UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI TECNOLOGIE DELL'INFORMAZIONE

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN INFORMATICA

Algoritmi di programmazione matematica per lo scheduling di satelliti per l'osservazione terrestre

RELATORE:

Prof. **Giovanni Righini**

CORRELATORE:

Prof. **Roberto Cordone**

TESI DI LAUREA DI:

Federico Gandellini

Matricola n°: 618145

Anno Accademico 2004/2005

Ringraziamenti

Un ringraziamento particolare va al relatore Giovanni Righini ed al correlatore Roberto Cordone che mi hanno guidato nelle scelte e nel lavoro.

Ringrazio il Mino, la Savi e il Ludo, che mi sono stati vicini, hanno assecondato tutte le mie scelte e...senza i quali non sarei qui adesso!

Un grazie grande grande va alla Laurisia, che mi è sempre stata vicina, si è sorbita tutti i miei scleri e che mi ha sempre saputo sostenere quando impazzivo.

Ovviamente ringrazio gli amici del WebLab, tutti i ragazzi degli altri laboratori e tutte le persone (pazze e non) che ho imparato a conoscere in questi anni al Polo! Vorrei elencarle tutte, ma per farlo mi servirebbe...una seconda tesi!

I gnari e le gnare del paesello si meritano un grazie speciale, perchè hanno sempre dato tutto quello che potevano per aiutarmi anche nelle situazioni più incasinate e mi hanno regalato momenti di allegria e divertimento.

Grazie a tutti, amici!

Indice

Introduzione **3**

1 Descrizione del problema **5**

1.1 Le osservazioni da satellite 5

1.2 Le osservazioni nadir 11

1.3 Attività dei satelliti nadir 15

2 Il Modello **18**

2.1 Il Swath Segment Selection Problem 18

2.2 Una formulazione matematica 20

2.3 Complessità computazionale del problema 22

3 L'algoritmo lagrangiano **23**

3.1 Il rilassamento lagrangiano 25

3.2 Il branch and bound 29

3.3 L'upper bound 32

3.3.1 Il sottogradiente 34

3.4 Il lower bound 39

3.5 Strategia di branching 40

3.6 Strategia di visita 42

3.7 Le penalità lagrangiane 43

4 Prove sperimentali **45**

4.1 La generazione delle istanze 45

4.2 Ambiente e parametri di simulazione 49

4.3 Sperimentazione su varianti dell'algoritmo 50

4.4 Prove sperimentali sulle istanze piccole 51

4.5 Prove sperimentali sulle istanze grandi 55

Conclusioni e sviluppi futuri **57**

Bibliografia **60**

Introduzione

Lo scopo di questa tesi è la realizzazione di un algoritmo per lo scheduling di attività a bordo di un satellite in orbita attorno alla terra. Il satellite raccoglie immagini utilizzando una strumentazione rivolta verso il basso e con inquadratura fissa, da cui il termine *nadir*. Le osservazioni di questo tipo permettono di raccogliere soltanto immagini che si trovano direttamente sotto il satellite. Una volta suddivisa la porzione di superficie terrestre da fotografare in aree alle quali è associato un valore di guadagno che ne determina l'importanza, l'obbiettivo è fare in modo che il satellite raccolga le immagini delle aree che massimizzano il profitto totale, tenendo conto dei vincoli di capacità della memoria di bordo del satellite. Questo problema è chiamato *Swath Segment Selection Problem* (SSSP). Gli unici algoritmi esistenti per la risoluzione del problema richiedono ore di calcolo per ottenere soluzioni accettabili, cioè vicine alla soluzione ottima. Abbiamo quindi creato un software che riuscisse a generare soluzioni aventi una distanza dall'ottimo estremamente ridotta, se non nulla. Per meglio comprendere il problema e riuscire quindi a risolverlo ne abbiamo creato un modello matematico. Abbiamo determinato il grado di difficoltà del SSSP riconducendolo a un altro problema di tipo *NP*-difficile. È stato possibile ricondurre la risoluzione del SSSP a quella di tanti sottoproblemi di Knapsack (KP) interagenti fra loro. Per trovare la soluzione ottima del problema abbiamo rilassato in modo

lagrangiano i vincoli che legano fra loro i sottoproblemi di knapsack. Per ottenere i moltiplicatori lagrangiani, abbiamo poi implementato l'algoritmo del sottogradiente. Il rilassamento così ottenuto fornisce, in tempi molto brevi, una stima per eccesso dell'ottimo, inoltre fornisce informazioni utili per trovare una soluzione euristica di buona qualità, vale a dire una stima per difetto dell'ottimo. Di conseguenza l'errore compiuto è quantificabile. Tali elementi sono la base di partenza per un algoritmo di tipo *branch and bound* che trova la soluzione ottima. La fase di test dell'algoritmo è stata effettuata utilizzando istanze del problema da noi generate in modo pseudo-casuale. A questo scopo, abbiamo definito alcune classi di istanze che si differenziano per le dimensioni e la struttura del guadagno e dell'area.

Capitolo 1

Descrizione del problema

1.1 Le osservazioni da satellite

Un satellite è, per definizione, un oggetto che orbita attorno ad un altro corpo. I satelliti costruiti dall'uomo hanno questo nome appunto perchè orbitano attorno ad un corpo celeste.

Le missioni spaziali che hanno come obiettivo l'osservazione terrestre e che si sono svolte da quarant'anni ad oggi, hanno reso possibile uno studio del globo sempre più attento e particolareggiato. I satelliti presenti in orbita permettono di monitorare il nostro pianeta da molteplici punti di vista e con i più disparati scopi.

Esistono missioni che hanno come obiettivo l'osservazione di sistemi nuvolosi, o fenomeni naturali, e che trovano la loro diretta applicazione in ambito metereologico e civile, in quanto permettono di prevedere terremoti o imminenti catastrofi naturali.

Un'organizzazione che si occupa di queste operazioni è l'ESA (European Space Agency) [1], impegnata nella missione MetOp [2] che utilizza satelliti POES (Polar Operational Environmental Satellites) [3] [4] per il monitorag-

gio di agglomerati nuvolosi (Figura 1.1). Da tale missione vengono ricavate informazioni fondamentali al fine di eseguire previsioni metereologiche mattutine e pomeridiane, con particolare attenzione al livello di umidità e alla quantità di ozono presente nell'aria nonchè alla direzione dei venti. Le navette spaziali sono inoltre utilizzate per il soccorso civile, in quanto alcune delle apparecchiature installate sui satelliti in questione, che fanno uso di tecnologia infrarossa, non sono influenzate da fenomeni atmosferici che invece occultano il campo di azione delle apparecchiature che osservano nello spazio visibile.

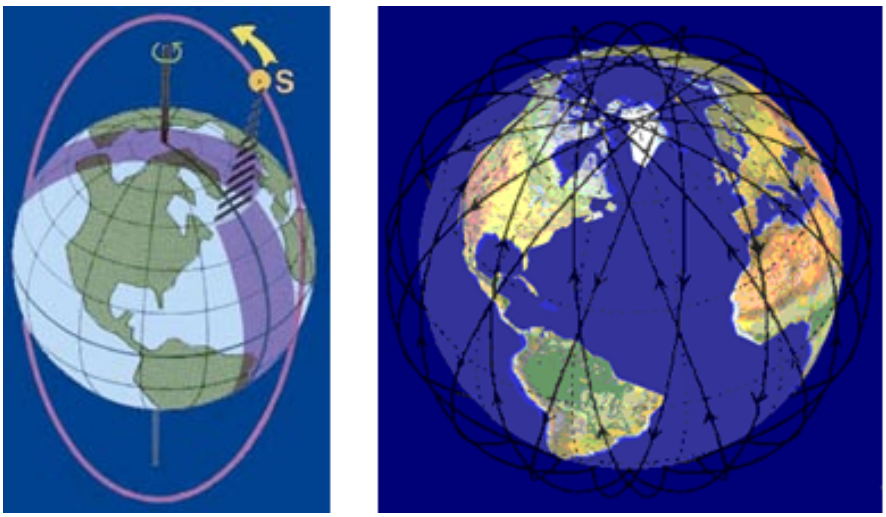

Figura 1.1: Esempi di orbite di satelliti POES.

In ambito scientifico le osservazioni dallo spazio sono utili allo studio e al controllo dei tassi di inquinamento di aria e acqua, nonché di fenomeni naturali di particolare interesse, come sismi di ridotta entità, maree ed eruzioni (Figura 1.2).

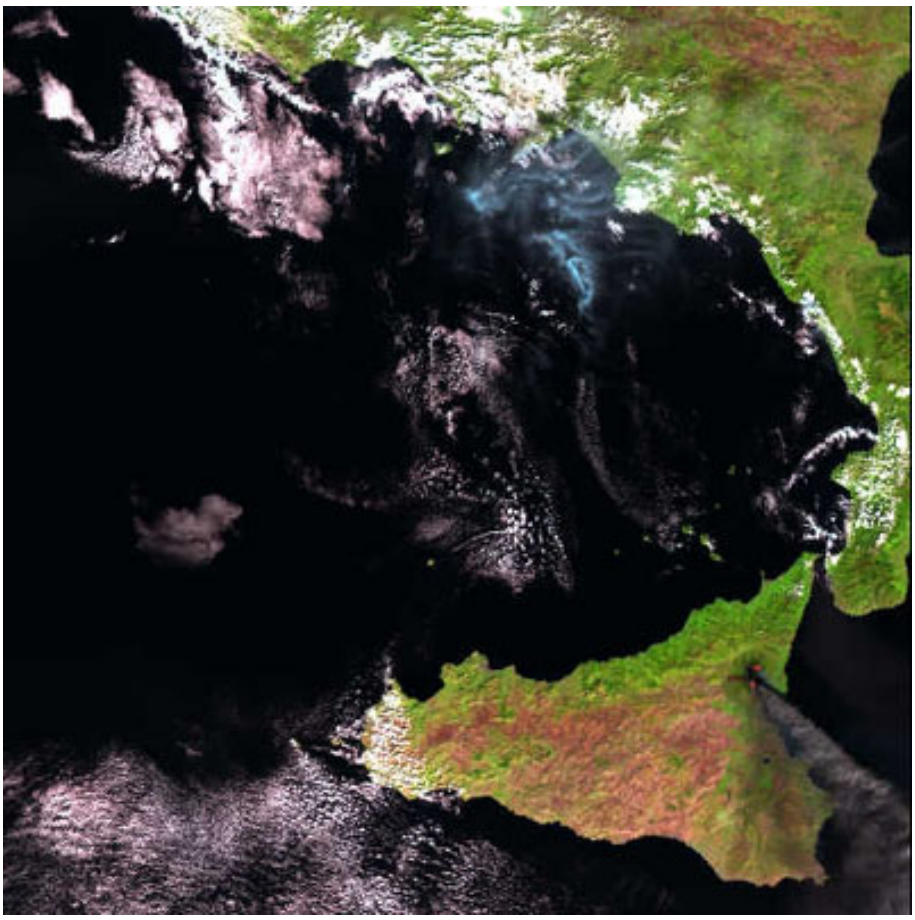

Figura 1.2: Immagine del vulcano Etna nell'eruzione del 28 ottobre 2002.

Queste operazioni trovano la loro diretta applicazione anche in campo cartografico. A tale proposito il progetto GLOBCOVER [5], anch'esso attivato dall'Agenzia Spaziale Europea, ha come obiettivo la produzione di cartografia terrestre ad elevato livello di dettaglio, al fine di comporre una mappa mondiale formata da un mosaico di immagini digitali le cui dimensioni ammontano a ben venti terabyte di memoria. L'immagine in figura 1.3 mostra una delle mappe mondiali realizzata dal progetto GLOBCOVER e riporta il numero di rilevazioni eseguite in ogni zona del globo terrestre nel periodo di tempo che intercorre tra il 1 dicembre 2004 e l'8 aprile 2005.

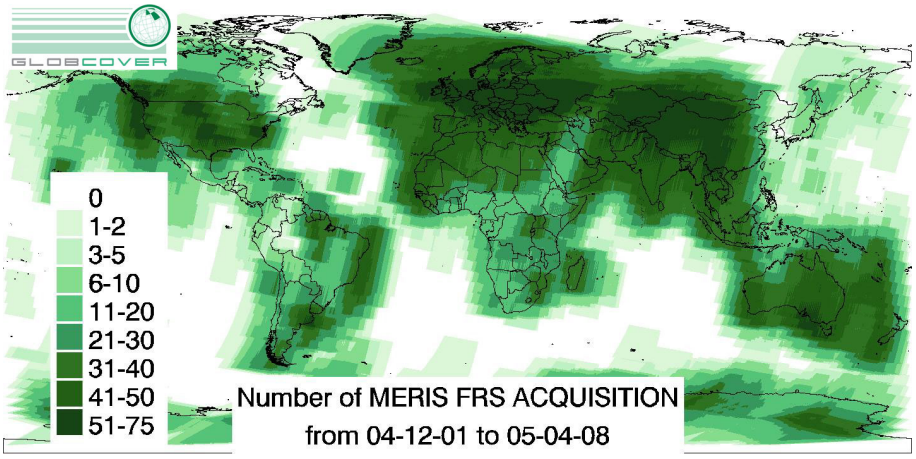

Figura 1.3: Mappa mondiale.

La navetta spaziale GOCE (Gravity Field and Ocean Circulation Explorer) [6], costruita e inviata nello spazio dall'ESA, è un esempio di applicazione di tale tecnologia, in quanto utilizza il monitoraggio dei movimenti del campo gravitazionale terrestre e lo studio del geoide per prevedere manifestazioni sismiche e vulcaniche che potrebbero verificarsi sul nostro pianeta. Un esempio del modello del geoide terrestre prodotto si può vedere in figura 1.4.

Anche l'Italia, attraverso l'ASI (Agenzia Spaziale Italiana), supervisiona la missione COSMO SkyMed [7], avviata allo scopo di monitorare l'ambiente ed il clima nella zona del Mediterraneo. L'obiettivo è quello di prevenire cataclismi e salvaguardare, così, non solo preziose risorse naturali quali coste e bacini idrici, ma anche la popolazione, permettendo di allertare in tempo la protezione civile in caso di pericolo. Come esempio, riportiamo la figura 1.5 relativa all'operazione MetOp, la missione europea corrispondente a COSMO SkyMed, che mostra la rilevazione di un uragano e l'orbita di un satellite metereologico.

Altri impieghi di satelliti per l'osservazione terrestre si hanno nel set-

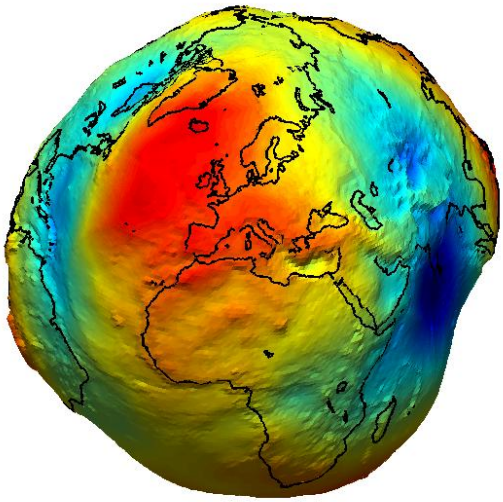

Figura 1.4: Geoide terrestre.

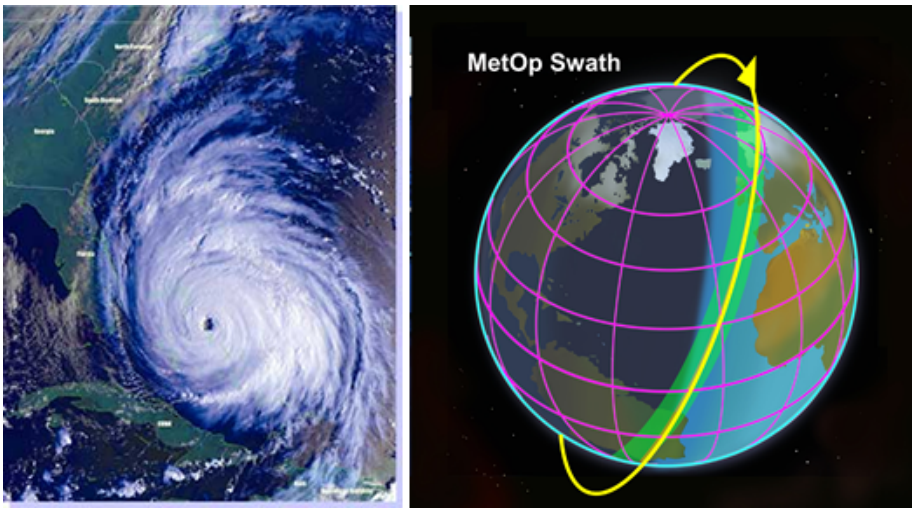

Figura 1.5: Una rilevazione e l'orbita di un satellite metereologico.

tore delle telecomunicazioni aeree, navali o terrestri, oppure in quello della localizzazione e controllo di mezzi mobili, come camion o automobili.

Importanti applicazioni si trovano anche nell'ambito dell'organizzazione del territorio, che può essere monitorato con sistemi satellitari e gestito capillarmente sia in termini di distribuzione di servizi (acquedotti o gasdotti), sia in termini di adeguamento a nuove norme legislative in ambito ecologico e organizzativo (piani regolatori).

Molte di queste navette spaziali utilizzate per scopi civili spesso appartengono all'esercito, che se ne serve per organizzare e pianificare missioni su fronti di guerra o per mantenere la sicurezza nazionale.

Requisiti essenziali che devono possedere i satelliti che si occupano di applicazioni come quelle appena descritte sono: un'alta risoluzione delle immagini acquisite, un' elevata frequenza di rivisitazione delle aree interessate al monitoraggio ed un'elevata capacità di trasmissione dei dati raccolti alle postazioni presenti sulla terraferma.

1.2 Le osservazioni nadir

La classe di osservazioni da satellite di nostro interesse è quella formata dalle osservazioni di tipo *nadir*. Con questo termine indichiamo il punto della sfera celeste sulla verticale dell'osservatore verso il basso [8]. Esso è infatti situato agli antipodi dello *zenit*.

I satelliti che eseguono questo tipo di osservazioni possiedono una strumentazione ottica di bordo immobile e rivolta verso il corpo celeste attorno al quale orbitano. Le immagini che tali tipi di apparecchiature possono catturare sono, ad ogni istante, solo quelle relative all'area sottostante l'obiettivo, cioè quelle per le quali il satellite si trova allo *zenith*.

Sfruttando la rotazione terrestre, è possibile combinare i vantaggi di transiti a bassa quota, cioè la possibilità di ottenere immagini più nitide e dettagliate, con una copertura globale del pianeta; questo obiettivo è raggiunto utilizzando i satelliti POES (Polar Operational Environmental Satellites). Queste navette spaziali eseguono osservazioni di tipo *nadir*. Idealmente l'orbita di un POES dovrebbe essere esattamente passante per i poli, ma nella realtà le orbite che questi satelliti possiedono sono inclinate e non puramente polari. L'immagine 1.6 mostra la differenza tra le orbite puramente polare (A) e non puramente polare (B) di un satellite POES.

Passiamo ora a definire alcuni termini che saranno utilizzati in seguito [9]. L'area di larghezza costante fotografata durante una porzione di orbita percorsa dalla navetta spaziale è detta *swath*, ed è suddivisa in *segment*, cioè in parti corrispondenti all'area che il satellite, in ogni istante, riesce ad immagazzinare in un'immagine. A causa della rotazione terrestre le tracce dei passaggi del satellite dal polo Nord al polo Sud, si intersecano con quelle dal polo Sud al polo Nord a formare una matrice (figure 1.7 e 1.8).

Immaginiamo di sviluppare la superficie terrestre in una mappa rettan-

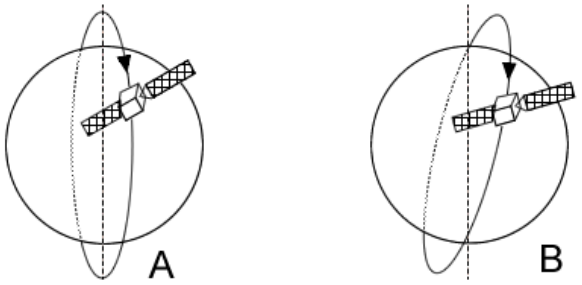

Figura 1.6: Orbita puramente polare (A) e non puramente polare (B) di un satellite POES.

golare, come fosse una cartina geografica dell'intero pianeta. Guardando tale mappa che riporta lo sviluppo piano delle tracce dei viaggi di un satellite POES (Figura 1.7) possiamo notare che ogni punto della terra è coperto da almeno due di queste tracce che identifichiamo come orizzontale (zona rossa dell'immagine) e verticale (zona blu dell'immagine). Nella figura le frecce indicano le direzioni del moto del satellite.

In figura 1.8 riportiamo un esempio della configurazione degli swath e dei loro segment. L'immagine mostra l'intersezione degli swath di più satelliti, nell'ingrandimento i differenti colori indicano tracce di satelliti diversi.

Le porzioni di Terra interessanti per la rilevazione, che non hanno una forma geometrica regolare, vengono definite *target*. La porzione utile di ciascun segment, cioè quella che si sovrappone ad un target, viene detta *shard*, dal termine inglese che si usa per indicare le schegge di vetro. In figura 1.9, con le due diverse tonalità di arancio, mostriamo gli shard determinati dalla sovrapposizione di swath orizzontali (linee rosse) e verticali (linee blu) con un target.

Nella realtà non solo gli swath associati a passaggi in direzioni opposte si intersecano, ma anche quelli associati a passaggi nella stessa direzione pos-

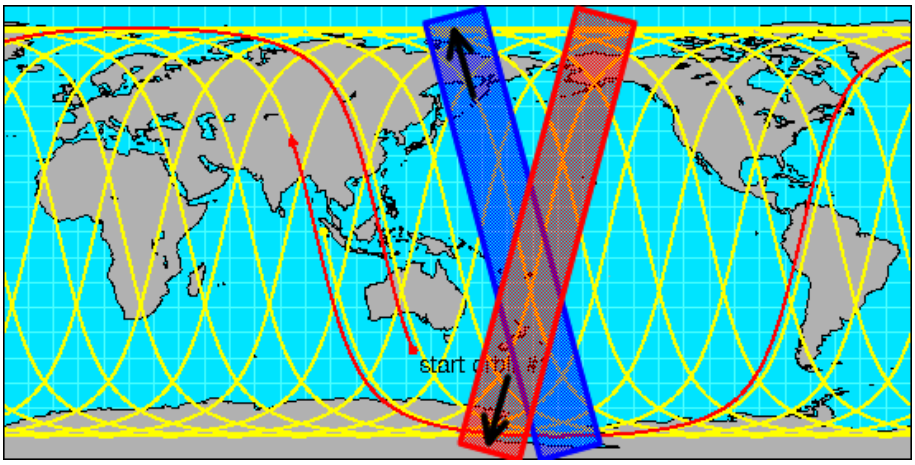

Figura 1.7: Sviluppo piano dell'orbita di un satellite POES.

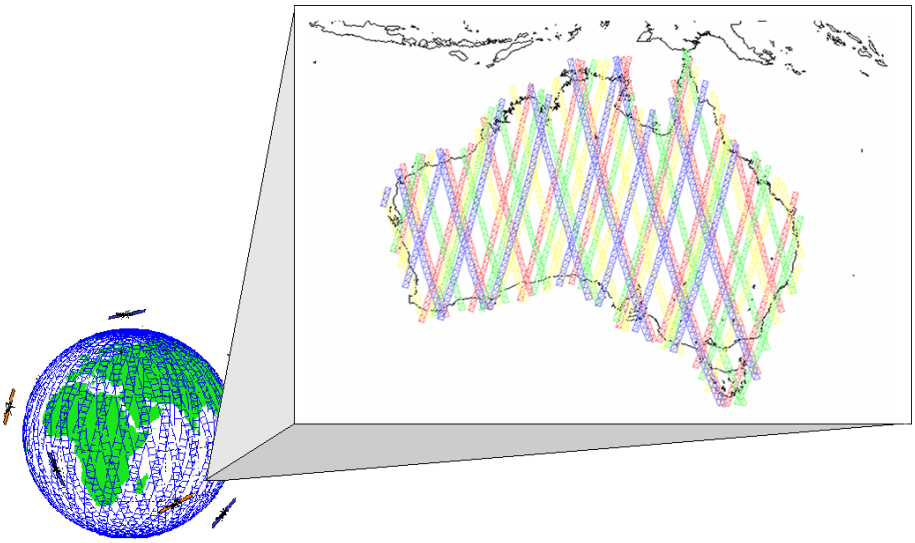

Figura 1.8: La griglia formata dall'intersezione degli swath.

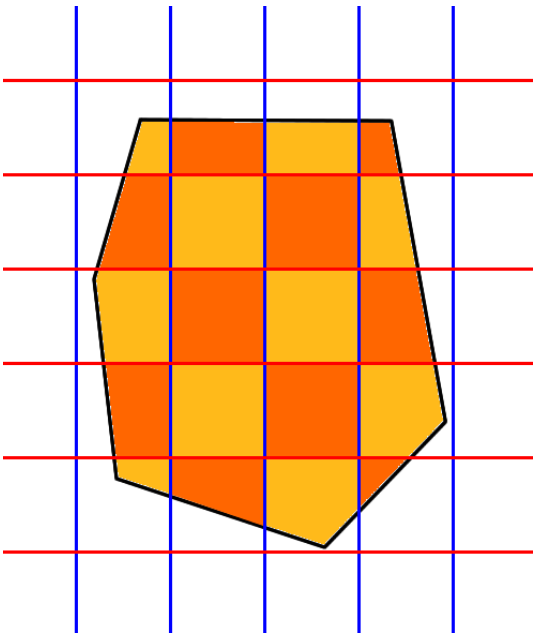

Figura 1.9: L'intersezione di un target con gli swath identifica gli shard.

sono sovrapporsi parzialmente. Questo comportamento è particolarmente frequente nelle vicinanze dei poli. Nel nostro modello trascuriamo queste sovrapposizioni e assumiamo che per ogni segment vi siano due soli passaggi durante i quali il satellite può riprendere. Per convenzione definiremo i due passaggi: orizzontale e verticale.

1.3 Attività dei satelliti nadir

Prendiamo in considerazione un satellite che orbita attorno ad un corpo celeste, ad esempio la Terra, e raccoglie immagini corrispondenti ad alcuni dei segment sopra ai quali transita. Al termine di ogni passaggio la navetta scarica a terra i dati raccolti in volo. Questa operazione spesso avviene in corrispondenza dei poli terrestri, in quanto i satelliti POES presentano un'alta frequenza di rivisitazione di tali luoghi. Le finestre temporali atte allo scambio dei dati con la terraferma sono ridotte e sono caratterizzate da una larghezza di banda limitata (*downlink*). La quantità di dati scaricabile in un certo intervallo di tempo e con un determinato downlink è detta *capacity*. Il satellite non può quindi immagazzinare nella memoria di bordo, in un solo passaggio, una quantità di dati maggiore della capacity, dato che non riuscirebbe a trasmetterli a terra alla fine di quel passaggio.

Possiamo definire i tragitti di un satellite *nadir* come un insieme di passaggi orizzontali e verticali. Ogni passaggio orizzontale identifica uno swath rettangolare, che interseca un altro swath osservato durante un passaggio verticale (e viceversa), dando origine ad una maglia di celle (i segment).

Il problema che affronteremo in seguito consiste nell'organizzare il lavoro giornaliero del satellite. Il nostro obiettivo è scegliere quali segment di questa griglia fotografare, tenendo conto sia delle capacità della memoria di bordo del satellite sia del fatto che non tutte le aree della mappa sono caratterizzate da eguale importanza. È importante notare che se decidiamo di fotografare un segmento durante un passaggio orizzontale, sarà inutile acquisire nuovamente quel segmento in un passaggio verticale. Tale operazione non porta vantaggio e, al tempo stesso, sottrae parte della memoria di bordo.

Come sostenuto da Ciriani *et al* [10], non tutte le immagini raccolte

possiedono lo stesso grado di interesse, anzi, esso varia a seconda dell'utilizzatore di tali rilevazioni. Ad esempio un ente che si occupa di eseguire previsioni metereologiche è tipicamente più interessato a immagini relative ad agglomerati urbani, che ad immagini relative all'oceano. Inoltre, se l'area di interesse è molto grande, come accade per intere nazioni o continenti, non può essere contenuta in una sola immagine, ma è composta da molte immagini acquisite in momenti differenti.

Ogni segment è caratterizzato da due valori che rappresentano l'area ripresa, cioè la memoria occupata a seconda della direzione del passaggio.

Infatti, mentre la larghezza dello swath ripreso è fissa, la sua lunghezza può variare quando lo shard di interesse non copre interamente il segment considerato, è possibile attivare l'apparecchio solo durante la frazione di tempo utile ad immagazzinare lo shard. Ad esempio, supponiamo che un segment contenga una porzione di territorio da fotografare con una forma allungata in orizzontale, come in figura 1.10 nella quale vogliamo acquisire lo shard identificato dalla penisola: in rosso sono indicati i limiti dello swath orizzontaleche la copre ed in azzurro quelli dello swath verticale. Sono possibili due casi a seconda della direzione di percorrenza del satellite (indicata dalle frecce bianche). La parte colorata in arancio indica l'immagine non utile immagazzinata in ciascuno dei due casi.

Caso A: Tale immagine può essere catturata durante un viaggio orizzontale del satellite, attivando lo strumento presente a bordo per tutta la lunghezza del segment. Questo causa la cattura di tutto il segment e oltre a raccogliere immagini di scarso interesse, spreca una grande quantità di memoria di bordo per immagazzinare dati inutili.

Caso B: Scegliendo di raccogliere la suddetta striscia di territorio durante una passata verticale, invece, lo strumento di acquisizione di immagini

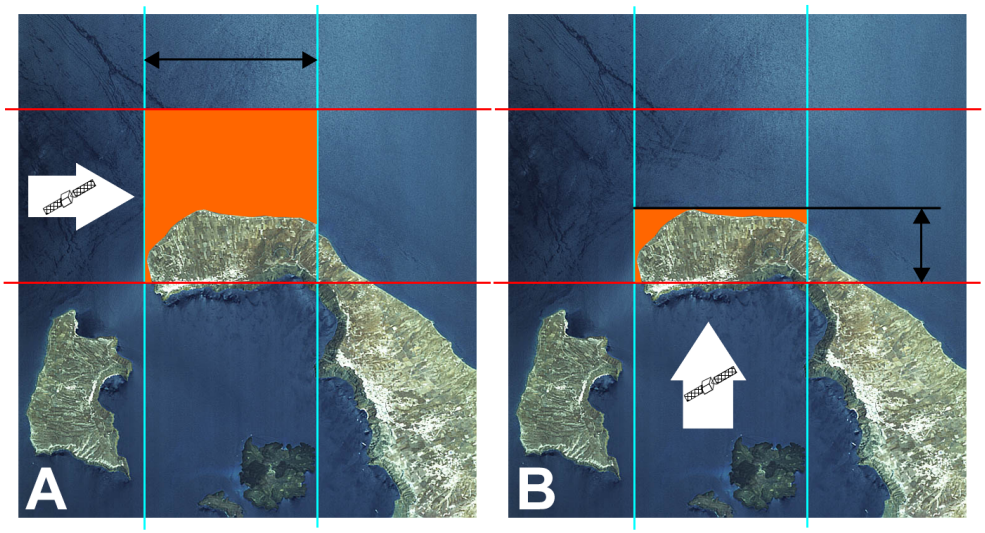

Figura 1.10: La quantità di immagine non utile cambia al variare della direzione di acquisizione.

avrebbe potuto rimanere attivo solo in corrispondenza dell'area utile, minimizzando lo spreco di memoria di bordo.

I due diversi valori di area associati allo stesso segment a seconda della direzione di viaggio del satellite, quindi, fanno in modo che venga favorita la scelta di catturare la porzione di territorio interessante nella direzione più economica, perchè ciò non influenza le altre scelte.

Capitolo 2

Il Modello

2.1 Il Swath Segment Selection Problem

Il *Swath Segment Selection Problem* (SSSP nel seguito) consiste nella scelta ottimale dei segment di swath, in modo che gli shard fotografati dal satellite rispettino le capacità di downlink e la memoria di bordo. Le uniche soluzioni algoritmiche esistenti per il SSSP sono l'algoritmo ASTER di Muraoka [11] e l'algoritmo branch-and-bound di Knight e Smith [9]. Il primo è un algoritmo deterministico di tipo greedy, è molto veloce nel trovare una soluzione, ma non ne garantisce l'ottimalità. Il secondo è un algoritmo branch-and-bound con strategia di ricerca depth-first, nel quale l'upper bound è ottenuto risolvendo un rilassamento del SSSP costituito da un problema di flusso massimo su uno speciale grafo a livelli. Questo secondo algoritmo, però, presenta almeno due aspetti che ne limitano il campo di applicazione alle situazioni reali:

1. Il guadagno associato ad ogni shard è direttamente proporzionale alla dimensione dello shard stesso, e quindi alla occupazione di memoria che esso comporta. Tale assunzione non è sempre vera, in quanto, nei

problemi reali, la correlazione tra dimensione e importanza dello shard può essere molto labile: un'applicazione di monitoraggio del traffico stradale, ad esempio, avrebbe un basso guadagno associato ad una grande porzione di terreno scarsamente abitato, mentre una piccola porzione di città avrebbe una grande importanza.

2. Il rilassamento realizzato è equivalente ad un rilassamento lineare, e non rispetta i vincoli strutturali di costruzione del satellite e funzionamento del dispositivo che raccoglie le immagini. Tale rilassamento, infatti, fraziona i segment e assume che lo strumento ne possa acquisire solo una parte, il che non è sempre vero.

2.2 Una formulazione matematica

In questa sezione presentiamo il modello matematico del SSSP da noi realizzato e utilizzato per progettare un algoritmo risolutore.

È dato il numero di swath orizzontali (m) e verticali (n), che vengono modellati come righe e colonne di una matrice. Ogni cella (i,j) di tale matrice identifica un segment e il corrispondente shard. Ad ogni shard sono associati tre valori: l'area dello shard considerato durante un passaggio orizzontale ($a_{ij}^{(h)}$) o verticale ($a_{ij}^{(v)}$) del satellite descrive l'occupazione di memoria associata alla ripresa dello shard, mentre il *reward* (r_{ij}) rappresenta l'importanza di quello shard, cioè il guadagno ad esso associato. Per ogni swath orizzontale e verticale è definito un valore di downlink (rispettivamente $d_i^{(h)}$ e $d_j^{(v)}$) che rappresenta la quantità massima di dati, cioè l'area massima, che è possibile scaricare a terra alla fine di quel passaggio. I valori dei downlink possono essere diversi fra loro per descrivere il fatto che i passaggi non sono rigorosamente identici. Infatti essendo l'orbita non puramente polare, le possibilità di trasmissione dei dati a terra cambiano da passaggio a passaggio.

La quantità di memoria di bordo t del satellite influenza il problema allo stesso modo della capacità di downlink. Di conseguenza, ne terremo conto considerando come downlink di riga o colonna il valore minore fra l'effettivo downlink di riga o colonna e il valore della memoria di bordo:

$$d_i^{(h)} \leftarrow \min\{d_i^{(h)}, t\}$$
$$d_j^{(v)} \leftarrow \min\{d_j^{(v)}, t\}$$

La soluzione del SSSP può essere descritta specificando quali shard ven-

gono ripresi durante i passaggi orizzontali e quali durante quelli verticali. Di conseguenza la variabile $x_{ij}^{(h)}$ vale 1 quando lo shard (i,j) è fotografato durante un passaggio orizzontale, mentre $x_{ij}^{(v)}$ vale 1 quando esso è catturato durante un passaggio verticale.

Abbiamo formulato il SSSP come segue:

$$SSSP: \quad \max \; z = \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}(x_{ij}^{(h)} + x_{ij}^{(v)}) \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij}^{(h)} x_{ij}^{(h)} \leq d_i^{(h)} \qquad i = 1..m \tag{2.2}$$

$$\sum_{i=1}^{m} a_{ij}^{(v)} x_{ij}^{(v)} \leq d_j^{(v)} \qquad j = 1..n \tag{2.3}$$

$$x_{ij}^{(h)} + x_{ij}^{(v)} \leq 1 \qquad i = 1..m, \;\; j = 1..n \tag{2.4}$$

$$x_{ij}^{(h)}, \;\; x_{ij}^{(v)} \in \{0,1\} \qquad i = 1..m, \;\; j = 1..n \tag{2.5}$$

L'obiettivo (2.1) è massimizzare i guadagni totali, intesi come sommatoria dei guadagni (r_{ij}) associati ai soli shard fotografati.

Il problema presenta un vincolo di capacità associato ad ogni swath orizzontale (2.2), ed uno associato ad ogni swath verticale (2.3) che proibiscono di accumulare più dati di quelli che sono trasmissibili al termine del passaggio, più un vincolo di selezione associato ad ogni shard, che indica il fatto che esso può essere fotografato solo durante uno dei due passaggi, quello orizzontale o quello verticale (2.4).

È possibile osservare che i vincoli (2.4) sono gli unici che impediscono una risoluzione del SSSP con metodologie standard, perchè coinvolgono sia le variabili $x_{ij}^{(h)}$ che si riferiscono ai viaggi orizzontali, sia le $x_{ij}^{(v)}$ che si riferiscono a quelli verticali. Se tali vincoli non esistessero, il problema si

ridurrebbe all'unione di una serie di problemi di knapsack definiti sulle variabili che descrivono solo i passaggi orizzontali e verticali. Proprio questa particolare caratteristica del problema ci ha suggerito l'idea di applicare a questo vincolo un rilassamento di tipo lagrangiano.

2.3 Complessità computazionale del problema

Si può dimostrare che il SSSP è un problema *NP*-difficile. La dimostrazione si basa sulla osservazione che, considerando come problema di SSSP il caso particolare di una matrice avente un solo swath orizzontale, cioè avente $m = 1$, e n swath verticali, si ottiene esattamente un problema di knapsack, che è un problema di tipo *NP*-difficile.

Capitolo 3

L'algoritmo lagrangiano

Nella prima sezione (3.1) di questo capitolo descriveremo come eseguire il rilassamento lagrangiano dei vincoli (2.4). Una volta ottenuta la formulazione del problema rilassato, mostreremo come scomporla in tanti problemi di knapsack indipendenti. Passeremo quindi alla trattazione delle parti principali dell'algoritmo da noi sviluppato per la risoluzione del SSSP affrontandolo con una metodologia di tipo *top-down*. Inizieremo presentando l'algoritmo *branch and bound* (sezione 3.2) e identificandone le due componenti principali, cioè le procedure per il calcolo dell'*upper bound* (sezione 3.3) e del *lower bound* (sezione 3.4) che vengono aggiornati in ogni nodo dell'albero dei sottoproblemi. Mostreremo quindi il rilassamento utilizzando l'algoritmo del *sottogradiente* [12] (sezione 3.3.1), quindi spiegheremo come tale algoritmo ne stimi i moltiplicatori lagrangiani. In seguito tratteremo il calcolo della soluzione euristica ammissibile, basata su una soluzione lagrangiana corretta. L'algoritmo del sottogradiente utilizza al suo interno l'algoritmo di Pisinger [13] modificato da Ceselli [14] per la risoluzione dei sottoproblemi di knapsack derivanti dalla scomposizione del problema rilassato. Nell'ultima sezione del capitolo tratteremo l'algoritmo che esegue

le penalità lagrangiane, cioè una procedura che permette, in alcuni casi, di determinare il valore ottimo di una parte delle variabili.

3.1 Il rilassamento lagrangiano

Il rilassamento lagrangiano [12] [15] è una tecnica molto efficace utilizzata per rilassare problemi aventi una particolare struttura dei vincoli. La caratteristica dei problemi in questione è quella di presentare una serie di vincoli che impediscono di risolvere il problema con metodologie già note. Il rilassamento di tali vincoli, quindi, genera un nuovo problema che è possibile risolvere con algoritmi esistenti. I vincoli che vengono rilassati si definiscono talvolta *vincoli complicanti*.

L'SSSP è proprio un problema di questo genere, infatti se mancassero tutti i vincoli (2.4), i vincoli (2.2) e (2.3) potrebbero essere considerati separatamente in due famiglie di sottoproblemi di knapsack che coinvolgono le sole variabili $x_{ij}^{(h)}$ per ciascuna i, o le sole $x_{ij}^{(v)}$ per ciascuna j. Questi sottoproblemi possono essere risolti utilizzando l'algoritmo di Pisinger per la risoluzione dei problemi di knapsack.

Il rilassamento lagrangiano consiste nel rimuovere i vincoli complicanti e sottrarre nella funzione obiettivo il valore della loro violazione, opportunamente pesato con un coefficiente detto *moltiplicatore lagrangiano*, in modo da penalizzare le soluzioni inammissibili. Nel caso specifico, la violazione di ciascuno degli mn vincoli (2.4) vale $(x_{ij}^{(h)} + x_{ij}^{(v)} - 1)$. Essa viene moltiplicata per un coefficiente $\lambda_{ij} \geq 0$ e, dato che il problema è di massimizzazione, sottratta alla funzione obiettivo. Si ottiene così il nuovo problema:

$$LR: \quad \max \; z = \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}(x_{ij}^{(h)} + x_{ij}^{(v)}) - \sum_{i=1}^{m}\sum_{j=1}^{n} \lambda_{ij}(x_{ij}^{(h)} + x_{ij}^{(v)} - 1) \quad (3.1)$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij}^{(h)} x_{ij}^{(h)} \leq d_i^{(h)} \qquad i = 1..m \quad (3.2)$$

$$\sum_{i=1}^{m} a_{ij}^{(v)} x_{ij}^{(v)} \leq d_j^{(v)} \qquad j = 1..n \quad (3.3)$$

$$x_{ij}^{(h)}, \; x_{ij}^{(v)} \in \{0,1\} \qquad i = 1..m, \; j = 1..n \quad (3.4)$$

Eseguendo una riorganizzazione dei termini della funzione obiettivo (3.1), otteniamo la formulazione finale del nuovo problema:

$$LR: \quad \max \; z = \sum_{i=1}^{m}\sum_{j=1}^{n} (r_{ij} - \lambda_{ij}) x_{ij}^{(h)} + \sum_{i=1}^{m}\sum_{j=1}^{n} (r_{ij} - \lambda_{ij}) x_{ij}^{(v)} + \sum_{i=1}^{m}\sum_{j=1}^{n} \lambda_{ij} \quad (3.5)$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij}^{(h)} x_{ij}^{(h)} \leq d_i^{(h)} \qquad i = 1..m \quad (3.6)$$

$$\sum_{i=1}^{m} a_{ij}^{(v)} x_{ij}^{(v)} \leq d_j^{(v)} \qquad j = 1..n \quad (3.7)$$

$$x_{ij}^{(h)}, \; x_{ij}^{(v)} \in \{0,1\} \qquad i = 1..m, \; j = 1..n \quad (3.8)$$

La formulazione LR è definita *rilassamento lagrangiano* del problema originario. La funzione obiettivo (3.5) utilizza come reward i valori:

$$r_{ij}^{(\lambda)} = (r_{ij} - \lambda_{ij}) \qquad i = 1..m, \; j = 1..n \quad (3.9)$$

Tali valori sono detti *rewards lagrangiani*.

Il nuovo problema è in realtà costituito dalla giustapposizione dei seguenti $m + n$ problemi indipendenti:

$$KP_i^{(h)}: \quad \max \ \xi_i(\lambda) = \sum_{j=1}^{n} r_{ij}^{(\lambda)} x_{ij}^{(h)} \tag{3.10}$$

$$\text{s.t.} \ \sum_{j=1}^{n} a_{ij}^{(h)} x_{ij}^{(h)} \leq d_i^{(h)} \tag{3.11}$$

$$x_{ij}^{(h)} \in \{0,1\} \qquad j = 1..n \tag{3.12}$$

con $i = 1..m$.

$$KP_j^{(v)}: \quad \max \ \varphi_i(\lambda) = \sum_{i=1}^{m} r_{ij}^{(\lambda)} x_{ij}^{(v)} \tag{3.13}$$

$$\text{s.t.} \ \sum_{i=1}^{m} a_{ij}^{(v)} x_{ij}^{(v)} \leq d_j^{(v)} \tag{3.14}$$

$$x_{ij}^{(v)} \in \{0,1\} \qquad i = 1..m \tag{3.15}$$

con $j = 1..n$.

Come possiamo notare, i vincoli di downlink (2.2) e (2.3) sono rimasti invariati, e per risolvere il problema è necessario dividere la formulazione negli $m + n$ sottoproblemi di knapsack. Il modello finale del rilassamento lagrangiano del SSSP è mostrato qui di seguito (3.16). È da notare il fatto che nella funzione obiettivo è presente anche un termine costante che rappresenta la somma di tutti i λ_{ij}.

$$
\begin{aligned}
LR: \quad \max \; z \;=\; & \sum_{i=1}^{m} \left\{ \begin{array}{lll} \max & \xi_i^{(\lambda)} = \sum_{j=1}^{n} r_{ij}^{(\lambda)} x_{ij}^{(h)} & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij}^{(h)} x_{ij}^{(h)} \leq d_i^{(h)} & + \\ & x_{ij}^{(h)} \in \{0,1\} \qquad j = 1..n & \end{array} \right. \\
& + \sum_{j=1}^{n} \left\{ \begin{array}{lll} \max & \varphi_i^{(\lambda)} = \sum_{i=1}^{m} r_{ij}^{(\lambda)} x_{ij}^{(v)} & \\ \text{s.t.} & \sum_{i=1}^{m} a_{ij}^{(v)} x_{ij}^{(v)} \leq d_j^{(v)} & + \\ & x_{ij}^{(v)} \in \{0,1\} \qquad i = 1..m & \end{array} \right. \\
& + \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij}
\end{aligned}
\tag{3.16}
$$

3.2 Il branch and bound

L'algoritmo di branch and bound da noi sviluppato implementa un albero di ricerca, i cui nodi rappresentano i sottoproblemi nei quali è scomposto il problema iniziale. Lo pseudo codice in figura 3.1 descrive tale algoritmo.

L'algoritmo riceve il numero di righe (m), il numero di colonne (n), reward r e le aree $a^{(h)}$ e $a^{(v)}$. Nella fase iniziale, l'algoritmo esegue la procedura del sottogradiente nel nodo radice dell'albero, ed ottiene (come descritto nella sezione 3.3.1):

1. La soluzione x_{UB} del rilassamento del problema e il suo valore Z_{UB}, che viene salvato come valore di upper bound corrente sull'ottimo.

2. La miglior soluzione euristica x^*_{LB} trovata ed il suo valore Z^*_{UB}, che rappresenta la stima per difetto dell'ottimo.

3. La matrice λ dei moltiplicatori lagrangiani che corrisponde al rilassamento.

4. L'indicazione dello shard (i,j) su cui conviene effettuare il branching, se necessario.

A questo punto, se il gap tra upper e lower bound è chiuso, l'algoritmo termina, altrimenti il nodo radice generato è inserito nella lista dei nodi aperti rispettando la strategia di visita prescelta (vedere sezione 3.6). Inizia quindi un ciclo che estrae il primo nodo dalla lista e controlla se esso contiene ancora soluzioni promettenti. Nel caso l'esito del test sia positivo, scompone il problema in sottoproblemi, fissando il valore delle due variabili dello shard (i,j) secondo una delle strategie descritte nella sezione 3.5, quindi riesegue il sottogradiente in ognuno di essi, valutando se contengono soluzioni promettenti e aggiornando le relative stime dei bound. In caso positivo li

```
function BranchAndBound(m, n, r, a^(h), a^(v))

begin

    Z*_LB ← 0

    x*_LB ← 0

    { Compute root node solution }

    (Z_UB, x_UB, Z*_LB, x*_LB, λ, i_B, j_B) ← Subgradient(m, n, r, a^(h), a^(v), λ_0)

    if Z_UB > Z_LB then

        PushQ(Z_UB, x_UB, λ, i_B, j_B)

    while not EmptyQ() do

        (Z_UB, x_UB, i_B, j_B, λ) ← ExtractQ()

        if Z_UB > Z_LB then

            for k = 1 to SonNumber do

                { Branch on i_B, k_B to generate the k-th subproblem }

                GenerateSubProblem(k, i_B, j_B)

                { Compute node solution }

                (Z'_UB, x'_UB, Z*_LB, x*_LB, λ', i'_B, j'_B) ←
                  Subgradient(m, n, r, a^(h), a^(v), λ_0)

                if Z_UB > Z_LB then PushQ(Z'_UB, x'_UB, i'_B, j'_B, λ')

                end if

            end for

        end if

    end while

    return (Z*_LB, x*_LB)

end
```

Figura 3.1: Pseudo codice dell'algoritmo branch and bound utilizzato.

inserisce nella lista, sempre rispettando la strategia di inserimento scelta. Il ciclo termina quando la lista si svuota. A questo punto il valore della miglior soluzione intera nota rappresenta la soluzione del nostro problema.

3.3 L'upper bound

Data una matrice di moltiplicatori lagrangiani $\lambda \geq 0$, il problema LR consiste in m problemi di knapsack indipendenti nelle variabili $x_{ij}^{(h)}$ e n problemi di knapsack indipendenti nelle variabili $x_{ij}^{(v)}$.

È da notare che le soluzioni di questi problemi sono calcolate indipendentemente con l'algoritmo di Pisinger, quindi può accadere che lo stesso shard venga fotografato sia durante il passaggio orizzontale sia durante quello verticale. In questo caso entrambe le matrici delle soluzioni presenteranno un 1 in corrispondenza dello shard (i,j) che è stato scelto, e quindi accadrà che $x_{ij}^{(h)} = x_{ij}^{(v)} = 1$. Questa soluzione viola il vincolo (2.4).

Il valore della soluzione del rilassamento è calcolato con la seguente formula:

$$Z_{UB} = \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}^{(\lambda)} x_{ij}^{(h)} + \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}^{(\lambda)} x_{ij}^{(v)} + \sum_{i=1}^{m}\sum_{j=1}^{n} \lambda_{ij} \tag{3.17}$$

Si dimostra che, per qualsiasi valore $\lambda \geq 0$, il valore della (3.17) è non inferiore all'ottimo del problema originale.

Il calcolo dell'upper bound, quindi è eseguito utilizzando guadagni modificati dai moltiplicatori lagrangeani. Avremo quindi i seguenti casi:

1. Se lo shard(i,j) è fotografato in una sola delle due direzioni, cioè accade che $x_{ij}^{(h)} = 1$ e $x_{ij}^{(v)} = 0$ oppure $x_{ij}^{(h)} = 0$ e $x_{ij}^{(v)} = 1$, il valore del moltiplicatore lagrangiano λ_{ij} non influisce sull'obiettivo perchè viene sottratto dal reward r_{ij}, come si vede dalla (3.9), ma viene nuovamente sommato dall'ultimo termine della (3.17).

2. Se lo shard (i,j) è fotografato in entrambe le direzioni, allora risulta $x_{ij}^{(h)} = x_{ij}^{(v)} = 1$. In questo caso il valore dell'upper bound è incremen-

tato di $(2r_{ij} - \lambda_{ij})$. In assenza di rilassamento tale soluzione avrebbe violato uno dei vincoli (2.4) nel problema iniziale. D'altra parte se i vincoli (2.4) non esistessero, una soluzione di questo tipo avrebbe dovuto incrementare il valore dell'upper bound di $2r_{ij}$. Il rilassamento lagrangiano impedisce questo comportamento penalizzando tale soluzione non ammissibile con il termine $-\lambda_{ij}$.

3. Se lo shard (i,j) non è fotografato in alcuna direzione, allora risulta $x_{ij}^{(h)} = x_{ij}^{(v)} = 0$ e il valore dell'upper bound è incrementato di λ_{ij}.

È quindi ovvio che si tratta di un rilassamento perchè il problema LR ammette più soluzioni di quello originale e, nelle soluzioni che sono ammissibili per entrambi i problemi, il valore della funzione obiettivo di LR è identico (caso 1) o più alto (caso 3). D'altra parte lo stesso discorso vale per il rilassamento ottenuto ignorando i vincoli (2.4) ma, dopo tale operazione, il caso 2 fornisce soluzioni più alte che non con il rilassamento lagrangiano.

3.3.1 Il sottogradiente

A questo punto rimane il problema di calcolare un valore per ogni moltiplicatore lagrangiano λ_{ij}. Lo scopo dell'algoritmo del sottogradiente è proprio tarare questi moltiplicatori in modo da guidare l'algoritmo di Pisinger nella risoluzione dei sottoproblemi di knapsack.

L'algoritmo del sottogradiente procede aumentando il valore dei moltiplicatori lagrangeani λ_{ij} per i vincoli che tendono ad essere violati, e diminuendolo per gli altri. Infatti se lo shard è fotografato in entrambe le direzioni, come riportato nel caso 2 della sezione 3.3, è bene che il relativo λ_{ij} assuma un valore maggiore rispetto a quello che potrebbe assumere, se lo shard non venisse acquisito in alcuna direzione, come descritto nel caso 3.

L'algoritmo del sottogradiente da noi realizzato è una variante di quello presentato da Beasley in [12]. La struttura dell'algoritmo è mostrata nello pseudo codice in figura 3.2.

Il nostro algoritmo esegue un ciclo, le cui condizioni di uscita sono rappresentate da un numero massimo di cicli e un gap fissato tra valore del rilassamento e valore della soluzione euristica. In questo ciclo esso calcola la soluzione del rilassamento lagrangiano del problema come già descritto e, nel caso tale soluzione sia migliorata rispetto a quella trovata precedentemente, esegue il calcolo della soluzione euristica come descritto nella sezione 3.4.

I moltiplicatori lagrangiani (λ_{ij}) sono inizializzati ad un valore λ_0, che è nullo per il nodo radice, mentre in ogni altro caso è il valore restituito dalla procedura subgradient per il nodo padre. L'aggiornamento dei moltiplicatori lagrangiani (λ_{ij}) è eseguito determinando innanzitutto la violazione s_{ij} relativa allo shard (i,j). Si calcola cioè la penalizzazione per i vincoli (2.4) che erano stati rilassati e vengono violati dalla soluzione del rilassamento.

```
procedure Subgradient(m, n, r, a^(h), a^(v), λ_0)

begin

    λ* ← λ

    t ← t_0

    while (t_p ≥ t_0) and (Z_UB − Z*_LB > δ) do

        { Compute the lagrangian rewards }

        r̃ ← r − λ

        { Compute the solution of the lagrangian relaxation }

        (Z_UB, x_UB) ←SolveLagrangeanProblem(m, n, r̃, a^(h), a^(v), λ_0)

        if Z_UB < Z*_UB then Z*_UB ← Z_UB, x*_UB ← x_UB

        { Compute the heuristic solution }

        (Z_LB, x_LB) ←Heuristic(m, n, r, a^(h), a^(v), λ_0)

        { Update best lower bound and multipliers }

        if Z_LB > Z*_LB then Z*_LB ← Z_LB, x*_LB ← x_LB, λ* ← λ

        { Compute violations }

        s ← x^(h)_UB + x^(v)_UB − 1

        { Update the multipliers }

        λ ← max(λ + ts (Z_UB − Z*_LB) / ‖s‖^2 , 0)

        If λ > r then λ ← r

        { Update the step constants }

        t ← 0.98t

    End While

        (i_B, j_B) ←ChooseBranchingVariable(m, n, r, a^(h), a^(v), λ_0)

    Return (Z*_UB, x*_UB, Z*_LB, x*_LB, λ*, i_B, j_B)

End
```

Figura 3.2: Pseudocodice dell'algoritmo del sottogradiente.

La violazione è calcolata come:

$$s_{ij} = x^{(h)}_{ij_{UB}} + x^{(v)}_{ij_{UB}} - 1 \qquad i = 1..m,\ \ j = 1..n \tag{3.18}$$

Osservando la formula notiamo subito che se la soluzione sceglie di fotografare lo shard (i,j) in una direzione soltanto avremo che $x^{(h)}_{ij} = 1$ e $x^{(v)}_{ij} = 0$ oppure $x^{(h)}_{ij} = 0$ e $x^{(v)}_{ij} = 1$. In questi casi $s_{ij} = 0$ e la funzione obiettivo non viene penalizzata. Nel caso l'algoritmo scelga lo shard in entrambe la direzioni avremmo che $s_{ij} = 1$ e la funzione obiettivo sarà penalizzata. Se l'algoritmo sceglie di non fotografare lo shard (i,j) in alcuna direzione, avremo $x^{(h)}_{ij} = x^{(v)}_{ij} = 0$, $s_{ij} = -1$ e il moltiplicatore λ_{ij} comparirà nella funzione obiettivo a causa del termine costante $\left(\sum_{i=1}^{m}\sum_{j=1}^{n}\lambda_{ij}\right)$.

Abbiamo implementato due differenti algoritmi per l'aggiornamento dei moltiplicatori lagrangiani a partire dalla matrice delle violazioni.

Sottogradiente classico: È proposto da Held e Karp in [15] e consiste nel calcolare la variazione T_{ij} dei moltiplicatori con la seguente formula:

$$T_{ij} = \frac{t(Z_{UB} - Z_{LB})}{\|s_{ij}\|^2} \qquad i = 1..m,\ \ j = 1..n \tag{3.19}$$

dove Z_{UB} e Z_{LB} sono rispettivamente i valori della soluzione del rilassamento e dell'euristica, mentre il parametro t assume valori che vanno da 2 a 0. Il valore iniziale di t e la regola di aggiornamento di questo parametro sono stati identificati in modo sperimentale dopo una breve campagna di test. Abbiamo scelto di aggiornare t moltiplicandolo per 0.98 ad ogni iterazione di sottogradiente.

Il valore dei moltiplicatori è quindi aggiornato come segue:

$$\lambda_{ij}^{'} = \begin{cases} 0 & \text{se}(\lambda_{ij} + T_{ij}s_{ij}) < 0 \\ \lambda_{ij} + T_{ij}s_{ij} & \text{se}(\lambda_{ij} + T_{ij}s_{ij}) < r_{ij} \\ r_{ij} & \text{se}(\lambda_{ij} + T_{ij}s_{ij}) \geq r_{ij} \end{cases} \quad i = 1..m, \;\; j = 1..n \tag{3.20}$$

Il valore di $\lambda_{ij}^{'}$ è costretto a rimanere nell'intervallo [0,r_{ij}] perchè

1. Se $\lambda_{ij}^{'} < 0$, il risultato problema LR non è più necessariamente un upper bound.
2. Se $\lambda_{ij}^{'} > r_{ij}$, il risultato del problema LR è sicuramente un valore peggiore, cioè più alto, del valore ottenuto con $\lambda_{ij}^{'} = r_{ij}$. Infatti i reward lagrangiani $r_{ij}^{(\lambda)}$ della formula (3.9) diventano minori di zero, quindi la soluzione lagrangiana avrà $x_{ij}^{(h)} = x_{ij}^{(v)} = 0$. D'altra parte, il termine costante $\left(\sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij} \right)$ risulta più alto.

Sottogradiente modificato: Proposto da Camerini *et al* in [16], è costruito a partire dall'osservazione che l'algoritmo precedente, utilizzando i parametri T_{ij}, aggiorna i moltiplicatori con un andamento molto oscillante, cioè quelli che variando fortemente la loro direzione di aggiornamento. Più efficiente sarebbe conservare una direzione media nell'aggiornamento. A tale scopo, i moltiplicatori vengono aggiornati secondo una regola che tiene conto non solo delle violazioni s_{ij} correnti, ma anche di quelle registrate nei passi precedenti. L'aggiornamento è ottenuto come combinazione lineare delle violazioni s_{ij} correnti con quelle precedenti.

Esiste una tecnica per non penalizzare troppo le soluzioni che non fotografano i segment in alcuna direzione. Infatti nel caso un vincolo non sia violato ed il relativo moltiplicatore lagrangiano sia nullo, possiamo considerarlo come vincolo non violato. A tale proposito abbiamo applicato una tecnica di uso comune in letteratura, che consiste nell'ignorare i vincoli che risultano sovra-rispettati (cioè quelli con $s_{ij} < 0$) e che hanno un moltiplicatore nullo. Tali vincoli, infatti, sono spesso ridondanti e tenerne conto nel procedimento ha l'effetto di rallentarlo dato che $\|s\|^2$ aumenta e quindi il passo di aggiornamento si accorcia.

3.4 Il lower bound

Nelle sezioni precedenti abbiamo identificato i vincoli (2.4) come i più critici nella risoluzione del SSSP. La nostra euristica è stata ideata proprio per rispettare questi vincoli, ma contemporaneamente trovare una soluzione ammissibile di buona qualità.

Una possibile euristica applicabile a questo problema consiste nel calcolare dapprima una soluzione per gli m problemi di knapsack indipendenti associati alla direzione orizzontale, ignorando i passaggi verticali. Quindi si ottimizzano gli n problemi di knapsack associati alla direzione verticale evitando tutti gli shard già ripresi durante i passaggi orizzontali.

La nostra soluzione euristica è scelta confrontando la soluzione proposta qui sopra e quella ottenuta calcolando prima la soluzione verticale ed in seguito quella orizzontale. La soluzione finale è lamigliore tra le due.

3.5 Strategia di branching

La scelta della strategia di branching per la generazione dei nodi figli è fortemente influenzata dalla tipologia di problema. Nella nostra implementazione i figli sono generati concentrandosi su uno shard ben definito e imponendone nella matrice della soluzione finale tutti i fissaggi ammissibili, cioè forzando l'inserimento dello shard in orizzontale o in verticale, o non inserendolo. Il branching implementato è ternario, cioè l'algoritmo genera tre nodi figli:

1. Lo shard è fotografato solo durante il passaggio orizzontale ed è proibito acquisirlo durante quello verticale ($x_{ij}^{(h)} = 1$ e $x_{ij}^{(v)} = 0$).
2. Lo shard è fotografato solo durante il passaggio verticale ed è proibito acquisirlo durante quello orizzontale ($x_{ij}^{(h)} = 0$ e $x_{ij}^{(v)} = 1$).
3. Lo shard non è acquisito ($x_{ij}^{(h)} = 0$ e $x_{ij}^{(v)} = 0$).

In casi particolari può avvenire che la procedura di calcolo delle penalità lagrangiane descritta nella sezione 3.7 abbia già proibito di fotografare lo shard in una delle due direzioni. Di conseguenza il branching eseguito sarà binario:

1. Lo shard è acquisito solo nell'altra direzione.
2. Lo shard non è acquisito.

La scelta della variabile di branching è guidata anch'essa dal rilassamento lagrangiano. Se vi sono shard fotografati in entrambe le direzioni dalla soluzione lagrangiana, si usa come shard di branching quello fra loro che è associato al λ_{ij} massimo. L'obiettivo è ottenere tre sottoproblemi il più possibile diversi dal problema padre. Se nessuno shard è ripreso in entrambe le direzioni si cerca lo shard con λ_{ij} massimo. Infine se tutti gli shard sono

ripresi in una direzione, necessariamente il gap tra upper bound e lower bound è nullo, e il problema corrente è risolto all'ottimo.

3.6 Strategia di visita

Nel branch and bound l'albero è implementato con una lista bidirezionale con sentinella. Questa struttura dati consente di realizzare in modo semplice ed efficiente le strategie di visita dell'albero [17]. Supponendo di estrarre l'elemento da analizzare sempre dalla testa di questa lista, le varie strategie di visita si ottengono cambiando la modalità di inserimento dei nuovi nodi nella lista stessa. Abbiamo quindi implementato le seguenti strategie di visita:

***Depth first*:** inserimento del nuovo elemento in testa alla lista.

***Breadth first*:** inserimento del nuovo elemento in coda alla lista.

***Best first*:** inserimento del nuovo elemento in modo che la lista rimanga ordinata in modo decrescente secondo il valore di upper bound dei nodi che la compongono.

***Worst first*:** inserimento del nuovo elemento in modo che la lista rimanga ordinata in modo crescente secondo il valore di upper bound dei nodi che la compongono.

La strategia depth first corrisponde ad una visita in profondità dell'albero, quella breadth first invece visita in ampiezza. Le due strategie di maggiore interesse sono la best first e la worst first. La ricerca best first permette di cercare la soluzione nel nodo più promettente, cioè quello che presenta il valore più elevato di upper bound, dato che in esso è più probabile che si trovino soluzioni migliori di quella attuale. La ricerca worst first, al contrario, cerca la soluzione nei nodi che presentano un piccolo gap tra upper e lower bound allo scopo di giungere rapidamente alla loro chiusura.

3.7 Le penalità lagrangiane

L'idea che sta alla base dell'algoritmo delle penalità lagrangeane è quella di provare a fissare, uno per volta, i valori di tutte le variabili $x_{ij}^{(h)}$ e $x_{ij}^{(v)}$, valutando il peggioramento nella funzione obiettivo del problema LR dovuto a ciascun fissaggio, che equivale al peggioramento dell'upper bound. Se l'upper bound risulta peggiore della miglior soluzione euristica nota (il lower bound) o anche di pari valore, il tentativo di fissaggio è da considerarsi fallito, e quindi la variabile viene fissata al valore complementare.

L'algoritmo da noi implementato si può dividere in due parti: una che interessa le variabili della soluzione orizzontale ($x_{ij}^{(h)}$), l'altra coinvolge le variabili della soluzione verticale ($x_{ij}^{(v)}$).

Per ogni riga i della matrice delle variabili della soluzione orizzontale, l'algoritmo calcola il contributo $c_i^{(h)}$ che essa fornisce al valore finale del rilassamento. Per ogni variabile esegue i seguenti controlli:

- Se $x_{ij}^{(h)} = 1$, l'algoritmo impone $x_{ij}^{(h)} = 0$ e ricalcola la soluzione di riga $s_i^{(h)}$ rispettando il fissaggio della variabile $x_{ij}^{(h)}$. A questo punto, se $\left|c_i^{(h)} - s_i^{(h)}\right| \geq |Z_{UB} - Z_{LB}^*|$ significa che la variazione da noi applicata fa peggiorare la soluzione fino ad escludere che essa possa essere meglio di quella euristica. Quindi rinunciamo al tentativo e fissiamo $x_{ij}^{(h)} = 1$.
- Se $x_{ij}^{(h)} = 0$, l'algoritmo impone $x_{ij}^{(h)} = 1$ e ricalcola il nuovo gap, come al punto precedente, eventualmente concludendo che la variabile deve essere fissata a 0.

Per quanto riguarda le variabili $x_{ij}^{(v)}$, l'algoritmo lavora nello stesso modo, ma calcola il contributo $c_j^{(v)}$ della colonna j e controlla se accade che $\left|c_j^{(v)} - s_j^{(v)}\right| \geq |Z_{UB} - Z_{LB}^*|$, dove $s_j^{(v)}$ è il valore della soluzione di colonna una volta fissata la variabile $x_{ij}^{(v)}$.

I fissaggi eseguiti dall'algoritmo delle penalità lagrangeane sono rispettati durante il calcolo dell'upper bound, e del lower bound.

Capitolo 4

Prove sperimentali

4.1 La generazione delle istanze

Smith e Knight [9] riportano alcuni risultati dei test da loro eseguiti. Essi non definiscono, però, come sono state create le loro istanze di test. Questo fatto ci ha impedito di eseguire un confronto diretto fra il nostro algoritmo di risoluzione ed il loro, e ci ha portato alla creazione di un algoritmo per la generazione delle istanze sulle quali abbiamo eseguito le nostre prove. In questo capitolo tratteremo le problematiche affrontate nella generazione delle istanze.

Per semplicità abbiamo assunto che il numero di shard orizzontali e verticali fossero identici, abbiamo quindi lavorato solo con matrici quadrate. Il software da noi creato, comunque accetta e risolve anche istanze che presentano un diverso numero di shard orizzontali e verticali. Abbiamo identificato due grandi classi di dimensione:

1. Istanze piccole: la dimensione varia da $m = n = 10$ a $m = n = 100$, per passi da 10.

2. Istanze grandi: la dimensione varia da $m = n = 200$ a $m = n = 500$, per passi da 100.

Abbiamo utilizzato la prima classe per verificare la capacità dell'algoritmo di risolvere all'ottimo il problema, la seconda per valutare la sua capacità di trovare soluzioni di qualità molto buona (rispetto a un bound) in tempi ragionevoli. Tutte le istanze sono state utilizzate per confrontare i risultati da noi ottenuti con quelli di un problem solver generico: CPLEX 8.0 con impostazioni di default.

I valori di area e reward dei segment delle istanze generate sono stati creati utilizzando la funzione `ran_1` descritta in [18]. Tale funzione viene inizializzata con un seme negativo e lo modifica per generare numeri casuali con una distribuzione uniforme nel range [0,1]. I numeri generati, a parità di seme iniziale, sono identici. In questo modo siamo riusciti ad ottenere istanze con valori casuali riproducibili.

Abbiamo identificato due range di valori per le aree ed i reward, al fine di verificare il comportamento del nostro algoritmo in presenza di problemi più o meno complessi. I range che abbiamo determinato sono:

S: I valori generati da `ran_1` vengono scalati nel range [0,100].

L: I valori generati da `ran_1` vengono scalati nel range [5.000,10.000].

Otteniamo quindi le quattro combinazioni:

SS $a_{ij}^{(h)} = a_{ij}^{(v)} \in [0, 100], r_{ij} \in [0, 100]$.

SL $a_{ij}^{(h)} = a_{ij}^{(v)} \in [0, 100], r_{ij} \in [5.000, 10.000]$.

LS $a_{ij}^{(h)} = a_{ij}^{(v)} \in [5.000, 10.000], r_{ij} \in [0, 100]$.

LL $a_{ij}^{(h)} = a_{ij}^{(v)} \in [5.000, 10.000], r_{ij} \in [5.000, 10.000]$.

Abbiamo scelto di imporre $a_{ij}^{(h)} = a_{ij}^{(v)}$ in quanto il problema di scelta della direzione nella quale acquisire lo shard (i,j) è più complesso che nel caso in cui le aree orizzontale e verticale siano diverse fra loro. Infatti l'algoritmo non ha motivo, a priori, di preferire una direzione rispetto all'altra.

Tutti i valori generati nei test sono interi, ma l'algoritmo riesce a risolvere anche istanze con valori di reward frazionari.

I valori di downlink di riga e colonna sono calcolati come segue. Anzi tutto si valutano le seguenti quantità:

$$D^{(h)} \leftarrow \min_{i=1..m} \sum_{j=1}^{n} a_{ij}^{(h)} \tag{4.1}$$

$$D^{(v)} \leftarrow \min_{j=1..n} \sum_{i=1}^{m} a_{ij}^{(v)} \tag{4.2}$$

Per definire diversi gradi di difficoltà delle istanze, abbiamo assegnato ad ogni capacità di downlink una frazione del valore calcolato con le formule (4.1) e (4.2). Le classi da noi identificate sono:

1. $d_i^{(h)} = 0.2D^{(h)}$ e $d_j^{(v)} = 0.2D^{(v)}$
2. $d_i^{(h)} = 0.3D^{(h)}$ e $d_j^{(v)} = 0.3D^{(v)}$
3. $d_i^{(h)} = 0.4D^{(h)}$ e $d_j^{(v)} = 0.4D^{(v)}$

Passando dalla classe 1 alla classe 3 delle percentuali di downlink, l'istanza diventa sempre più facile da risolvere. Infatti la maggiore capacità del downlink permette all'algoritmo di Pisinger di scegliere più segment da inserire nella soluzione. Essendovi 14 classi rispetto alla dimensione del problema, 4 rispetto alla struttura di area e reward e 3 rispetto alla capacità di downlink, le istanze generate sono $14 \times 4 \times 3 = 168$.

Il nome di ciascun file riassume tutte le caratteristiche dell'istanza che contiene, secondo la seguente convenzione (il simbolo $\oplus$ indica concatenazione):

`n<classe_dimensione>` $\oplus$

$\oplus$ `d<percentuale_downlink>` $\oplus$

$\oplus$ `a<classe_area>` $\oplus$

$\oplus$ `r<classe_reward>` $\oplus$

$\oplus$ `i<numero_istanza>` $\oplus$

$\oplus$ `.dat`

Alcuni esempi sono:

- `n100d20a1r3i1.dat`
- `n200d30a1r1i4.dat`
- `n10d40a3r1i5.dat`

Dal nome del file, è possibile anche risalire al valore del seme di generazione che viene passato alla funzione `ran_1` per la generazione dei numeri casuali. Esso è dato dalla giustapposizione dei valori di seguito indicati nell'ordine in cui compaiono. Ricordiamo che il seme deve essere un numero negativo, quindi, una volta calcolato il suo valore, ne invertiremo il segno.

`-<numero_istanza>` $\oplus$

$\oplus$ `<classe_area>` $\oplus$

$\oplus$ `<classe_reward>` $\oplus$

$\oplus$ `<percentuale_downlink>` $\oplus$

$\oplus$ `<classe_dimensione>`

I semi dei tre file di esempio riportati sono quindi:

- `-11320100`
- `-41130200`
- `-5314010`

4.2 Ambiente e parametri di simulazione

Il software da noi sviluppato è stato scritto in linguaggio C, il compilatore utilizzato per creare il file eseguibile con il quale effettuare le prove è il GNU C Compiler (gcc) con opzioni di compilazione `-O3 -mfpmath=sse -std=c99`. Per effettuare tutte le prove sono stati utilizzati Personal Computer da 1.6GHz.

Il solver è stato tarato con i seguenti parametri:

- Il minimo gap accettabile tra upper bound e lower bound per chiudere un nodo è stato impostato a 1, perchè i dati sono tutti interi.
- Il minimo gap entro il quale entra in azione l'algoritmo delle penalità lagrangeane è 2.
- La strategia di visita dei nodi dell'albero di branching scelta è best first.
- Il moltiplicatore t_0 inizial usato per il calcolo della variazione dei moltiplicatori lambda è 2.
- Il numero di iterazioni del sottogradiente nel nodo radice dell'albero di branching è stato impostato a 1000, mentre nei nodi figli vengono eseguite 100 iterazioni.

4.3 Sperimentazione su varianti dell'algoritmo

In questa sezione presenteremo i risultati ottenuti applicando alcune varianti del nostro algoritmo.

La prima variante è il sottogradiente modificato presentato nella sezione 3.3.1. Questo algoritmo si è rivelato poco efficace, in quanto il valore dell'upper bound assume un comportamento oscillante nonostante il vettore del gradiente sia aggiornato in modo da evitare brusche oscillazioni.

La correzione alla formula classica di aggiornamento dei moltiplicatori λ_{ij} che ignora ogni vincolo sovra-rispettato con moltiplicatore nullo durante il calcolo della norma del sottogradiente ha anch'essa portato ad un peggioramento nei risultati.

Gli esiti negativi di tutte le prove hanno portato a scartare provvisoriamente le varianti descritte, che meritano comunque un approfondimento futuro.

Un algoritmo che invece è presente nel nostro risolutore è quello delle penalità lagrangiane (sezione 3.7). Tale algoritmo si è rivelato molto utile solo in presenza di piccoli gap. La lentezza di questo algoritmo ha impedito di applicarlo costantemente nel risolutore, l'algoritmo entra infatti in azione quando il gap presente è già minimo e lo chiude. La lentezza dell'algoritmo è dovuta principalmente al fatto che esso è costretto a risolvere molti sottoproblemi di knapsack.

L'idea per il futuro è quella di evitare di ricalcolare tutti i sottoproblemi, ma di ricalcolare solo quelli che effettivamente potrebbero trovare soluzioni più promettenti.

4.4 Prove sperimentali sulle istanze piccole

Nelle tabelle 4.1 e 4.2 riportiamo i risultati ottenuti rispettivamente dal nostro algoritmo e da CPLEX sulle istanze da $m = n = 10$ a $m = n = 100$. Le prove sono state eseguite impostando un tempo limite di 30 minuti. Nella prima colonna è riportato il numero di righe e colonne, nella seconda il range assegnato ai valori dell'area (S o L), e nella terza il range assegnato al reward (S o L). Le colonne successive riportano i risultati sperimentali. Esse sono ulteriormente suddivise in due gruppi: soluzione al nodo radice e soluzione migliore. Per ogni gruppo, la prima colonna riporta il valore di upper bound, la seconda il lower bound, la terza il gap tra i due valori, la quarta il gap in percentuale definito come $((Z_{UB} - Z^*_{LB})/Z^*_{LB})$ e l'ultima il tempo di CPU necessario per ottenere quel risultato. Il tempo di CPU necessario a CPLEX a trovare una soluzione nel nodo radice è omesso.

I risultati sono medie sulle tre istanze con pari valore di N,A ed R, ma appartenenti ognuna ad una classe di downlink differente (20%, 30%, 40%). Abbiamo riportato i valori medi, in quanto i risultati delle diverse istanze raggruppate non si discostavano molto tra loro. Si può notare che:

1. I gap alla radice ottenuti dal nostro algoritmo sono molto ristretti. Il gap massimo è 2,87%, ma nella media il gap non supera il valore 1%.

2. Il problema è difficile, infatti le istanze risolte all'ottimo sono tutte confinate, sia per il nostro algoritmo sia per CPLEX, nella classe con valori $m = n = 10$.

3. Tra i vari tipi di istanze, quelle che presentano area e reward nel range LL si rivelano le più complesse. Per tali istanze non peggiora il gap alla radice, ma aumenta il tempo necessario per trovare lo stesso gap.

Istanza			Nodo radice					Migliore				
N	**A**	**R**	**UB**	**LB**	**Gap**	**Gap (%)**	**CPU**	**UB**	**LB**	**Gap**	**Gap (%)**	**CPU**
10	S	L	523073	516043	7030	1.44%	0.23	521455		0	0.00%	0.03
10	L	S	3697	3673	24	0.54%	0.11	3696		0	0.00%	600.02
10	L	L	484457	482523	1934	0.36%	0.21	482557		0	0.00%	499.28
20	S	L	2113204	2068371	44833	2.23%	1.51	2111271	2072366	38906	1.90%	> 30 min
20	L	S	16305	16055	250	1.51%	8.70	16298	16116	181	1.02%	> 30 min
20	L	L	2058719	2022638	36080	1.73%	28.14	2057999	2032503	25496	1.11%	> 30 min
30	S	L	4833485	4771991	61495	1.31%	2.66	4833485	4771991	61495	1.31%	> 30 min
30	L	S	36050	35585	465	1.35%	316.19	36050	35587	463	1.35%	> 30 min
30	L	L	4588171	4535102	53069	1.22%	531.17	4588171	4540093	48078	1.09%	> 30 min
40	S	L	8674055	8593669	80386	0.97%	4.05	8674055	8596010	78045	0.94%	> 30 min
40	L	S	65582	64885	697	1.10%	1003.78	65582	64887	695	1.10%	> 30 min
40	L	L	8071111	8012687	58425	0.75%	953.88	8071111	8012687	58425	0.75%	> 30 min
50	S	L	14007231	13923159	84071	0.62%	5.99	14007231	13926230	81000	0.60%	> 30 min
50	L	S	100048	99220	828	0.86%	1313.62	100026	99232	794	0.82%	> 30 min
50	L	L	12726032	12657285	68747	0.55%	> 30 min	12726032	12657285	68747	0.55%	> 30 min
60	S	L	20253054	20163541	89513	0.45%	8.90	20253054	20166154	86900	0.44%	> 30 min
60	L	S	146516	145485	1030	0.72%	> 30 min	146516	145485	1030	0.72%	> 30 min
60	L	L	18433555	18358895	74660	0.42%	1665.88	18433216	18358895	74321	0.42%	> 30 min
70	S	L	27379068	27285609	93459	0.35%	11.55	27379068	27286037	7721721	0.35%	> 30 min
70	L	S	198556	197344	1212	0.62%	> 30 min	198556	197344	1212	0.62%	> 30 min
70	L	L	24999229	24917385	81844	0.34%	1764.01	24998289	24917385	80904	0.34%	> 30 min
80	S	L	35847219	35744150	103068	0.30%	14.71	35847219	35744195	103023	0.30%	> 30 min
80	L	S	260381	258896	1485	0.58%	> 30 min	260381	258896	1485	0.58%	> 30 min
80	L	L	32768483	32670682	13466246	0.31%	> 30 min	32768483	32670682	97801	0.31%	> 30 min
90	S	L	45122583	45011218	111365	0.25%	20.98	45122583	45011218	111365	0.25%	> 30 min
90	L	S	329373	327581	1792	0.55%	> 30 min	329373	327581	1792	0.55%	> 30 min
90	L	L	41615258	41505861	109397	0.26%	> 30 min	41615258	41505861	109397	0.26%	> 30 min
100	S	S	420021	418497	1524	0.36%	24.53	419714	418504	1210	0.29%	> 30 min
100	L	L	51300603	51172012	128591	0.25%	> 30 min	51300603	51172012	128591	0.25%	> 30 min

Tabella 4.1: Risultati delle prove effettuate su istanze piccole con il nostro solver

Istanza			Nodo radice				Migliore				
N	**A**	**R**	**UB**	**LB**	**Gap**	**Gap (%)**	**UB**	**LB**	**Gap**	**Gap (%)**	**CPU**
10	S	L	523785	507767	16018	3.22%	521455		0	0.00%	3.23
10	L	S	3709	3525	185	4.55%	3696		0	0.00%	3.22
10	L	L	486470	466072	20399	3.78%	482557		0	0.00%	603.69
20	S	L	2113976	2023062	90914	4.58%	2113940	2073528	40412	2.12%	> 30 min
20	L	S	16368	15708	660	4.42%	16331	16019	312	1.93%	> 30 min
20	L	L	2062665	1994227	68438	3.50%	2062313	2011240	51072	2.55%	> 30 min
30	S	L	4833464	4714921	118543	2.55%	4833455	4742532	90923	1.96%	> 30 min
30	L	S	36057	35158	899	2.74%	36057	35238	819	2.44%	> 30 min
30	L	L	4589057	4489103	99955	2.34%	4589054	4489103	99951	2.34%	> 30 min
40	S	L	8673847	8532881	140966	1.71%	8674001	8532890	141111	1.71%	> 30 min
40	L	S	65539	64410	1129	1.85%	65539	64410	1129	1.85%	> 30 min
40	L	L	8074345	7976851	97495	1.32%	8074345	7976851	97494	1.32%	> 30 min
50	S	L	14007333	13843667	163667	1.23%	14007074	13843623	163451	1.23%	> 30 min
50	L	S	99947	98625	1322	1.41%	99947	98625	1322	1.41%	> 30 min
50	L	L	12724170	12598640	125530	1.01%	12724215	12598774	125442	1.01%	> 30 min
60	S	L	20252333	20067000	185333	0.95%	20252243	20067123	185120	0.95%	> 30 min
60	L	S	146324	144868	1456	1.05%	146324	144868	1456	1.05%	> 30 min
60	L	L	18429667	18282667	147000	0.83%	18429463	18282573	146891	0.83%	> 30 min
70	S	L	27377333	27182667	194667	0.73%	27377336	27182535	194801	0.73%	> 30 min
70	L	S	198228	196505	1723	0.93%	198228	196505	1723	0.93%	> 30 min
70	L	L	24989667	24813333	176333	0.73%	24989573	24813052	176521	0.73%	> 30 min
80	S	L	35844333	35609333	235000	0.67%	35844544	35609365	235179	0.67%	> 30 min
80	L	S	259955	258258	1697	0.70%	259955	258258	1697	0.70%	> 30 min
80	L	L	32758177	32569388	188789	0.60%	32758224	32569220	189004	0.60%	> 30 min
90	S	L	45122179	44943533	178645	0.43%	45122052	44943602	178451	0.43%	> 30 min
90	L	S	329122	327149	1973	0.63%	329122	327149	1973	0.63%	> 30 min
90	L	L	41615202	41505559	109643	0.27%	41615202	41505559	109643	0.27%	> 30 min
100	S	S	419572	417557	2015	0.48%	419572	417557	2015	0.48%	> 30 min
100	L	L	51266333	51050333	216000	0.44%	51265999	51050324	215674	0.44%	> 30 min

Tabella 4.2: Risultati delle prove effettuate su istanze piccole con CPLEX 8.0

Dalle prove sperimentali sembra che le tre classi si dispongano in questo ordine di difficoltà: SL, LS ed LL.

4. Le classi SS sono state omesse in quanto sono risultate molto più semplici rispetto alle altre.

5. Il gap finale ottenuto dal nostro algoritmo è quasi sempre minore di quello al nodo radice, questo significa che il branch and bound non riesce a risolvere le istanze nel tempo limite, però dà buoni risultati, in quanto riesce a diminuire il gap tra upper e lower bound.

6. In generale i gap di CPLEX al nodo radice sono peggiori (più grandi) di quelli ottenuti con il nostro algoritmo. Questo accade perchè CPLEX utilizza un rilassamento di tipo lineare, mentre il nostro solver utilizza un rilassamento lagrangiano che tiene conto dei problemi di knapsack. CPLEX però aggiunge automaticamente tagli per rafforzare il rilassamento lineare e quindi non è ovvio che il suo risultato sia peggiore. La nostra sperimentazione dimostra che tali tagli sono insufficienti a recuperare lo svantaggio rispetto al nostro algoritmo.

7. Il gap trovato da CPLEX è generalmente più largo di quello trovato dal nostro solver su entrambi i versanti: infatti spesso, l'upper bound di CPLEX è più alto del nostro, mentre il lower bound è più basso.

8. La strategia di branching di CPLEX appare molto meno efficiente: infatti il gap presente al nodo radice, oltre ad essere superiore al nostro, spesso non decresce.

4.5 Prove sperimentali sulle istanze grandi

In tabella 4.3 riportiamo le istanze da $m = n = 200$ a $m = n = 500$. Da tali istanze si è risolto solo il problema al nodo radice. Inoltre le prove sono state eseguite impostando in tempo limite di 30 minuti per il nostro solver e di 60 minuti per CPLEX, in quanto, non sempre CPLEX riesce a risolvere il nodo radice nel giro di 30 minuti, e in tali casi non fornisce un bound.

La struttura della tabella è simile a quella descritta nella sezione 4.4. Anche in questa tabella i risultati sono medie sulle tre istanze che si differenziano solo per la percentuale di downlink. Le classi di istanze utilizzate per queste prove sono solo la classe SS e la classe LL dato che si sono dimostrate rispettivamente la più facile e la più difficile. Si può notare che l'ultima riga delle istanze più grandi ($m = n = 500$) del gruppo di colonne relativo a CPLEX non presenta alcun valore. Questo accade perchè CPLEX non è riuscito, entro il tempo limite di 60 minuti, a trovare una soluzione al nodo radice del problema.

I risultati confermano le osservazioni fatte nella sezione precedente, cioè che il nostro algoritmo produce un gap più ristretto sia per quanto riguarda l'upper bound (dunque il rilassamento migliore), sia per quanto riguarda il lower bound (dunque l'euristica è migliore). Inoltre i tempi di calcolo per le istanze SS (nelle quali il nodo radice viene completato prima del tempo limite di 30 minuti) risulta molto inferiore a quello di CPLEX.

Istanza			SSSP Solver				CPLEX			
N	A	R	UB	LB	Gap	Gap (%)	CPU	UB	LB	Gap	Gap (%)	CPU
200	S	S	1686136	1681701	4435	0.25%	59.12	1683662	1679567	4095	0.24%	> 60 min
200	L	L	205646381	205281255	365126	0.18%	> 30 min	205430000	205060000	370000	0.19%	> 60 min
300	S	S	3803935	3794856	9079	0.23%	130.47	3797702	3793698	4004	0.11%	> 60 min
300	L	L	464438359	463687939	750420	0.16%	> 30 min	463890000	462956667	933333	0.22%	> 60 min
400	S	S	6738449	6723268	15181	0.22%	228.15	6726996	6709464	17533	0.27%	> 60 min
400	L	L	828165136	826832078	1333058	0.16%	> 30 min	827095572	824828888	2266684	0.30%	> 60 min
500	S	S	10583805	10560991	22814	0.21%	353.82	10565520	10536587	28933	0.28%	> 60 min
500	L	L	1287810675	1285705707	2104968	0.17%	> 30 min					> 60 min

Tabella 4.3: Risultati delle prove effettuate su istanze grandi

Conclusioni e sviluppi futuri

Il SSSP è un problema interessante dal punto di vista scientifico, perchè si presta ad essere risolto con le più svariate tecniche algoritmiche. La risoluzione del SSSP permette di risparmiare inoltre molte risorse a bordo del satellite, dalla memoria al carburante. Questo risparmio si può tradurre in una migliore efficienza del satellite stesso e, di conseguenza, in un migliore servizio fornito agli utilizzatori finali. La letteratura riguardante il SSSP è piuttosto scarsa, quindi questo problema può essere ancora studiato a fondo in tutte le sue varianti. In questa tesi abbiamo proposto un algoritmo di tipo branch and bound basato su rilassamento lagrangiano. Il problema lagrangiano è un insieme di problemi di knapsack indipendenti fra loro, per cui tiene conto di aspetti del problema che verrebbero invece ignorati da un rilassamento di tipo lineare. Il nostro rilassamento si è rivelato molto efficace e permette di trovare soluzioni euristiche di ottima qualità, infatti abbiamo ottenuto piccoli gap anche su istanze molto grandi (ad esempio istanze con $m = n = 1000$ non riportate nel capitolo dei risultati). Il confronto con CPLEX mostra che il nostro algoritmo compie un errore su upper e lower bound al nodo radice mediamente molto più piccolo di quello prodotto da CPLEX. Questo conferma il fatto che sia il rilassamento, sia l'euristica sono di buona qualità.

Confrontando i risultati dei test effettuati con il nostro risolutore e quelli

ottenuti con CPLEX 8.0, abbiamo notato che solo con particolari istanze il nostro algoritmo ha fornito soluzioni peggiori di quelle trovate da CPLEX. Queste istanze sono casi particolari del SSSP. In esse, infatti, accade che $a_{ij}^{(h)} = a_{ij}^{(v)} = r_{ij}$: gli $m + n$ problemi di knapsack che ne derivano sono in realtà noti come problemi di *subset sum* per i quali esistono algoritmi di risoluzione specifici diversi da quello di Pisinger, il quale ha difficoltà a scegliere tra elementi che presentano costi e guadagni identici. Confrontando gli output di CPLEX con quelli del nostro algoritmo, abbiamo notato che, a pari velocità di clock (1.6 GHz in entrambi i casi), CPLEX arriva alla soluzione ottima nel nodo radice in poco meno di un'ora. Il nostro algoritmo, dopo due ore di calcolo, trova un errore medio del 7% circa. Questa differenza è dovuta al fatto che CPLEX utilizza, al suo interno, procedure dedicate per il calcolo di problemi di *subset sum*. L'idea è quindi quella di sviluppare ulteriormente il progetto inserendo una procedura di identificazione della tipologia di problema caricato e, nel caso esso si possa ricondurre ad uno di tipo subset sum, risolverlo con un algoritmo specifico.

Bibliografia

[1] ESA Portal. http://www.esa.int/esaCP/index.html.

[2] ESA-MetOp. http://www.esa.int/esaME/index.html.

[3] GOES POES Program - POES Home. http://goespoes.gsfc.nasa.gov/poes/index.html.

[4] Polar Orbits. http://www.newmediastudio.org/DataDiscovery/Hurr_ED_Center/Satellites_and_Sensors/Polar_Orbits/Polar_Orbits.html.

[5] GOFC-GOLD Global Observation of Forest and Land Cover Dynamics. http://www.fao.org/gtos/gofc-gold/.

[6] ESA - Living Planet Programme - GOCE. http://www.esa.int/esaLP/goce.html.

[7] IL PROGRAMMA ITALIANO COSMO-SkyMed. http://www.asi.it/sito/programmi_cosmo.htm.

[8] *Il grande dizionario Garzanti della lingua italiana*. Garzanti Editore s.p.a., Settembre 1987.

[9] R.Knight and B. Smith. *Optimal nadir observation scheduling*. Fourth International Workshop on Planning and Scheduling for Space, Darmstadt 2004.

[10] T. A. Ciriani, G. Fasano, S. Gliozzi, and R. Tadei. *Operations research in space and air*. Kluwer Academic Publishers, 2003.

[11] H. Muraoka, R.H. Cohen, T. Ohno, and N. Doi. *ASTER observation scheduling algorithm*. Space Ops 98, Tokio 1998.

[12] J. E. Beasley. Algorithms for the Steiner problem in graphs. 14:147–159, 1984.

[13] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 46(5):758–767, 1995.

[14] G. Righini A. Ceselli. A branch-and-price algorithm for the capacitated p-median problem. *Networks*, 45(3):125–142, 2005.

[15] M. Held and R. M. Karp. The travelling-salesman problem and minimum spanning trees. 18:1138–1162, 1970.

[16] P. M. Camerini, L. Fratta, and F. Maffioli. On improving relaxation methods by modified gradient techniques. *Mathematical Programming Study*, 3:26–34, 1975.

[17] G. Righini. Guida alla realizzazione di algoritmi branch-and-bound. Technical report, Dipertimento di Teconologie dell'Informazione, Sede di Crema, 9 Marzo 2000.

[18] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C The Art of Scientific Computing*. Cambridge University Press, second edition edition, 1999.