_{ik} + \alpha c_{kl} + \delta c_{lj}) x_{ijkl} \\ & + \sum_{i,j \in V} (t_{ij} d_{ij} + b_{ij}) y_{ij} && (5.12) \\ \text{Subject to} \quad & \sum_{l \in V} x_{ijkl} \leq z_{ik}, \qquad \forall i, j, k \in V && (5.13) \\ & \sum_{k \in V} x_{ijkl} \leq z_{jl}, \qquad \forall i, j, l \in V && (5.14) \\ & \sum_{k \in V} \sum_{l \in V} x_{ijkl} + y_{ij} = 1, \quad \forall i, j \in V : t_{ij} > 0 && (5.15) \\ & x_{ijkl} \geq 0, \qquad \forall i, j, k, l \in V && (5.16) \\ & y_{ij} \in \{0, 1\}, \qquad \forall i, j \in V. && (5.17) \end{aligned}$$

If the support, say Ξ, of the random vector $\boldsymbol{\xi}$ is finite, we can index the scenarios in the set $S = \{1, \ldots, |\Xi|\}$. Accordingly, we can also index the stochastic parameters and the second-stage decision as follows: for $s \in S$, t_{ijs} is the traffic to be sent from i to j under scenario s, c_{iks}, c_{kls}, and c_{ljs} correspond to the cost of shipping all the traffic t_{ijs} via the edges (i, k), (k, l), and (l, j), respectively, under scenario s, d_{ijs} denotes the cost of shipping the traffic t_{ijs} using a non-stop service under scenario s, x_{ijkls} is the the proportion of the traffic t_{ij} from node i to node j that travels along the path $i \to k \to l \to j$, where k and l are the nodes that will be used as hubs, and y_{ijs} is a

binary variable equal to 1 if a non-stop service is used in scenario s for shipping the traffic t_{ijs} and 0 otherwise.

Using this new notation and representing by π_s the probability associated with scenario $s \in S$, we can present the so-called extensive form of the deterministic equivalent:

$$\begin{aligned}\min \quad & \sum_{i,k\in V, i\neq k} a_{ik} z_{ik} \\ & + \sum_{s\in S} \pi_s \left[\sum_{i,j,k,l\in V} t_{ijs}(\chi c_{iks} + \alpha c_{kls} + \delta c_{ljs}) x_{ijkls} \right. \\ & \left. + \sum_{i,j\in V} (t_{ijs} d_{ijs} + b_{ijs}) y_{ijs} \right] \end{aligned} \tag{5.18}$$

Subjec to: (5.2) – (5.4), (5.10)

$$\sum_{l\in V} x_{ijkls} \leq z_{ik}, \qquad \forall i,j,k \in V,\ s \in S \tag{5.19}$$

$$\sum_{k\in V} x_{ijkls} \leq z_{jl}, \qquad \forall i,j,l \in V,\ s \in S \tag{5.20}$$

$$\sum_{k\in V} \sum_{l\in V} x_{ijkls} + y_{ijs} = 1, \qquad \forall i,j \in V,\ s \in S \tag{5.21}$$

$$x_{ijkls} \geq 0, \qquad \forall i,j,k,l \in V,\ s \in S \tag{5.22}$$

$$y_{ijs} \in \{0,1\}, \qquad \forall i,j \in V,\ s \in S. \tag{5.23}$$

The above model will be denoted by $\mathcal{P}$. We note that the non-anticipativity principle is implicit in this model since the choice to be made for the z-variables should will result the same no matter the occurring scenario. In other words, the challenge here is to select a set of hubs and assign the terminals to these hubs in a way that performs well in every possible situation (scenario).

5.2.3 A minmax regret model

The stochastic model just presented lies on the assumption that the probabilities π_s are known. If this is not the case, then, alternatives are necessary for formulating the problem. One possibility explored within the context of hub location by Alumur et al. [7] is to consider a min-max regret model. We can propose the same type of model for the UrApHMP-NSS under uncertainty.

We first notice that model (5.2) – (5.4), (5.10), (5.18) – (5.23) can be solved for a subset of scenarios and, in particular, the model can be solved for a single scenario $s \in S$ by setting $\pi_s = 1$. The resulting solution is the optimal solution for scenario s whose value we can denote by $\mathcal{V}_s$.

After having computed the values $\mathcal{V}_s$, $s \in S$, we can compute the so-called regret of some solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ with respect to a scenario s. This is done as follows:

$$R_s = \sum_{i,k \in V, i \neq k} a_{ik} z_{ik} + \left[\sum_{i,j,k,l \in V} t_{ijs}(\chi c_{iks} + \alpha c_{kls} + \delta c_{ljs}) x_{ijkls} + \sum_{i,j \in V} (t_{ijs} d_{ijs} + b_{ijs}) y_{ijs} \right] - \mathcal{V}_s, \quad s \in S. \tag{5.24}$$

The problem consists of finding the solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ that minimizes the maximum regret we can observe according to:

$$\min \left\{ \max_{s \in S} R_s \right\} \tag{5.25}$$

$$\text{s.t.} \quad (5.2) - (5.4), (5.10), (5.19) - (5.23),$$

$$R_s = \sum_{i,k \in V, i \neq k} a_{ik} z_{ik} + \left[\sum_{i,j,k,l \in V} t_{ijs}(\chi c_{iks} + \alpha c_{kls} + \delta c_{ljs}) x_{ijkls} + \sum_{i,j \in V} (t_{ijs} d_{ijs} + b_{ijs}) y_{ijs} \right] - \mathcal{V}_s, \qquad s \in S. \tag{5.26}$$

5.3 A greedy attributive scenario based constructive method

In this section we present an algorithm for the stochastic UrApHMP-NSS. Since problem $\mathcal{P}$ is a mixed integer linear program with a large number of binary variables even for small instances, it is a very hard task to solve it to optimality. Hence, we propose a heuristic algorithm for finding high-quality feasible solutions to the problem. The idea is to iteratively build solutions to $\mathcal{P}$ using the solutions of single-scenario problems.

For each scenario $s \in S$, let us denote by $\mathcal{P}_s$ the problem associated with s:

$$\min \sum_{i,k \in V, i \neq k} a_{ik} z_{ik} + \sum_{i,j,k,l \in V} t_{ijs}(\chi c_{iks} + \alpha c_{kls} + \delta c_{ljs}) x_{ijkls} + \sum_{i,j \in V} (t_{ijs} d_{ijs} + b_{ijs}) y_{ijs} \tag{5.27}$$

$$\text{s.t.} \quad (5.2) - (5.4), (5.10),$$

$$\sum_{l \in V} x_{ijkls} \leq z_{ik}, \qquad \forall i, j, k \in V \tag{5.28}$$

$$\sum_{k \in V} x_{ijkls} \leq z_{jl}, \qquad \forall i, j, l \in V \tag{5.29}$$

$$\sum_{k \in V} \sum_{l \in V} x_{ijkls} + y_{ijs} = 1, \qquad \forall i, j \in V \tag{5.30}$$

$$x_{ijkls} \geq 0, \qquad \forall i, j, k, l \in V \tag{5.31}$$

$$y_{ijs} \in \{0, 1\}, \qquad \forall i, j \in V. \tag{5.32}$$

The solutions for problems $\mathcal{P}_s$, $s \in S$, may render different network designs, i.e., distinct hubs selected as well as distinct allocations of terminals. Even, the number of hubs to which a terminal is assigned to may be different for one scenario to another. Moreover, problems $\mathcal{P}_s$ are also $\mathcal{NP}$-hard (they have the classical multiple allocation hub location problem as a particular case) and thus they can hardly be solved to optimality even using a specially tailored algorithm. Accordingly, we can also resort to heuristics in order to find good feasible solutions.

The algorithm we proposed next is based upon the idea that good solutions for the single-scenario problems $\mathcal{P}_s$, $s \in S$, may contain information about the good attributes of a good solution (possibly optimal) to $\mathcal{P}$. Furthermore, a well-known feature of the optimal solutions to stochastic programming problems is that they represent a trade-off between the solutions for the different scenarios (for the different realizations of the uncertainty).

These facts motivate our method, which aims at combining the information provided by the single-scenario problems to obtain a solution for $\mathcal{P}$. The full procedure, whose pseudo-code is shown in Algorithm 10, is performed until a previously defined number of iterations or time limit is reached. The final solution is the best found throughout the process.

Input: G, t_{max}, $itermax$
Initialize $\beta^* \leftarrow +\infty$ and $\beta_s \leftarrow \infty, \forall s \in S$
while t_{max} is not reached **do**
  foreach $s \in S$ **do**
    value of $(z, x, y)^s \leftarrow +\infty$
    for $iter \leftarrow 1$ **to** $itermax$ **do**
      Construct $(z, x, y)^s_{iter}$
      $\mathsf{LS}_{\mathsf{change}}((z, x, y)^s_{iter})$
      $\mathsf{LS}_{\mathsf{reduce}}((z, x, y)^s_{iter})$
      // Update the incumbent solution
      if value of $(z, x, y)^s_{iter}$ < value of $(z, x, y)^s$ **then**
        $(z, x, y)^s \leftarrow (z, x, y)^s_{iter}$
        update β_s;
  Obtain $(z, x, y)^{\mathcal{P}}$ from $(z, x, y)^s$, for all $s \in S$;
  if value of $(z, x, y)^{\mathcal{P}} < \beta^*$ **then**
    $\beta^* \leftarrow$ value of $\mathcal{P}$
    $(z, x, y)^* \leftarrow (z, x, y)^{\mathcal{P}}$
Output: $(z, x, y)^*$

Algorithm 10: Main loop of the algorithm to solve $\mathcal{P}$

5.3.1 A heuristic for the UrApHMP-NSS

In this section we describe the heuristic method devised to obtain feasible solutions to a problem $\mathcal{P}_s$, $s \in S$, which turns out to be a heuristic for the UrApHMP-NSS. It consists of a constructive phase followed by a local search phase.

5.3.1.1 Constructive phase

A solution to the UrApHMP-NSS is fully determined by (i) selecting the p hubs, (ii) allocating each terminal to at most r hubs, and (iii) transporting the traffics. Each of this components can be looked at as a subproblem. The constructive procedure (denoted by Construct $(z, x, y)^s_{iter}$ in Algorithm 10, line 6) is outlined in Algorithm 11.

Selecting p hubs

In order to determine the set H containing the p nodes that will be selected as hubs (lines 1 to 17 in Algorithm 11) we start by selecting q ($q < p$) nodes; afterwards we select the remaining $p - q$ nodes.

Initially, we compute the values

$$T_i = \sum_{j \in V} (t_{ij} + t_{ji}), \forall i \in V,$$

representing the total traffic originated and destined to each node $i \in V$. Defining $T = \sum_{i \in V} T_i$ we can compute the weights $w_i = \dfrac{T_i}{T} \in [0, 1]$ and such that $\sum_{i \in V} w_i = 1$. One node, say h, is randomly selected according to the values (probabilities) w_i and we set $H \leftarrow \{h\}$. For selecting the following $q - 1$ hubs, we update T as $T = \sum_{i \in V \setminus H} T_i$ and the weights $w_i, \forall i \in V \setminus H$, accordingly, and proceed as before until we get $|H| = q$. This way of selecting the first q hubs gives advantage to the nodes having the largest amount of traffics. The remaining $p - q$ hubs are now selected using a different criterion: for each $h \in H$, we compute

$$c^h_{\max} = \max_{i \in V \setminus H} \{\frac{c_{hi} + c_{ih}}{2}\} \quad \text{and} \quad c^h_{\min} = \min_{i \in V \setminus H} \{\frac{c_{hi} + c_{ih}}{2}\},$$

and define a threshold $\mathcal{T}$ as

$$\mathcal{T} = \frac{\sum_{h \in H} \left(c^h_{\max} + c^h_{\min}\right)}{|H|}.$$

Now, the set of terminals with average transporting cost to and from the hubs less than or equal to the computed threshold is defined:

$$D = \left\{ i \in V \setminus H : \frac{\sum_{h \in H} (c_{ih} + c_{hi})}{|H|} \leq \mathcal{T} \right\}.$$

Set D would correspond to those nodes that, if they were selected as hubs, the transportation costs between them and the hubs already in H would be small. This is motivated

Input: q, λ
// Choose the first q hubs
foreach $i \in V$ **do**
  $T_i \leftarrow \sum_{j \in V}(t_{ij} + t_{ji})$
$H \leftarrow \emptyset$
while $|H| < q$ **do**
  $T \leftarrow \sum_{i \in V \setminus H} T_i$
  $w_i \leftarrow T_i / T.\ i \in V \setminus H$
  select $h \in V \setminus H$ in $V \setminus H$ according to the discrete probability distribution induced by the values w_i, $i \in V \setminus H$
  $H \leftarrow H \cup \{h\}$
// Choose the remaining $p - q$ hubs
while $|H| < p$ **do**
  foreach $h \in H$ **do**
    $c^h_{\min} \leftarrow \min_{i \in V \setminus H}\{(c_{hi} + c_{ih})/2\}$ $c^h_{\max} \leftarrow \max_{i \in V \setminus H}\{(c_{hi} + c_{ih})/2\}$
  $\mathcal{T} \leftarrow \lambda \frac{\sum_{h \in H}(c^h_{\max} + c^h_{\min})}{|H|}$
  $D \leftarrow \left\{ i \in V \setminus H \mid \frac{\sum_{h \in H}(c_{ih} + c_{hi})}{|H|} \leq \mathcal{T} \right\}$
  select $h \in D$ according to a uniform distribution in $V \setminus H$
  $H \leftarrow H \cup \{h\}$
// Assign each terminal to a single hub
foreach $i \in V \setminus H$ **do**
  bestMeasure$(i) \leftarrow +\infty$;
  foreach $h \in H$ **do**
    Compute *Measure*(i, h)
    if *Measure*$(i, h) \leq$ *bestMeasure*(i) **then**
      bestMeasure$(i) \leftarrow$ *Measure*(i, h)
      $h^* \leftarrow h$;
  $H_i \leftarrow \{h^*\}$
// Increase of the terminals assignments
foreach $i \in V \setminus H$ **do**
  repeat
    Compute $c(T_i)$
    Let $h^* = \arg\min_{h \in H \setminus H_i}\{a_{ih}\}$
    Compute $c'(T_i)$
    if $c'(T_i) + a_{ih^*} \leq c(T_i)$ **then**
      $H_i \leftarrow H_i \cup h^*$
    else
      continue $\leftarrow$ FALSE;
  until $(|H_i| = r)$ or (continue is FALSE) ;
// Traffic transportation
Solve optimally the routing problem to obtain $(z, x, y)^s_{iter}$
Output: $(z, x, y)^s_{iter}$

Algorithm 11: Construct $(z, x, y)^s_{iter}$.

by the fact that most of the traffics of the network will traverse the arcs among the hubs; thus the inter-hubs transportation costs are desirably small. Once set D is defined, a first node h is selected at random from this set and inserted into H. If $|H| < p$, $c^h_{\max}$ and $c^h_{\min}$ are updated for all $h \in H$, $\mathcal{T}$ and D to choose another hub. Although q has been thought as a search parameter, we think that for the values of p we consider in our computational experiments ($3 \leq p \leq 5$), a fixed value of 2 for q is reasonable, and this is the value we have used in all the experiments.

Allocating terminals to hubs

We are interested in finding a good feasible solution as earliest as possible. This is guaranteed by allocating each terminal to (at least) a single hub. Accordingly, in order to find an assignment of terminals to hubs, we assume as a starting point that all the outgoing/ingoing traffics from/to a terminal are transported through a single hub.

With this purpose, we considered three different measures to help making a decision about the allocation allocating of a terminal i to a hub h (*Measure(i,h)*, line 22 in Algorithm 11).

The first measure is simply defined by the assignment cost a_{ih} of terminal i to hub h; the second one takes into account the unitary transportation cost. It is defined as follows:

$$\chi c_{ih} + \delta c_{hi} + \alpha \left(\frac{\sum_{\ell \in H \setminus \{h\}} c_{h\ell}}{|H| - 1} \right). \tag{5.33}$$

In the above expression, the third term is motivated by the fact that it is expected that most of the traffic will be routed through the hub subnetwork. In particular, we are considering the average unitary cost between hub h and all the other hubs. A drawback of measure (5.33) is that is does not take into account the cost associated with non-stop services.

The third measure attempted, overcomes this issue; it captures in a single value the promising features of the previous measures:

$$a_{ih} + \chi c_{ih} \sum_{j \in V} t_{ij} + \delta c_{hi} \sum_{j \in V} t_{ji} + T_i \alpha \left(\frac{\sum_{\ell \in H \setminus \{h\}} c_{k\ell}}{|H| - 1} \right). \tag{5.34}$$

Each terminal i is allocated to the hub k^* yielding the lowest value of the adopted measure. At this point, we have $|H_i| = 1, \forall i \in V \setminus H$.

After the initial allocation of terminals is performed, we check whether it compensates to increase the number of hubs to which each terminal is allocated. For every terminal $i \in V \setminus H$, define $c(T_i)$ as the cost associated with the transportation of all the traffics to and from i via H_i and through the hub network induced by H. It is evident that is convenient allocating i to another hub $h^* \in H \setminus H_i$ if $c'(T_i) + a_{ih^*} \leq c(T_i)$, where $c'(T_i)$ represents the cost of transporting all the traffics to and from i in the network defined by H, where $H_i := H_i \cup h^*$ and all the other subsets $H_j, \forall j \neq i$, have not been modified. Since computing all the possibilities is too much time consuming, not

all hubs $h^* \in H \setminus H_i$ are tried, only the hub h^* for which $a_{ih^*} = \min_{h \in H \setminus H_i}\{a_{ih}\}$. If $c'(T_i) + a_{ih^*} \leq c(T_i)$, we set $H_i := H_i \cup h^*$ and repeat the procedure for the same terminal i, while $|H_i| \leq r$; otherwise, we proceed with another terminal j. The whole allocation procedure is summarized in lines 18 to 37 in Algorithm 11.

Traffics transportation

Once the set of hubs is known, as well as the allocation of terminals to hubs, the problem of finding the optimal route for the traffics among any pair of nodes is solved by computing the shortest paths and considering non-stop services as well. Note that for a given pair of nodes i and j, the optimal route for the traffic t_{ij} may be different to the one associated with traffic t_{ji}.

When the optimal routes for sending the traffics have been computed, we have a feasible solution to the UrApHMP-NSS, or, taking into account the context of the stochastic problem, to a single-scenario problem $\mathcal{P}_s$, $s \in S$.

5.3.1.2 Improving a solution

Two local search procedures are devised for improving the solution obtained using the constructive algorithm. They correspond to lines 7 and 8 in Algorithm 10, and are based on changing the subsets H_i, $i \in V \setminus H$, as well as on reducing their size, for some terminals i. These two procedures (denoted by $\mathsf{LS}_{\mathsf{change}}$ and $\mathsf{LS}_{\mathsf{reduce}}$ in Algorithm 11), are described next.

Changing the assignments of terminals to hubs ($\mathsf{LS}_{\mathsf{change}}$)

Consider a terminal $i \in V \setminus H$ as well as the set of hubs, H_i, to which it is currently allocated. The procedure $\mathsf{LS}_{\mathsf{change}}$ iteratively explores the possibility of replacing one hub $\ell \in H_i$ by another hub $\ell' \in H \setminus H_i$. It should be noticed that in order to check whether such a move "improves" the current solution, we need to recompute the cost associated with the transportation of all the traffics involving node i, and not just those transported through hub ℓ. We start by computing

$$\mathcal{R}(i) = \frac{\frac{1}{|H_i|}\sum_{k \in H_i} a_{ik}}{T_i}, \quad i \in V \setminus H.$$

Note that the numerator of $\mathcal{R}(i)$ is the average allocation cost of i to its associated hubs. Hence, $\mathcal{R}(i)$ is a ratio between that cost and all the traffic to and from i. Therefore, we obtain a unitary traffic average cost involving terminal i. The values of $\mathcal{R}(i)$ are now sorted non-increasingly. This induces a sequence for nodes in $V \setminus H$ that we denote by $(i_1, \ldots, i_{|V \setminus H|})$. The improvement procedure takes the terminals iteratively, according to this sequence. For each terminal $i \in V \setminus H$, all the possible pairs (ℓ, ℓ'), with $\ell \in H_i$ and $\ell' \in H \setminus H_i$ are tested. The pseudo-code associated with this procedure is detailed in Algorithm 12.

Input: $(z, x, y)^s_{iter}$
continue $\leftarrow$ TRUE;
while continue *is* TRUE **do**
  continue $\leftarrow$ FALSE
  compute $\mathcal{R}(i), \forall i \in V \setminus H$
  find a sequence $(i_1, \ldots, i_{|V \setminus H|})$ induced by sorting the values $\mathcal{R}(i)$ $(i \in V \setminus H)$ non-increasingly
  foreach $j = 1, ..., |V \setminus H|$ **do**
    foreach $\ell \in H_{i_j}$ **do**
      foreach $\ell' \in H \setminus H_{i_j}$ **do**
        if (cost using ℓ') $<$ (cost using ℓ) **then**
          replace ℓ by ℓ' in H_{i_j}
          continue $\leftarrow$ TRUE
Output: $(z, x, y)^s_{iter}$

Algorithm 12: $\mathsf{LS}_{\mathsf{change}}$

Reducing the number of allocations to hubs ($\mathsf{LS}_{\mathsf{reduce}}$)

This procedure, which is detailed in Algorithm 13, aims at reducing the cardinality of some subsets H_i, $|H_i| \geq 2$, $i \in V \setminus H$, in case we conclude that this is advantageous from a cost perspective. In order to accomplish this, we compute

$$\mathcal{R}'(i) = \frac{\sum_{k \in H_i} a_{ik}}{T_i}$$

for all terminals $i \in V \setminus H$, which is the average allocation cost of i per traffic unit. As before, a sequence of terminals is induced by sorting the values $\mathcal{R}'(i)$ non-increasingly. The terminals are them analyzed according to this sequence.

For a terminal $i \in V \setminus H$ such that $|H_i| \geq 2$, we check the possibility of decreasing $|H_i|$ by choosing the hub $\ell \in H_i$ that appears the least in the paths associated with traffics T_i (that we denote by $Paths(i)$). If the total cost decreases by removing the allocation of node i to hub ℓ, then we do so. In order to check whether the costs decreases, we only need to recompute the paths in Paths(i^*) that make use of hub ℓ. Note that removing ℓ from H_i increases the transportation cost but decreases the total cost by $a_{i\ell}$. This process, that attempts to remove one allocation, is performed for all terminals in $V \setminus H$. Afterwards, $\mathcal{R}'(i)$ is recomputed for the terminals i for which the cardinality of H_i has been reduced and the procedure restarts but considering only such terminals. The process continues until no decrease in the cost can be achieved.

5.3.2 Constructing a feasible solution to the stochastic problem

In this section we introduce a mechanism that allows the combination of attributes from solutions to the single-scenario problems in order to build a feasible solution to the

Input: $(z, x, y)^s_{iter}$
continue ← TRUE;
while continue *is* TRUE **do**
  continue ← FALSE
  Compute $\mathcal{R}'(i), \forall i \in V \setminus H : |H_i| \geq 2$
  find a sequence $(i_1, \ldots, i_L)$ induced by sorting non-increasingly the values $\mathcal{R}'(i)$, $i \in V \setminus H : |H_i| \geq 2$
  foreach $j = 1, \ldots, L$ **do**
    Select $\ell \in H_{i_j}$ as the hub used less times in Paths(i_j)
    Compute $Cost$ as the cost of routing traffics T_{i_j} using H_{i_j}
    Compute $\overline{Cost}$ as the cost of routing traffics T_{i_j} using $H_{i_j} \setminus \{\ell\}$
    if $\overline{Cost} < Cost + a_{i_j \ell}$ **then**
      $H_{i_j} \leftarrow H_{i_j} \setminus \{\ell\}$
      continue ← TRUE
Output: $(z, x, y)^s_{iter}$

Algorithm 13: $\mathsf{LS}_{\mathsf{reduce}}$

overall problem $\mathcal{P}$.

We denote by $\overline{\beta} = (\beta_1, \ldots, \beta_{|S|})$ the vector containing the value of the best solution found for the single-scenario problems, $\mathcal{P}_s$, $s \in S$. Additionally, let z^s_{ik} be the values of the z-variables in the solution found for $\mathcal{P}_s$, $s \in S$. Furthermore, consider the $|V| \times |V|$ matrix, denoted by $\mathcal{U}$, whose generic entry is $u_{ij} = \pi_1 z^1_{ij} + \pi_2 z^2_{ij} + \ldots + \pi_{|S|} z^{|S|}_{ij} = \sum_s \pi_s z^s_{ij}$, $i, j \in V$. Let U_j be the j-th column of $\mathcal{U}$ ($j \in V$). Taking into account that $z_{ij} = 1$ if terminal i is assigned to hub j, and u_{ij} contains the same information averaged across the different scenarios, when we examine matrix $\mathcal{U}$ we have to consider that each row i represents a terminal node and each column j a potential hub.

In order to quantify the "attractiveness" of a node to become a hub in the solution for the overall problem $\mathcal{P}$, we compute the marginal vector, $\overline{u}$, resulting from summing all rows of $\mathcal{U}$, i.e., a vector whose generic component is $\overline{u}_j = \sum_{i \in V} u_{ij}$, $j \in V$.

The starting point for building a feasible solution to $\mathcal{U}$ is to randomly select a scenario s^* according to the probabilities $\pi_1, \ldots, \pi_{|S|}$. Then, a process inspired on the Path Relinking methodology ([44]) that gradually transforms an initiating solution by incorporating to it attributes of some guiding solutions is devised. In particular, we propose here to consider the attributes of the best solutions obtained in each scenario averaged according to their probability. This information is contained in matrix $\mathcal{U}$. The steps of the process can be summarized as follows:

(i) Consider the set of p hubs, H_{s^*}, in the feasible solution obtained for $\mathcal{P}_{s^*}$.

(ii) Find the p indices associated with the larges values of $\overline{u}_j$ for $j \in V \setminus H_{s^*}$.

(iii) Denote by J=(j_1, ..., j_p) a sequence of the indices found in (ii) resulting from sorting the corresponding values of $\overline{u}_j$ non-increasigly.

(iv) Set $\ell = j_1$.

(v) For each $h \in H_{s^*}$,

(a) denote by $H'_{s^*}(h)$ the set of hubs resulting from replacing h by ℓ;

(b) for each $k \in H'_{s^*}(h)$ and for each $i \in V$, if $u_{ik} > 0$ set $z_{ik} = 1$, i.e., allocate node i to hub k.

(c) for each node $i \in V$ check whether the allocations set for the terminal are feasible. If not, repair them. Three cases can be distinguished:

c1. terminal i has been assigned to more than r hubs (i.e., $|H_i| > r$). In this case, the $|H_i| - r$ assignments of i to the hubs in $H'_{s^*}(h)$ having the smallest values u_{ij} are discarded by setting the corresponding z-variables to 0.

c2. terminal i has not been assigned to any hub (i.e., $|H_i| = 0$). This may be the case of a node i assigned only to hub h. The assignment of i to the existing hubs is evaluated at each scenario using the measure (5.34) described in Section 5.3.1.1, and the number of times a hub is identified as the best for i is calculated. The hub k appearing most is the one selected for allocating i to.

c3. node i is a hub and was allocated to other hubs. In this case, all the assignments of i to other hubs are removed.

(d) consider $H'_{s^*}(h)$ as the set of operating hubs together with the allocations z_{ik} resulting from (b) and (c). For each scenario $s \in S$ do the following:

1. compute the cost under scenario s, if we implement such feasible network design for that particular scenario. Denote that cost by $\hat{c}^s_h$.

2. compute the value $r^s_h = \frac{\hat{c}^s_h - \beta_s}{\beta_s}$. This value represents a sort of relative "regret" if the network design implemented in scenario s is the one induced by $H'_{s^*}(h)$ instead of the best network design known so far for that scenario. (Recall that the incumbent upper bound on the optimal value for scenario s is β_s.).

3. compute $r^{\max}_h = \max_{s \in S}\{r^s_h\}$ as the maximum relative "regret" across all scenarios if we take the network design induced by $H'_{s^*}(h)$ with the allocations resulting from (b) and (c).

(vi) p sets of p nodes result from (v), since ℓ is replacing in turn every hub $h \in H_{s^*}$. The next step is to compare those sets of hubs and decide for one of them. To do so, we use the values $r^{\max}_h$ computed in (v)(d)(3.). In particular, we select the hub set $H_{s^*}(h')$ such that

$$h' \in \arg\min_{h \in H_{s^*}} \{r^{\max}_h\}.$$

(vii) Setting the network design induced by $H_{s^*}(h')$ as explained above for the overall problem $\mathcal{P}$, we can now easily solve the resulting transportation problem and eventually get a complete feasible solution to the stochastic problem. Accordingly,

we should update the best incumbent solution for $\mathcal{P}$ if the cost found is smaller than the cost of the incumbent (in case some already exists).

(viii) This mechanism proceeds now by setting $\ell = j_2$ and analyzing the $p-1$ ways of replacing a column in $H_{s^*} \setminus \{j_1\}$ by ℓ.

(ix) When, finally, we set $\ell = j_p$ there is only one possible replacement in $H_{s^*} \setminus \{j_1, \ldots, j_{p-1}\}$ which is the only one attempted.

(x) The best solution for $\mathcal{P}$ is updated each time a new solution with less cost is obtained.

The above mechanism can (and should) be repeated several times since the starting point is randomized.

The steps (i)–(x) above, define the heuristic procedure we propose for obtaining feasible solutions to the stochastic UrApHMP-NSS. Note that if in step (v)(d) we find a negative "regret" for some scenario, this means that we have found a feasible solution under that scenario better than the best one known so far. In this case, we should update the corresponding value of β_s in accordance to the new upper bound found.

It is worth noticing that the above presented scheme is quite flexible. In fact, it is modular in the sense that some parts can simply be replaced without the need of changing the global structure. For instance, if a different approach is considered for tackling the single-scenario problems, the above structure can be adopted exactly as presented.

5.4 Computational experiments

In this section we describe the characteristics of the instances tested and the computational results obtained with the above proposed algorithm.

5.4.1 Test instances

Since the Stochastic UrApHMP-NSS has been introduced in this paper, there are no benchmark instances available. Hence, we have generated a set of instances in order to evaluate the behavior of our new method. To generate instances for the Stochastic UrApHMP-NSS we have followed a similar reasoning as the one proposed by Alumur et al. in [7] taking as starting point the CAB25 data set. From the original CAB data file, which contains information on distances and traffics between 25 major cities in the USA, we have generated a total of 74 instances with:

- $|V| \in \{15, 20, 25\}$;
- $p \in \{3, 4, 5\}$;
- $r \in \{2, \ldots, p-1\}$.

As it is often done when using the CAB data, the traffics are scaled (by dividing them by the total traffic) so that the total demand is equal to 1. The values for parameters χ and δ, representing the unit rates for collection (origin-hub) and distribution (hub-destination), are set to 1. Parameter α (the unit rate for transfer hub-hub) takes values in the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. The other parameters are generated according to the following:

- From the traffics τ_{ij} in the original CAB instances, we have generated traffics t_{ij} for each scenario as follows:

  (i) a number w is randomly generated in the interval $[1, 100]$;

  (ii) if $w \leq 66$, t_{ij} is randomly selected in the interval $[0.01\tau_{ij}, 5\tau_{ij}]$; otherwise, t_{ij} is randomly selected in $]5\tau_{ij}, 10\tau_{ij}]$.

- The assignment costs, a_{ik}, $i, k \in V$ have been randomly generated in the interval $[10 \log \sum_{j \in V} t_{ij}, 20 \log \sum_{j \in V} t_{ij}]$.

- For each scenario, the costs c_{ij}, have been randomly generated in the interval $[0.5\gamma_{ij}, 1.5\gamma_{ij}]$, where γ_{ij} $(i, j \in V)$ represent the original costs.

- The non-stop transportation costs d_{ij} have been randomly selected in the interval $[0.2(\chi + \alpha + \delta)\gamma_{ij}, 0.8(\chi + \alpha + \delta)\gamma_{ij}]$.

- The costs b_{ij} have been randomly chosen in $[10 \log(t_{ij} + t_{ji}), 20 \log(t_{ij} + t_{ji})]$.

Nine scenarios have been considered for all the instances, and the probabilities π_s associated with the scenarios have been randomly generated.

5.4.2 Computational results

In this section we report the computational results obtained with the method we proposed in Section 5.3 for solving the Stochastic UrApHMP-NSS. The procedure has been implemented in C. The results of the proposed method reported in this section have been obtained with an Intel core i7–3770 at 3.40GHz using a single thread and 16GB of RAM, under Ubuntu 14.04.03 GNU/Linux – 64 bits operating system, while those corresponding to CPLEX have been obtained with a SGI Altix UV 1000[1] system with 20 processors (out of 64) at 2.67 GHz, 120 cores, and 120GB of RAM.

The heuristic for a single scenario A relevant component in the heuristic procedure described in Section 5.3.2 is the algorithm proposed for the deterministic (single-scenario) problems. For this reason, we start by analyzing the quality of the feasible solutions obtained by that algorithm. The information can be found in Table 5.1, where each row contains average results for the 9 single-scenario instances generated for each combination of n, p, r, α.

[1] Supercomputer Vives del Servei d'Informàtica de la Universitat de València

Table 5.1. Comparison of the heuristic for the UrApHMP-NSS with CPLEX on some CAB instances.

				CPLEX	Construct				Construct + $\mathsf{LS}_{\mathsf{change}}$				Construct + $\mathsf{LS}_{\mathsf{change}}$ + $\mathsf{LS}_{\mathsf{reduce}}$			
				Opt.	100 iter		1000 iter		100 iter		1000 iter		100 iter		1000 iter	
n	p	r	α	CPU	Dev(%)	CPU	Dev(%)	CPU	Dev(%)	CPU	Dev(%)	CPU	Dev(%)	CPU	Dev(%)	CPU
15	4	2	1.00	2.17	56.1	0.00	53.6	0.03	53.6	0.02	50.9	0.18	4.2	0.02	1.5	0.19
15	4	2	0.80	2.32	56.8	0.00	52.6	0.03	55.1	0.02	51.4	0.17	3.8	0.02	1.7	0.18
15	4	2	0.60	2.22	57.6	0.00	51.2	0.03	56.4	0.02	50.2	0.17	3.7	0.02	1.6	0.17
15	4	2	0.40	1.36	58.3	0.00	51.1	0.03	56.8	0.02	50.1	0.16	4.1	0.02	1.7	0.15
15	4	2	0.20	1.22	59.6	0.00	53.5	0.03	58.9	0.01	52.9	0.15	4.2	0.01	1.3	0.15
20	4	2	1.00	56.68	39.1	0.00	36.1	0.05	33.1	0.04	30.9	0.40	4.5	0.04	2.2	0.39
20	4	2	0.80	64.02	39.3	0.01	36.4	0.05	34.6	0.04	31.7	0.37	3.3	0.04	1.6	0.36
20	4	2	0.60	44.91	42.0	0.01	37.0	0.05	38.3	0.04	34.1	0.35	4.1	0.03	1.4	0.34
20	4	2	0.40	26.33	44.0	0.01	36.7	0.05	41.7	0.03	35.4	0.31	3.8	0.03	1.2	0.31
20	4	2	0.20	12.13	44.4	0.01	38.2	0.05	42.7	0.03	37.1	0.28	3.6	0.03	0.8	0.28
20	5	2	1.00	66.94	36.5	0.01	34.4	0.05	29.9	0.05	28.6	0.52	3.7	0.05	2.3	0.50
20	5	2	0.80	64.94	37.8	0.01	34.4	0.05	32.7	0.05	30.1	0.49	3.4	0.05	1.4	0.48
20	5	2	0.60	36.77	41.7	0.01	37.9	0.05	37.5	0.05	33.8	0.45	4.7	0.05	1.8	0.44
20	5	2	0.40	16.69	46.0	0.00	39.5	0.05	43.5	0.04	37.1	0.41	5.8	0.04	2.1	0.40
20	5	2	0.20	8.75	49.7	0.00	42.1	0.05	48.6	0.04	41.3	0.37	6.1	0.03	2.1	0.35
25	5	2	1.00	701.98	40.7	0.01	37.0	0.07	34.3	0.09	32.6	0.87	4.6	0.09	2.6	0.86
25	5	2	0.80	454.14	43.5	0.01	40.8	0.07	38.4	0.08	36.5	0.81	4.2	0.08	2.9	0.82
25	5	2	0.60	116.36	47.0	0.01	43.1	0.07	43.6	0.08	40.3	0.76	4.3	0.08	2.8	0.76
25	5	2	0.40	56.85	50.2	0.01	46.4	0.07	47.8	0.07	44.7	0.69	3.9	0.07	2.3	0.69
25	5	2	0.20	28.37	54.4	0.01	49.8	0.07	53.1	0.06	49.0	0.63	4.6	0.06	2.9	0.62
Average				88.26	47.2	0.01	42.6	0.05	44.0	0.04	39.9	0.43	4.2	0.04	1.9	0.42

In this table, the column headed with "CPLEX" contains the average CPU time (seconds) required to solve the instances to optimality using this solver. We then observe 3 groups of columns (4 columns each). The first group contains the results (percentage deviation w.r.t. optimum and CPU time when 100 or 1000 iterations were performed) for the feasible solutions provided by Algorithm 11. The second and third groups of columns contain the results obtained when Algorithm 12 (second group) and Algorithm 12 followed by 13 (third group) were performed for improving the initial feasible solution.

Observing Table 5.1 we conclude that the complete procedure (Algorithms 11–13) is a very efficient tool for obtaining high-quality solutions to the (deterministic) UrApHMP-NSS. In particular, we observe that running the whole procedure 1000 times renders extremely sharp upper bounds at the expenses of a negligible increase in the computational effort. The results obtained indicate that Algorithm 10 is a good element to put at the core of the overall procedure for the stochastic problem.

Table 5.2. Computational results on the 74 CAB instances.

			CPLEX	Heur 1000		Heur 2000	
n	p	# inst	CPU	Dev (%)	CPU	Dev (%)	CPU
15	3	5	271.5	1.0	9.5	1.0	18.8
15	4	10	65.3	3.2	18.2	2.2	36.1
15	5	15	44.7	1.9	29.6	1.3	58.9
20	3	5	16261.2	1.6	18.8	1.2	37.2
20	4	10	10479.9	2.2	37.0	1.4	72.7
20	5	15	10080.7	2.9	61.9	1.8	123.2
25	5	14*	129615.7	3.9	104.3	3.8	207.2
Summary		74	29116.4	2.6	47.7	2.0	94.6

The heuristic for the stochastic UrApHMP-NSS Table 5.2 reports the results obtained using the procedure developed in Section 5.3.2 for the Stochastic UrApHMP-NSS. The information presented in each row corresponds to the average values obtained for all the instances with the characteristics described. First two columns show the number of nodes of the group of instances and the number of hubs to be open, respectively. As mentioned before, one instance was generated for each value of p, $r \in \{2, \ldots, p-1\}$, and $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. The number of instances in each group is shown in column "# inst". For $n = 25$ and $p = 5$ we report results for 14 out of the 15 CAB instances considered. This is marked with "$\star$". The omitted instance (corresponding to $r = 4$ and $\alpha = 0.6$), has been removed from the table since CPLEX was unable of providing even a lower bound after 5 days of computing time.

All the instances reported have been solved to optimality with CPLEX and heuristically with our algorithm. The CPU time (seconds) required by CPLEX as well as by two alternatives for our method (1000 and 2000 iterations in total) and the average per-

centage deviation with respect to the optimal solution are reported in columns "CPU" and "Dev (%)".

The results obtained give strong evidence to the high efficiency of the new heuristic proposed. In particular, we can observe an average deviation of 2.6% after 1000 runs of the procedure in less than 50 seconds (on average), while the average deviation reduces to 2% when 2000 runs are performed which is accomplished in less that 100 seconds. Additionally, it is worth noticing that for 1000 runs, a deviation smaller than 1% was obtained for 15 out of the 74 instances, and a maximum deviation of 7.1% was observed; when 2000 runs were performed, the method led to a deviation smaller than 1% in 23 out of the 74 instances, while the maximum deviation obtained was 5.5%. Note that the average CPU time required by CPLEX for solving the instances in the last group is approximately 36 hours, while in some of the instances it takes up to 5 days to find the optimal solution with the SGI Altix UV 1000 system. The detailed results (for each instance tested) are reported in Tables 5.3 and 5.4.

One important element in the heuristic algorithm proposed in Section 5.3.2 for the Stochastic UrApHMP-NSS concerns the vector $\beta_1, \ldots, \beta_{|S|}$ that contains the best known upper bound for each of the $|S|$ single-scenario problems. During the process, the changes performed in the solutions to the stochastic problem may allow finding better solutions for some single-scenario problems, which leads to changes in the above vector. In Figure 5.1 we represent the evolution of the 9 βs, each one for one scenario, considered in the instance with $n = 25$, $p = 5$, $r = 2$, and $\alpha = 1.00$. In this figure, we also represent the evolution of the value of the best known solution to the stochastic problem. The figure shows that changing the values of the β is something that the process takes advantage from. Moreover, at least for this instance, we can observe larger improvements in an earlier stage of the process, which also indicates that the overall procedure seems to be effective not only when it comes to finding a good solution to the stochastic problem, but also in terms of sharpening the best upper bounds known for the single-scenario problems.

The revelance of the stochastic modeling framework Finally, we conclude the analysis of the results by evaluating the relevance of considering the stochastic modeling framework proposed for the UrApHMP-NSS instead of using a simplified deterministic model. We accomplish this analysis by computing the so-called Expected Value of Perfect Information (EVPI) that quantifies the amount that the decision maker would be willing to pay to access the perfect future information. A high EVPI indicates that the decision maker perceives as quite relevant having access to the perfect information which indicates that uncertainty may be a relevant factor in the problem. The EVPI is obtained as follows:

- First, the Wait-and-See value is computed according to $WS = \sum_{s \in S} \pi_s \mathcal{V}_s$, where $\mathcal{V}_s$ is the optimal value of problem $\mathcal{P}_s$.
- Second, the stochastic problem $\mathcal{P}$ is solved to optimality. Let $\mathcal{V}$ its value.
- Finally the EVPI is computed as: $\text{EVPI} = \mathcal{V} - WS$.

Table 5.3. Detailed results for the CAB instances with $n = 15$.

				CPLEX		Heur 1000			Heur 2000		
n	p	r	α	Value	CPU (scs)	Value	Dev (%)	CPU (scs)	Value	Dev (%)	CPU (scs)
			0.2	2782.5	50.6	2842.3	2.1	8.5	2842.3	2.1	16.9
			0.4	2920.2	90.4	2962.9	1.5	9.0	2958.0	1.3	17.8
15	3	2	0.6	3057.8	362.6	3057.9	0.0	9.6	3057.9	0.0	18.9
			0.8	3187.1	668.4	3187.1	0.0	10.0	3187.1	0.0	19.8
			1.0	3233.6	185.7	3278.8	1.4	10.5	3278.8	1.4	20.6
			0.2	2393.1	36.4	2514.4	5.1	12.9	2410.8	0.7	25.6
			0.4	2542.0	44.3	2542.0	0.0	13.9	2542.0	0.0	29.4
15	4	2	0.6	2690.8	77.0	2754.7	2.4	15.0	2754.7	2.4	29.5
			0.8	2839.6	44.9	2941.2	3.6	15.8	2845.6	0.2	31.3
			1.0	2957.6	187.0	3077.2	4.0	16.6	3048.8	3.1	32.8
			0.2	2393.1	35.7	2522.8	5.4	19.1	2522.8	5.4	38.0
			0.4	2542.0	41.6	2542.0	0.0	20.5	2542.0	0.0	40.9
15	4	3	0.6	2690.8	30.9	2807.5	4.3	21.7	2786.3	3.5	43.1
			0.8	2839.6	35.2	2961.7	4.3	23.5	2961.7	4.3	44.7
			1.0	2957.6	119.8	3029.9	2.4	23.2	3029.9	2.4	46.0
			0.2	2119.8	29.8	2128.7	0.4	15.9	2128.7	0.4	31.9
			0.4	2274.8	29.3	2274.8	0.0	17.3	2274.8	0.0	34.3
15	5	2	0.6	2429.8	27.8	2459.7	1.2	20.8	2459.7	1.2	51.4
			0.8	2584.8	30.6	2584.8	0.0	20.2	2584.8	0.0	41.3
			1.0	2733.8	61.4	2760.6	1.0	21.1	2736.2	0.1	42.0
			0.2	2119.8	32.8	2194.8	3.5	29.4	2161.2	2.0	56.7
			0.4	2274.8	38.2	2365.3	4.0	31.5	2274.8	0.0	61.3
15	5	3	0.6	2429.8	69.5	2459.7	1.2	34.2	2459.7	1.2	65.7
			0.8	2584.8	41.0	2627.6	1.7	35.0	2627.6	1.7	69.3
			1.0	2733.8	106.4	2814.9	3.0	36.2	2794.6	2.2	71.6
			0.2	2119.8	29.5	2179.4	2.8	32.8	2162.4	2.0	64.0
			0.4	2274.8	24.6	2325.5	2.2	34.7	2325.1	2.2	68.9
15	5	4	0.6	2429.8	47.4	2510.6	3.3	37.1	2474.1	1.8	72.6
			0.8	2584.8	36.2	2635.7	2.0	38.7	2635.7	2.0	75.0
			1.0	2733.8	66.0	2805.1	2.6	38.8	2796.1	2.3	76.9

Table 5.4. Detailed results for the CAB instances with $n = 20$ and 25.

				CPLEX		Heur 1000			Heur 2000		
n	p	r	α	Value	CPU (scs)	Value	Dev (%)	CPU (scs)	Value	Dev (%)	CPU (scs)
			0.2	5970.6	90.0	5998.07	0.5	16.3	5998.07	0.5	32.2
			0.4	6353.9	481.7	6495.68	2.2	17.7	6495.68	2.2	35.1
20	3	2	0.6	6737.2	4558.3	6737.15	0.0	19.1	6737.15	0.0	37.5
			0.8	7120.4	51470.9	7250.2	1.8	20.1	7120.44	0.0	39.8
			1.0	7435.7	24705.1	7690.44	3.4	21.0	7690.44	3.4	41.5
			0.2	5135.8	276.0	5135.81	0.0	25.6	5135.81	0.0	50.3
			0.4	5705.3	762.1	5740.23	0.6	28.3	5740.23	0.6	56.0
20	4	2	0.6	6214.0	4745.1	6350.84	2.2	31.4	6242.07	0.5	61.0
			0.8	6654.4	18544.3	6801.44	2.2	33.5	6729.13	1.1	65.1
			1.0	7037.4	29172.6	7165.38	1.8	35.1	7165.38	1.8	69.0
			0.2	5135.8	199.9	5135.81	0.0	37.6	5135.81	0.0	74.1
			0.4	5705.3	1229.2	5771.89	1.2	40.9	5771.89	1.2	80.7
20	4	3	0.6	6214.0	5750.7	6377.65	2.6	43.4	6248.07	0.5	86.0
			0.8	6654.4	24547.4	6959.13	4.6	46.2	6863.92	3.1	90.8
			1.0	7037.4	19571.9	7492.79	6.5	48.2	7398.64	5.1	93.8
			0.2	4499.1	93.1	4648.9	3.3	32.5	4499.1	0.0	64.7
			0.4	5126.4	98.3	5234.8	2.1	36.8	5126.4	0.0	72.8
20	5	2	0.6	5753.8	2443.2	5826.4	1.3	40.3	5826.4	1.3	80.9
			0.8	6304.2	16982.8	6381.2	1.2	43.6	6381.2	1.2	88.6
			1.0	6696.2	34889.4	6947.0	3.7	45.9	6903.6	3.1	91.3
			0.2	4499.1	72.1	4507.8	0.2	59.3	4507.8	0.2	117.4
			0.4	5126.4	78.8	5260.6	2.6	65.5	5260.6	2.6	129.8
20	5	3	0.6	5753.8	2875.6	5826.4	1.3	71.0	5818.3	1.1	141.4
			0.8	6304.2	19990.1	6471.3	2.7	75.0	6381.2	1.2	151.3
			1.0	6696.2	33709.4	7028.3	5.0	78.7	6894.7	3.0	156.4
			0.2	4499.1	168.0	4507.8	0.2	66.2	4507.8	0.2	134.5
			0.4	5126.4	211.0	5305.3	3.5	72.0	5214.5	1.7	142.9
20	5	4	0.6	5753.8	1214.8	6019.3	4.6	78.1	5959.0	3.6	151.7
			0.8	6304.2	25392.8	6623.8	5.1	81.0	6483.2	2.8	159.0
			1.0	6696.2	12991.4	7119.9	6.3	83.5	7039.9	5.1	164.7
			0.2	5313.3	11385.8	5528.21	4.0	56.4	5528.21	4.0	111.7
			0.4	5961.2	17156.0	6140.71	3.0	63.2	6140.71	3.0	125.2
25	5	2	0.6	6568.8	124321.0	6796.78	3.5	68.7	6796.78	3.5	136.4
			0.8	7159.2	221453.3	7365.45	2.9	74.2	7365.45	2.9	147.3
			1.0	7735.8	415802.1	8122.09	5.0	78.5	8099.9	4.7	156.3
			0.2	5313.3	30176.6	5505.8	3.6	102.5	5505.8	3.6	203.9
			0.4	5961.2	54987.0	6076.01	1.9	112.7	6076.01	1.9	226.9
25	5	3	0.6	6568.8	25439.3	6917.3	5.3	121.0	6908.77	5.2	240.4
			0.8	7159.2	137703.5	7436.9	3.9	129.0	7436.9	3.9	255.2
			1.0	7735.8	306435.3	8040.74	3.9	135.1	8040.74	3.9	267.7
			0.2	5313.3	7166.7	5470.89	3.0	114.6	5470.89	3.0	227.7
			0.4	5961.2	33398.2	6113.35	2.6	123.6	6113.35	2.6	246.1
25	5	4	0.8	7159.2	217706.7	7539.48	5.3	136.5	7539.48	5.3	272.5
			1.0	7735.8	211488.1	8287.62	7.1	144.3	8162.45	5.5	282.9

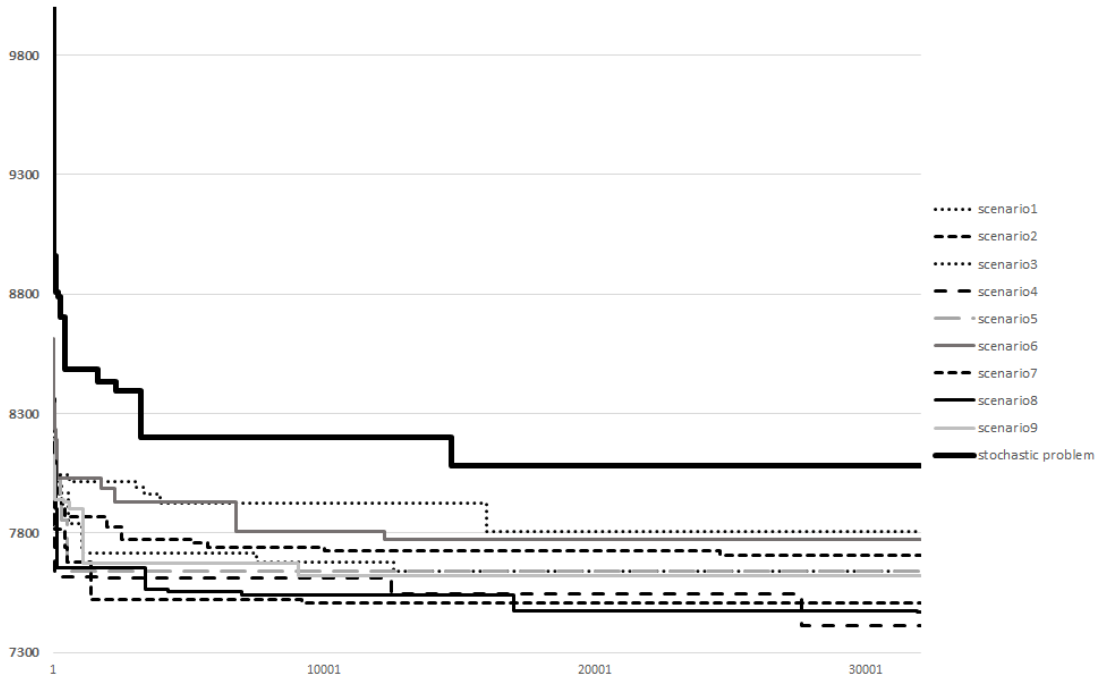

Figure 5.1. Beta evolution

A more informative measure is the relative EVPI: $100\frac{\mathcal{V}-WS}{\mathcal{V}}$, since the corresponding result (in percentage) ignores the magnitude of the values involved in the problem. The results obtained for the test instances we have considered can be observed in Table 5.5. A first aspect emerging from this table is the increase of the relative EVPI with α. This is not surprising since small values of α induce small inter-hub costs, which makes the impact of the decisions associated with the hubs smaller and thus the relevance of the uncertainty also decreases.

Overall, we observe percentages that are always positive and ranging up to 5.19%. Taking into account that we are working with fairly small instances (up to 25 nodes) this values show that considering stochasticity in our problem may be of great relevance.

5.5 Concluding remarks

In this chapter we have investigated a stochastic version of the r-allocation p-hub median problem with non-stop services. We have started by extending the already existing r-allocation p-hub median problem in order to capture non-stop services. Afterwards we have developed a stochastic programming modeling framework for the problem. Due to the difficulty in solving the problem to optimality, we have derived a heuristic approach for the new stochastic problem. A side contribution is the development of a heuristic approach for the deterministic (single-scenario) version of the problem. We have performed computational tests using instances generated from the well-known CAB data

Table 5.5. Exact Expected Value of the Perfect Information for some CAB instances.

n	p	r	α	Wait-and-see optimal value	Optimal value of the Stochastic UrApHMP-NSS	Relative EVPI
15	4	2	0.2	2386.3	2393.1	0.28%
15	4	2	0.4	2533.1	2542.0	0.35%
15	4	2	0.6	2678.4	2690.8	0.46%
15	4	2	0.8	2795.4	2839.6	1.56%
15	4	2	1.0	2878.4	2957.6	2.68%
20	4	2	0.2	5078.3	5135.8	1.12%
20	4	2	0.4	5584.6	5705.3	2.12%
20	4	2	0.6	6050.5	6214.0	2.63%
20	4	2	0.8	6430.6	6654.4	3.36%
20	4	2	1.0	6672.3	7037.4	5.19%
20	5	2	0.2	4458.4	4499.1	0.91%
20	5	2	0.4	5053.2	5126.4	1.43%
20	5	2	0.6	5619.3	5753.8	2.34%
20	5	2	0.8	6077.5	6304.2	3.60%
20	5	2	1.0	6366.5	6696.2	4.92%
25	5	2	0.2	5256.0	5313.3	1.08%
25	5	2	0.4	5877.9	5961.2	1.40%
25	5	2	0.6	6443.6	6569.8	1.92%
25	5	2	0.8	6974.6	7159.2	2.58%
25	5	2	1.0	7469.8	7735.8	3.44%
Average				5134.2	5264.5	2.17%

set. The results show the effectiveness of our new heuristic for obtaining high-quality feasible solutions to the problem with a small CPU time.

One important aspect of our heuristic is its modularity. For instance, in case a different algorithm is devised for finding feasible solutions to the single-scenario problems, the methodology described in Section 5.3.2 is still valid. Another possibility is to change the attribute matrix $\mathcal{U}$. Again, the procedure would be valid as presented.

The high quality of the results obtained in this work encourages the application of the same type of heuristic methodology to other stochastic hub location problems and even to consider more comprehensive models from a practical point of view.

Chapter 6

General conclusions and future research directions

In this thesis we have studied the following variants of hub location problems in the context of transportation networks:

- The capacitated single assignment hub location problem with modular link capacities (Chapter 2).
- The uncapacitated r-allocation p-hub median problem (Chapter 3).
- The uncapacitated r-allocation p-hub equitable center problem (Chapter 4).
- The stochastic uncapacitated r-allocation p-hub median problem with non-stop services (Chapter 5).

We have proposed solution algorithms based on metaheuristic methodologies to solve these $\mathcal{NP}$-hard problems. The proposed algorithms are based on construction procedures that find feasible solutions and on local improvement procedures aiming to improve the solutions found.

As it has already been discussed throughout this thesis, hub location problems can be understood as a combination of three optimization subproblems that are interconnected: a facility location problem, an assignment problem, and a transportation problem. We have proposed constructive algorithms for each problem studied here. All of them follow a logical three-steps structure: finding the hubs needed to obtain a feasible solution, assigning terminals to the hubs, and transporting traffics through the hub network. Depending on the variant in hand, each of these steps is performed following heuristic rules that are based on strategies that exploit the particular characteristics of the variant, with the exception of the transportation step that we always solve to optimality.

We have also designed some local search procedures to improve the solutions found through the exploration of their neighborhoods. Regarding the local optimizer methods designed for the facility location subproblem, they often explore the solution space in a more extensive way, but the evaluation of each neighbor solution is computationally

costly. From the experiments carried out along this thesis, we have learned that there is a need of finding neighborhoods that are easy to explore in this context as, for example, LS_{cluster} procedure presented in Section 2.5.2 for the CSHLPMLC. This procedure can efficiently explore the neighborhood when we make changes on the set H of hubs. For problems where we do not know an efficient way to perform this exploration, other heuristic rules inducing an order for scanning the possibilities of exploration are also very useful.

Regarding the neighborhoods of the assignment problem of terminals to hubs, it seems natural to explore them after changing the set H and not before. The procedures designed to explore these neighborhoods tend to find good solutions, but with a rate of improvement lower than the ones produced by the exploration of neighborhoods based on changing the set of hubs. In the variants where capacity constraints are involved, the neighborhoods based on "exchanges", opposite to the ones based on "insertions", work quite well because the latter often produce infeasible solutions when trying to insert terminals to hubs where there is no capacity left for acceptance of new assignments. Neighborhoods based on exchanging several terminals at the same time help terminals to be allocated to other hubs without violating too many times the capacity constraints.

The methodologies we have used to solve these hub location problems include GRASP, that has proved to be easy to implement and, at the same time, capable of exploring efficiently the solution space. Using GRASP as a general framework, we have been able to particularly adapt it to each problem variant and still be able to find different feasible solutions serving as starting points for local optimization search methods. We would also like to highlight the methodologies based on tabu search principles, such as the strategic oscillation, the adaptive memory programming, and the path relinking. By their own, and in combination with other methods, these methodologies have resulted to be good frameworks for an effective exploration of the solution space.

Throughout this thesis we have performed several experiments that have shown that general purpose solvers are a good tool to solve hub location problems when the size of the instance to be solved is small. For real sizes instances, these solvers have proven to be ineffective, mainly because they need a large amount of computational resources. Hence, we have proposed specially tailored heuristic algorithms for each HLP variant. They can be deployed and run on desktop computers, and are based on paradigms of artificial intelligence rather than on (generically much slower) mathematical programming techniques. Very good results have been found by the exploration of the solution space using these heuristics, with the drawback that we cannot guarantee neither the optimality nor the quality of the solutions found. Nevertheless, the heuristics proposed here should be also used, given their small computational cost, to help exact procedures to try to obtain the optimal solutions.

There are several hub location problems that we would like to explore in the future as, for example, some variants where it is not assumed that the hubs form a complete subnetwork and, therefore, the decision regarding the use of a connection between a pair of hubs is also a variable of the problem. These variants are known in the Literature as "incomplete hub location problems". We also believe that optimization problems

combining location and vehicle routing decisions may model many real situations that are nowadays being modeled and solved separately, thus producing suboptimal solutions. This field is known in the Literature as "Location–Routing", and we would also like to explore it.

Finally, since hub location problems involving stochasticity have not been much explored, we think that there is a need of studying more variants and solution methods. These problems are very difficult due to the huge amount of variables they contain and because their solutions depend on how the future is presented. Nevertheless, we believe that stochastic models reflect much better real-life problems, where changes in cost or demands, for example, vary from one time period to the next and, therefore, deterministic models do not describe well the changing future.

Derived works

Some of the main contributions appearing in this dissertation have been submitted or accepted for publication. They are the following:

- Corberán, Á., Peiró, J., Campos, V., Glover, F., and Martí, R.
  Strategic oscillation for the capacitated hub location problem with modular links.
  Journal of Heuristics, 22 (2): 221 – 244, 2016.

- Hoff, A., Peiró, J., Corberán, Á., and Martí, R.
  Adaptive memory programming for solving the capacitated hub location problem with modular link capacities.
  Universitat de València Technical report, June 2016. Submitted.

- Martí, R., Corberán, Á., and Peiró, J.
  Scatter search for an uncapacitated p-hub median problem.
  Computers & Operations Research, 58: 53 – 66, 2015.

- Martí, R., Corberán, Á., and Peiró, J.
  The scatter search methodology: An experimental evaluation on hub location problems.
  To appear in *Handbook of Heuristics*. Springer International Publishing.

- Peiró, J., Corberán, Á., Laguna, M., and Martí, R.
  Models and solution methods for the uncapacitated r-allocation p-hub equitable center problem.
  Universitat de València Technical report, April 2016. Submitted.

- Peiró, J., Corberán, Á., and Martí, R.
  GRASP for the uncapacitated r-allocation p-hub median problem.
  Computers & Operations Research, 43: 50 – 60, 2014.

- Peiró, J., Corberán, Á., Martí, R., and Saldanha-da-Gama, F.
  Heuristic solutions for the stochastic uncapacitated r-allocation p-hub median problem with non-stop services.
  Universitat de València Technical report, May 2016. Submitted.

Bibliography

[1] Abdinnour-Helm, S. A hybrid heuristic for the uncapacitated hub location problem. *European Journal of Operatiomal Research 106* (1998), 489–499.

[2] Abdinnour-Helm, S., and Venkataramanan, M. Solution approaches to hub location problems. *Annals of Operations Research 78* (1998), 31–50.

[3] Aiex, R. M., Resende, M. G. C., and Ribeiro, C. C. TTT plots: A perl program to create time-to-target plots. *Optimization Letters 1*, 4 (2007), 355–366.

[4] Albareda-Sambola, M., Alonso-Ayuso, A., Escudero, L., Fernández, E., and Pizarro, C. Fix-and-relax-coordination for a multi-period location–allocation problem under uncertainty. *Computers & Operations Research 40* (2013), 2878–2892.

[5] Alonso-Ayuso, A., Escudero, L., and Ortuño, M. BFC, a branch-and-fix coordination algorithmic framework for solving stochastic 0–1 programs. *European Journal of Operational Research 151* (2003), 503–519.

[6] Alumur, S., and Kara, B. Y. Network hub location problems: The state of the art. *European Journal of Operational Research 190*, 1 (2008), 1–21.

[7] Alumur, S., Nickel, S., and Saldanha-da-Gama, F. Hub location under uncertainty. *Transportation Research B 46* (2012), 529–543.

[8] Alumur, S., and Serper, E. The design of capacitated intermodal hub networks with different vehicle types. *Transportation Research B 86* (2016), 51–65.

[9] Avella, P., Benati, S., Cánovas-Martinez, L., Dalby, K., Di Girolamo, D., Dimitrijevic, B., Giannikos, I., Guttman, N., Hultberg, T. H., Fliege, J., Muñoz-Márquez, M., Ndiaye, M. M., Nickel, S., Peeters, P., Pérez-Brito, D., Policastro, S., Saldanha-da-Gama, F., and Zidda, P. Some personal views on the current state and the future of locational analysis. *European Journal of Operational Research 104* (1998), 269–287.

[10] Aykin, T. Lagrangian relaxation based approaches to capacitated hub-and-spoke network design problem. *European Journal of Operational Research 79* (1994), 501–523.

[11] Aykin, T. The hub location and routing problem. *European Journal of Operational Research 83* (1995), 200–219.

[12] Aykin, T. Networking policies for hub-and-spoke systems with application to the air transportation system. *Transportation Science 29* (1995), 201–221.

[13] Beasley, J. E. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society 41*, 11 (1990), 1069–1072.

[14] Bianchi, L., Dorigo, M., Gambardella, L., and Gutjahr, W. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing 8* (2009), 239–287.

[15] Bollapragada, R., Camm, J., Rao, U., and Wu, J. A two-phase greedy algorithm to locate and allocate hubs for fixed-wireless broadband access. *Operations Research Letters 33* (2005), 134–142.

[16] Brandeau, M., and Chiu, S. S. An overview of representative problems in location research. *Management Science 35* (1989), 645–674.

[17] Calik, H., Alumur, S. A., Kara, B., and Karasan, O. A tabu-based heuristic for the hub covering problem over incomplete hub networks. *Computers & Operations Research 36*, 12 (2009), 3088–3096.

[18] Campbell, J. F. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research 72*, 2 (1994), 387–405.

[19] Campbell, J. F. Hub location and the p-hub median problem. *Operations Research 44* (1996), 923–935.

[20] Campbell, J. F., and O'Kelly, M. E. Twenty-five years of hub location research. *Transportation Science 46*, 2 (2012), 153–169.

[21] Chen, J. A hybrid heuristic for the uncapacitated single allocation hub location problem. *Omega 35* (2007), 211–220.

[22] Contreras, I. Hub location problems. In *Location Science*, G. Laporte, S. Nickel, and F. Saldanha-da-Gama, Eds. Springer, 2015, pp. 311–344.

[23] Contreras, I., Cordeau, J.-F., and Laporte, G. Stochastic uncapacitated hub location. *European Journal of Operational Research 212* (2011), 518–528.

[24] Contreras, I., Díaz, J., and Fernández, E. Lagrangean relaxation for the capacitated hub location problem with single assignment. *OR Spectrum 31* (2009), 483–505.

[25] Contreras, I., Díaz, J., and Fernández, E. Branch and price for large-scale capacitated hub location problems with single assignment. *INFORMS Journal in Computing 23* (2011), 41–55.

[26] Cunha, C. B., and Silva, M. A genetic algorithm for the problem of configuring a hub-and-spoke network for a LTL trucking company in Brazil. *European Journal of Operational Research 179* (2007), 747–758.

[27] Dillenberger, C., Escudero, L., Wollensak, A., and Zang, W. On practical resource allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research 75* (1994), 275–286.

[28] Eiselt, H., and Marianov, V. A conditional p-hub location problem with attraction functions. *Computers & Operations Research 36* (2009), 3128–3135.

[29] Eiselt, H., and Marianov, V. *Foundations of location analysis.* Springer, New York, 2011.

[30] Ernst, A., and Krishnamoorthy, M. Solution algorithms for the capacitated single allocation hub location problem. *Annals of Operations Research 86* (1999), 141–159.

[31] Ernst, A. T., and Krishnamoorthy, M. Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science 4*, 3 (1996), 139–154.

[32] Escudero, L., and Salmerón, J. Fix-and-relax partitioning. an algorithmic framework for large-scale resource constrained project selection and scheduling. *Annals of Operations Research 140* (2005), 163–188.

[33] Fanjul-Peyro, L., and Ruiz, R. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research 207*, 1 (2010), 55–69.

[34] Farahani, R. Z., and Hekmatfar, M. *Facilities location: Concepts, models, algorithms and case studies.* Springer-Verlag, 2009.

[35] Farahani, R. Z., Hekmatfar, M., Arabani, A. B., and Nikbakhsh, E. Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering 64*, 4 (2013), 1096–1109.

[36] Feo, T. A., and Resende, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters 8*, 2 (1989), 67–71.

[37] Feo, T. A., and Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6*, 2 (1995), 109–133.

[38] Festa, P., and Resende, M. G. C. *Essays and Surveys in Metaheuristics.* Springer US, Boston, MA, 2002, ch. GRASP: An Annotated Bibliography, pp. 325–367.

[39] Festa, P., and Resende, M. G. C. An annotated bibliography of grasp–part ii: Applications. *International Transactions in Operational Research 16*, 2 (2009), 131–172.

[40] Festa, P., and Resende, M. G. C. GRASP: Basic components and enhancements. *Telecommunication Systems 46*, 3 (2011), 253–271.

[41] Glover, F. Heuristics for integer programming using surrogate constraints. *Decision Sciences 8*, 1 (1977), 156–166.

[42] Glover, F. Tabu search and adaptive memory programing – advances, applications and challenges. In *Interfaces in Computer Science and Operations Research* (1996), Kluwer, pp. 1–75.

[43] Glover, F. A template for scatter search and path relinking. In *Artificial Evolution* (1998), J. K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Eds., Lecture Notes in Computer Science 1363, Springer, pp. 13–54.

[44] Glover, F., and Laguna, M. *Tabu search*. Kluwer, Norwell, MA, 1997.

[45] Glover, F., Laguna, M., and Martí, R. Fundamentals of scatter search and path relinking. *Control and Cybernetics 29*, 3 (2000), 652–684.

[46] Hakimi, S. L. Optimal location of switching centers and the absolute centers and medians of a graph. *Operations Research 12* (1964), 450–459.

[47] Hakimi, S. L. Optimal distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research 13* (1965), 462–475.

[48] Hale, T. S., and Moberg, C. R. Location science research: a review. *Annals of Operations Research 123* (2003), 21–35.

[49] Holland, J. H. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, 1975.

[50] Ilić, A., Urošević, D., Brimberg, J., and Mladenović, N. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research 206*, 2 (2010), 289–300.

[51] Jacobs, L. W., and Brusco, M. J. A local-search heuristic for large set-covering problems. *Naval Research Logistics 42*, 7 (1995), 1129–1140.

[52] Klincewicz, J. Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research 40* (1992), 283–302.

[53] Korte, B., and Vygen, J. *Combinatorial Optimization: Theory and Algorithms.* Algorithms and Combinatorics. Springer Science & Business Media, Berlin Heidelberg, 2012.

[54] Kratica, J., Milanović, M., Stanimirović, Z., and Tošić, D. An evolutionary-based approach for solving a capacitated hub location problem. *Applied Soft Computing 11* (2011), 1858–1866.

[55] Kratica, J., Stanimirović, Z., Tošić, D., and Filipović, V. Two genetic algorithms for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research 182*, 1 (2007), 15–28.

[56] Kratica, J., Stanimirović, Z., Tošić, D., and Filipović, V. Genetic algorithm for solving uncapacitated multiple allocation hub location problem. *Computing and Informatics 24* (2012), 415–426.

[57] Kress, D., and Pesch, E. Sequential competitive location on networks. *European Journal of Operational Research 217*, 3 (2012), 483–499.

[58] Laguna, M., and Martí, R. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing 11* (1999), 44–52.

[59] Laguna, M., and Martí, R. *Scatter Search: Methodology and Implementations in C.* Kluwer Academic Publishers, 2003.

[60] Laporte, G., Nickel, S., and Saldanha-da-Gama, F. Introduction to location science. In *Location Science*, G. Laporte, S. Nickel, and F. Saldanha-da-Gama, Eds. Springer, 2015, pp. 1–18.

[61] Laporte, G., Nickel, S., and Saldanha-da-Gama, F. *Location Science.* Springer International Publishing, 2015.

[62] Love, R., Morris, J., and Wesolowski, G. *Facilities location: Models and methods.* Elsevier Science Publishing Co., New York, 1988.

[63] Lozano, M., Molina, D., and García-Martínez, C. Iterated greedy for the maximum diversity problem. *European Journal of Operational Research 214*, 1 (2011), 31–38.

[64] Lüer-Villagra, A., and Marianov, V. A competitive hub location and pricing problem. *European Journal of Operational Research 231* (2013), 734–744.

[65] Marianov, V., and Serra, D. Location models for airline hubs behaving as M/D/c queues. *Computers & Operations Research 30* (2003), 983–1003.

[66] Marianov, V., Serra, D., and ReVelle, C. Location of hubs in a competitive environment. *European Journal of Operational Research 114* (1999), 363–371.

[67] Martí, R., Laguna, M., and Glover, F. Principles of scatter search. *European Journal of Operational Research 169*, 2 (2006), 359–372.

[68] Melo, M. T., Nickel, S., and Saldanha-da-Gama, F. Facility location and supply chain management - a review. *European Journal of Operational Research 196*, 2 (2009), 401–412.

[69] Meyer, T., Ernst, A. T., and Krishnamoorthy, M. A 2-phase algorithm for solving the single allocation p-hub center problem. *Computers & Operations Research 36*, 12 (2009), 3143–3151.

[70] Michalewicz, Z. *Genetic algorithms + Data structures = Evolution programs.* Artificial intelligence series. Springer-Verlag, Berlin, Germany, 1992.

[71] Milanović, M. *A new evolutionary based approach for solving the uncapacitated multiple allocation p-hub median problem*, vol. 75 of *Advances in Intelligent and Soft Computing.* Springer-Verlag, Berlin, 2010.

[72] Nickel, S., and Puerto, J. *Location Theory: A unified approach.* Springer, 2005.

[73] Nickel, S., Schobel, A., and Sonneborn, T. Hub location problems in urban traffic networks. In *Mathematics Methods and Optimization in Transportation Systems*, J. Niittymaki and M. Pursula, Eds. Kluwer Academic Publishers, 2001, pp. 1–12.

[74] O'Kelly, M. E. Activity levels at hub facilities in interacting networks. *Geographical Analysis 18*, 4 (1986), 343–356.

[75] O'Kelly, M. E. Location of interacting hub facilities. *Transportation Science 20*, 2 (1986), 92–106.

[76] O'Kelly, M. E. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research 32*, 3 (1987), 393–404.

[77] Owen, S. H., and Daskin, S. Strategic facility location: A review. *European Journal of Operational Research 111* (1998), 423–447.

[78] Pamuk, F., and Sepil, C. A solution to the hub center problem via a single-relocation algorithm with tabu search. *IIE Transactions 33* (2001), 399–411.

[79] Peiró, J., Corberán, A., and Martí, R. GRASP for the uncapacitated r-allocation p-hub median problem. *Computers & Operations Research 43*, 1 (2014), 50–60.

[80] Pirkul, H., and Schilling, D. An efficient procedure for designing single allocation hub and spoke systems. *Management Science 44* (1998), S235–S242.

[81] Randall, M. Solution approaches for the capacitated single allocation hub location problem using ant colony optimization. *Computational Optimization and Applications 39* (2008), 239–261.

[82] Resende, M. G. C., Gallego, M., Duarte, A., and Martí, R. GRASP and path relinking for the max-min diversity problem. *Computers & Operations Research 37* (2010), 498–508.

[83] Resende, M. G. C., Ribeiro, C. C., Glover, F., and Martí, R. Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of Metaheuristics*, M. Gendreau and J. Y. Potvin, Eds., International Series in Operations Research & Management Science 146. Springer, 2010, pp. 87–107.

[84] Resende, M. G. C., and Werneck, R. F. A hybrid heuristic for the p-median problem. *Journal of Heuristics 10*, 1 (2004), 59–88.

[85] ReVelle, C. S., and Eiselt, H. A. Location analysis: A synthesis and survey. *European Journal of Operational Research 165*, 1 (2005), 1–19.

[86] ReVelle, C. S., Eiselt, H. A., and Daskin, M. S. A bibliography for some fundamental problem categories in discrete location science. *European Journal of Operational Research 184*, 3 (2008), 817–848.

[87] ReVelle, C. S., and Laporte, G. The plant location problem: new models and research prospects. *Operations Research 44* (1996), 864–874.

[88] Ribeiro, C. C., and Resende, M. G. C. Path-relinking intensification methods for stochastic local search algorithms. *Journal of Heuristics 18*, 2 (2012), 193–214.

[89] Rodríguez-Martín, I., and Salazar-González, J. J. An iterated local search heuristic for a capacitated hub location problem. *Lecture Notes in Computer Science 4030* (2006), 70–81.

[90] Rosing, K., and ReVelle, C. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research 97* (1997), 75–86.

[91] Ruiz, R., and Stützle, T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research 187*, 3 (2008), 1143–1159.

[92] Ruiz, R., and Stützle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research 177*, 3 (2008), 2033–2049.

[93] Sasaki, M., Suzuki, A., and Drezner, Z. On the selection of hub airports for an airline hub-and-spoke system. *Computers & Operations Research 26* (1999), 1411–1422.

[94] Silva, M., and Cunha, C. New simple and efficient heuristics for the uncapacitated single allocation hub location problem. *Computers & Operations Research 36* (2009), 3152–3165.

[95] Sim, T., Lowe, T., and Thomas, B. The stochastic p-hub center problem with service-level constraints. *Computers & Operations Research 36*, 12 (2009), 3166–3177.

[96] Smith, H. K., Laporte, G., and Harper, P. R. Locational analysis: highlights of growth to maturity. *Journal of Operational Research Society 60* (1996), S140–S148.

[97] Smith, K., Krishnamoorthy, M., and Palaniswami, M. Neural versus traditional approaches to the location of interacting hub facilities. *Location Science 4* (1996), 155–171.

[98] Stanimirovic, Z. A genetic algorithm approach for the capacitated single allocation p-hub median problem. *Computing and Informatics 29* (2010), 117–132.

[99] Storn, R., and Price, K. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11*, 4 (1997), 341–359.

[100] Sung, C., and Jin, H. Dual-based approach for a hub network design problem under non-restrictive policy. *European Journal of Operational Research 132* (2001), 88–105.

[101] Wagner, B. An exact solution procedure for a cluster hub location problem. *European Journal of Operational Research 178* (2007), 391–401.

[102] Yaman, H. *Concentrator Location in Telecommunication Networks*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, December 2002.

[103] Yaman, H. Allocation strategies in hub networks. *European Journal of Operational Research 211*, 3 (2011), 442–451.

[104] Yaman, H., and Carello, G. Solving the hub location problem with modular link capacities. *Computers & Operations Research 32*, 12 (2005), 3227–3245.

[105] Ying, K. C., and Cheng, H. M. Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications 37*, 4 (2010), 2848–2852.

[106] Zitzler, E., and Thiele, L. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation 3*, 4 (1999), 257–271.