Università degli studi di Padova Facoltà di Ingegneria

Tesi di Laurea in Ingegneria Informatica

Indirizzo: Automatica e Sistemi di Automazione Industriale

Algoritmi euristici per il progetto ottimo di una rete di interconnessione

RELATORE : Prof. Matteo Fischetti

Laureando : Carlo Polo

ANNO ACCADEMICO 2000 – 2001

Indice

pag.

SOMMARIO……..…1

INTRODUZIONE…… 3

1. IL SISTEMA UMTS…….. 7

1.1 INTRODUZIONE……. 7

1.2 CELLE RADIO…… 8

1.3 ELEMENTI DI BASE…… ……. 9

1.4 ARCHITETTURA DEL SISTEMA UMTS……...10

1.5 GENERAZIONE DI ISTANZE……..14

1.6 TEST BED…… 15

2. UN MODELLO DI PROGRAMMAZIONE LINEARE MIXED-INTEGER (MIP)…… 17

2.1 CARATTERISTICHE DEL MODELLO…….. 17

2.2 MODELLO MATEMATICO…….……18

2.3 MIGLIORAMENTO DEL MODELLO…….. ……..20

2.4 RISOLUZIONE DEL MODELLO…….. 21

3. LA TABU SEARCH…… ……..23

3.1 PRESENTAZIONE ALGORITMICA…… 23

3.2 APPLICAZIONE DELLA TABU SEARCH ALLA RETE DI INTERCONNESSIONE……... …….. 24

3.3 ESEMPI DI TEST E RISULTATI……..27

4. IL PROGRAMMA ILOG-CPLEX……..…… 33

4.1 PRESENTAZIONE E CARATTERISTICHE DI FUNZIONAMENTO…….. 33

4.2 I PARAMETRI DI SETTAGGIO FONDAMENTALI…… 34

4.3 ESEMPI DI TEST E RISULTATI…….. 35

5. METODO DI RICERCA VARIABLE NEIGHBOURHOOD SEARCH (VNS)…….. 39

5.1 INTRODUZIONE……..…… 39

5.2 ALGORITMO GENERALE VNS…….……39

5.3 ESTENSIONI DEL VNS…….. 41

5.4 APPLICAZIONE DEL VNS E DELLE SUE ESTENSIONI AD UNA RETE DI INTERCONNESSIONE…….. 43

5.4.1 Espressioni e vincoli caratteristici del metodo…… 43

5.4.2 Pseudocodice dell'algoritmo…….. 48

5.4.3 Commenti allo pseudocodice……..50

5.5 ESEMPI DI TEST E RISULTATI……..…. 53

6. METODO DI RICERCA DIVERSIFICATION – REFINING – TIGHT REFINING (DRT)…… 61

6.1 CONSIDERAZIONI GENERALI……...61

6.2 APPLICAZIONE DEL DRT AD UNA RETE DI INTERCONNESSIONE……. 62

6.2.1 Vincoli con coefficienti big-M……..……62

6.2.2 Espressioni e vincoli caratteristici del metodo…… 64

6.2.3 Pseudocodice dell'algoritmo……... 68

6.2.4 Commenti allo pseudocodice…….. 70

6.3 ESEMPI DI TEST E RISULTATI…….. 72

CONCLUSIONI E PROSPETTIVE FUTURE…… 81

APPENDICE A…….. 83

APPENDICE B…….. 91

BIBLIOGRAFIA……107

Sommario

La programmazione lineare mista-intera (MIP) riveste un ruolo centrale nella modellizzazione di problemi di ottimizzazione combinatori difficili da risolvere (NP-hard). Tuttavia, a causa della loro intrinseca difficoltà, la risoluzione esatta di questi modelli spesso non può essere ottenuta per problemi (e per dimensioni) d'interesse pratico. In questo caso, si ricorre spesso all'uso di tecniche euristiche ad-hoc studiate per una *specifica* classe di problemi, perdendo così il vantaggio di lavorare nel contesto completamente generale fornito dalla MIP. In effetti, la mancanza di efficaci tecniche euristiche per affrontare modelli MIP di tipo generale viene oggi da molti considerata uno dei principali svantaggi legati all'uso pratico di modelli MIP.

Nella presente tesi si è studiato l'uso di un risolutore MIP generico (Cplex 7.0) come una procedura "a scatola nera" da utilizzare nel contesto di algoritmi euristici di ricerca locale, con l'intento di ottenere un metodo euristico generale che utilizzi le potenzialità delle tecniche esatte (branch-and-cut) per l'esplorazione efficace di ampi sottoinsiemi di soluzioni.

I contributi principali della tesi possono essere così sintetizzati:

1. si è ideato e messo a punto un nuovo ed efficace schema di risoluzione euristica (detto DRT) basato sull'uso di Cplex per l'esplorazione di intorni ottenuti mediante l'aggiunta di opportuni vincoli lineari al modello (vincoli *non* validi nel contesto dell'ottimizzazione).
2. la nuova metodologia è stata implementata in C++ ed applicata alla risoluzione di un importante e difficile problema di ottimizzazione: il progetto della rete di interconnessione in un sistema di telefonia UMTS.
3. sono state altresì sviluppate ed implementate in C++ procedure euristiche alternative, ed in particolare una tecnica tabu-search specifica che non utilizza alcun modello MIP [2], ed una tecnica di tipo Variable Neighbourhood Search (VNS), basata sull'uso di Cplex per l'esplorazione di intorni.
4. sono stati generati e resi disponibili alla comunità scientifica internazionale 13 problemi test di dimensioni medio-grandi.
5. è stata svolta un'approfondita analisi computazionale delle tecniche presentate, dimostrando la netta superiorità della nuova metodologia DRT rispetto agli altri metodi.

In particolare, quest'ultimo risultato è molto significativo in quanto la tecnica DRT utilizza la conoscenza del problema specifico ad un livello di astrazione molto elevato, consistente semplicemente nella distinzione delle variabili decisionali in due classi (le "x" e le "y"). In quest'ottica, è quindi verosimile che la tecnica possa essere resa sufficientemente generale da

poter essere applicata a problemi MIP generici: l'utente, dopo aver generato il modello MIP del suo problema, potrà (1) applicare direttamente la tecnica DRT, oppure (2) specificare (in un file aggiuntivo) quali delle variabili binarie sono di tipo "x" o "y" in modo da attivare i meccanismi di "tight refining" della tecnica DRT che sfruttano tale informazione.

INTRODUZIONE

Il mercato della telefonia cellulare è stato protagonista negli ultimi anni di uno sviluppo e di un'espansione davvero sorprendenti: l'aumento progressivo ed esponenziale del numero di utenti e l'evoluzione della tecnologia hanno permesso l'abbassamento dei prezzi di vendita dei terminali radiomobili ed il miglioramento della qualità e l'incremento della quantità dei servizi da essi offerti, con l'effetto di attirare così altre potenziali classi di utenza e di innescare una sorta di retroazione positiva che, almeno nel prossimo futuro, non sembra destinata ad arrestarsi o a smorzarsi.

Di fronte ad una tale sempre crescente e differenziata richiesta, i gestori dei servizi radiomobili si sono dovuti adeguare, realizzando una rete di distribuzione via via sempre più fitta e ramificata, efficiente in termini di prestazioni e naturalmente progettata al fine di minimizzarne il costo operativo.

La topologia di questi tipi di rete è predefinita, e si basa su di una configurazione a stella, secondo cui le Stazioni Radiobase (BTS) si connettono ai dispositivi di ordine superiore, detti Controllori (CSS), i quali a loro volta si interfacciano con dispositivi di livello ancora superiore, chiamati Autocommutatori (LE). Tale architettura "a stella" viene preferita ad altre perché permette una vantaggiosa decentralizzazione delle funzioni di controllo del sistema.

Lo scopo della presente tesi è appunto quello di fornire una metodologia al processo di pianificazione della rete, che deve essere in grado di supportare la quantità di traffico telefonico generata dagli utenti, senza andare oltre i limiti di carico previsti per i vari dispositivi e facendo in modo che i costi degli apparati e delle loro connessioni siano i più contenuti possibile.

Si tratta dunque di un classico problema di Ricerca Operativa: trovare una combinazione di valori di variabili, scelti all'interno di un insieme di soluzioni ammissibili definito da un sistema di vincoli lineari, che minimizzi una funzione obiettivo. In questo caso, però, la grossa difficoltà che si presenta è legata al fatto che molte delle variabili suddette sono intere. In effetti, esistono in letteratura algoritmi in grado di risolvere in modo esatto simili classi di problemi, ma il tempo che essi impiegano ad ottenere il minimo globale è spesso inaccettabile, soprattutto quando le dimensioni del modello sono grandi e la struttura del problema è intrinsecamente difficile (NP-hard).

In questi casi è spesso preferibile l'uso di tecniche euristiche, ossia di metodi approssimati che si accontentano di trovare, in tempi ragionevoli, una soluzione sub-ottima.

Il problema del progetto ottimo di una rete di interconnessione è già stato affrontato in passato usando come tecnica euristica per la sua risoluzione la metodologia detta *Tabu Search*. In poche parole, la Tabu Search è un metodo che, data una soluzione ammissibile del problema, esplora un suo "intorno" (ossia un insieme di soluzioni che strutturalmente differiscono di poco da quella in esame) e sceglie all'interno di questo la miglior soluzione ammissibile. Nel metodo viene inoltre mantenuta una specie di memoria sulle

ultime soluzioni visitate, in modo da non dovervi ricadere ed incorrere così nel pericolo di cicli.

Il Capitolo 1 della tesi si occuperà della presentazione formale del sistema UMTS, della terminologia usata e delle caratteristiche topologiche e tecniche della rete. Verrà inoltre descritta la generazione delle istanze che fungeranno da input (e dunque da banco di prova) ai metodi analizzati, e si fornirà l'elenco delle dimensioni di rete relative agli esempi scelti per la sperimentazione (descrizione del "test bed"). Tale elenco sarà catalogato in classi, in modo che gli esempi di input appartenenti ad una stessa classe abbiano dimensioni fra loro abbastanza simili. Tutti i dati generati saranno resi disponibili alla comunità internazionale come *benchmark* per il confronto di tecniche alternative.

Il Capitolo 2 tratterà la modellizzazione del problema, ossia la sua formulazione come minimizzazione di una funzione obiettivo sottoposta a vincoli di ammissibilità. Verrà impiegato il modello di Mixed Integer Programming (MIP) proposto nell'articolo [1], e verrà fatto cenno all'algoritmo Branch and Cut (introdotto in tale articolo) che tenta di risolverlo in modo esatto.

Nel Capitolo 3 verrà sviluppata la traccia algoritmica relativa alla tecnica Tabu Search trattata in [2], e saranno esposti i risultati ottenuti sulla base delle istanze appartenenti al nostro test bed.

Nel Capitolo 4 si presenterà il software commerciale Cplex (versione 7.0) e la tecnica che esso adotta per risolvere problemi di programmazione mixed-integer. Applicando il software al nostro modello si evidenzia una notevole difficoltà di esplorazione dell'insieme delle soluzioni da parte dell'algoritmo, nonchè la non trascurabile influenza di alcuni parametri sui risultati.

Nei Capitoli 5 e 6 verranno esaminati due nuovi metodi euristici che definiscono opportuni intorni delle soluzioni correnti mediante l'aggiunta di vincoli lineari al modello principale, intorni che verranno poi esplorati applicando Cplex al modello così ottenuto. Il primo metodo userà la filosofia VNS (Variable Neighbourhood Search) introdotta in [3], mentre il secondo applicherà una tecnica originale di ricerca a diversi livelli (DRT: Diversification, Refining, Tight refining) mantenendo la storia delle soluzioni visitate (come nella Tabu Search) tramite l'aggiunta permanente di alcuni vincoli lineari.

Al termine di ogni capitolo, i risultati ottenuti verranno confrontati con quelli dei metodi precedentemente analizzati, onde valutare se le prestazioni offerte dalla nuova tecnica sono tali da rendere quest'ultima preferibile alle precedenti.

Saranno infine presentate le conclusioni del lavoro ed alcune sue possibili prospettive di evoluzione per il futuro. Una di queste, già in fase di progettazione, è la generalizzazione di uno dei metodi originali qui esposti (il DRT) per la risoluzione di un generico problema MIP avente variabili di tipo binario. L'idea è quella di implementare un programma completamente generale che riceva in input il modello MIP (generico) da risolvere, nonché eventualmente una partizione delle variabili binarie in due classi ("livelli")

decisa dall'utente in base alla struttura del problema di studio. L'algoritmo sfrutterà quest'ultima informazione (se presente) utilizzando uno schema DRT che, invece di affrontare direttamente la risoluzione del problema complessivo, in alcune sue fasi di intensificazione fisserà le variabili di livello 1 ad opportune configurazioni di valori, e tenterà di risolvere i sottoproblemi così creati (di dimensioni evidentemente minori) minimizzando la funzione obiettivo rispetto alle variabili di livello 2 ed alle altre variabili non binarie. Nel caso in cui il sottoproblema con variabili di livello 1 fissate risultasse ancora troppo difficile da risolvere, il metodo effettuerà automaticamente una fase di "ricerca locale" mirata, in cui si cercherà di migliorare progressivamente la soluzione corrente imponendo opportuni vincoli anche sulle variabili di livello 2.

In appendice, infine, vengono riportati i file d'ingresso utilizzati dai vari metodi per acquisire i dati di partenza, e vengono presentate le funzioni "main" dei programmi C++ che implementano le tecniche VNS e DRT descritte.

1. IL SISTEMA UMTS

1.1 Introduzione

Un sistema telefonico radiomobile mira a garantire la comunicazione tra terminali mobili e qualsiasi altro tipo di terminale telefonico, sia esso fisso oppure mobile: un utente provvisto di terminale mobile dovrebbe essere raggiungibile in qualsiasi momento ed in qualunque luogo dove la copertura radio sia garantita.

La connessione tra terminali mobili e stazioni radiobase fisse è ottenuta per mezzo di onde radio. Poiché un sistema a singola antenna non può coprire l'intera area di servizio (in quanto tale scelta richiederebbe un'alta potenza di irradiazione sia da parte delle stazioni fisse che dei terminali mobili, con conseguenti gravi rischi per la salute), si è obbligati ad adottare un approccio detto "a sistemi cellulari", costituito da numerose stazioni radiobase e relativi sistemi di antenne.

L'area di copertura di ciascuna stazione è chiamata "cella" ed offre il proprio servizio ad una piccola zona, la cui dimensione può variare dai 10-100 metri (zone urbane ad alto traffico di utenza) fino a 1-20 km (zone rurali a basso traffico di utenza).

Ciascuna stazione radiobase, tradizionalmente chiamata "Base Transceiver Station" (BTS), trasmette e riceve segnali su un numero variabile di frequenze. A seconda del tipo di sistema considerato, ciascuna frequenza consente l'allocazione di un numero variabile di canali (nel caso GSM ogni frequenza trasporta 8 canali).

Qualora un utente si sposti da una cella ad una adiacente mentre è in corso una comunicazione, un nuovo canale gli viene assegnato all'interno della cella in cui è entrato, mentre il canale utilizzato nella cella appena lasciata viene liberato e reso disponibile per successive assegnazioni. Questa tecnica viene comunemente chiamata "handover". Il vantaggio di questa rigorosa partizione sta nel fatto che ogni cella potrà così usare le stesse frequenze delle celle vicine, senza il pericolo di captarne le relative comunicazioni (e dunque di sovrapporle alle proprie).

I sistemi di telecomunicazione mobile di terza generazione standard in Europa sono noti sotto il nome di Universal Mobile Telecommunication System (UMTS).

Nel seguito verranno analizzate in dettaglio le caratteristiche topologiche e l'architettura di base di un sistema UMTS.

1.2 Celle radio

Come già accennato, nelle zone densamente popolate è opportuno che la copertura avvenga tramite numerose celle di piccole dimensioni piuttosto che da poche celle molto estese. Questo fondamentalmente per tre motivi:

1) dovendo garantire la copertura solo di una piccola regione, le antenne delle BTS possono trasmettere a bassa potenza (ed altrettanto possono fare i terminali mobili), non costituendo così un pericolo per la salute dei cittadini;
2) essendo tali zone per l'appunto molto popolate, il numero di utenti per Km^2 sarà di conseguenza elevato, e dunque, se non si vuole sovraccaricare una cella di eccessivo traffico telefonico, sarà bene assegnarle una regione non troppo estesa;
3) essendo sicuramente presenti oggetti schermanti e riflettenti molto voluminosi (grattacieli e tutti gli edifici in genere), con un'unica antenna sarebbe fisicamente impossibile arrivare a coprire aree di estensione superiore alle poche centinaia di metri; è più opportuno quindi progettare più celle, ognuna dedicata ad una piccola zona priva di ostacoli.

Esistono tre tipi fondamentali di celle:

- pico-celle: sono utilizzate in aree con elevata domanda di traffico e bassa velocità dell'utenza, generalmente in ambienti chiusi di tipo domestico e affari;

- micro-celle: sono utilizzate in città o in ambienti autostradali per coprire aree con traffico a bassa mobilità (piazze, centri commerciali, stazioni di servizio). L'area di copertura coincide, in linea di massima, con la strada stessa a causa dell'elevata direttività delle antenne e dell'effetto schermante degli edifici;

- macro-celle: sono utilizzate per fornire una copertura cellulare nel senso convenzionale del termine, tipica dei tradizionali sistemi cellulari; queste celle sono in grado di garantire la copertura di aree di grandi estensioni e, se necessario, possono essere impiegate nelle aree con copertura microcellulare non continua.

Per quanto riguarda l'effettiva estensione dell'area di copertura di ciascun tipo di cella, si hanno i seguenti valori:

- da 1 a 20 Km per le macro-celle di tipo tradizionale;

- da 100 metri a 1 Km per le micro-celle in zone delimitate da edifici con perimetri da 20x200 o 20x500 metri e in autostrada mediante antenne direttive;

- da 10 a 100 metri per le pico-celle, la cui copertura è influenzata dalle riflessioni dovute ad ostacoli quali muri, spigoli, ecc., tipici degli interni degli edifici.

1.3 Elementi di base

I dispositivi che compongono l'architettura base di una rete UMTS sono i seguenti:

- **MT** (**M**obile **T**erminal): rappresenta il terminale mobile multimediale attraverso il quale gli utenti possono accedere ai servizi messi a disposizione dalla rete nell'area di copertura nella quale si trovano. E' previsto che i terminali possano essere diversi per grado di complessità e caratteristiche: si va dai semplici terminali per il solo servizio voce a quelli più complessi, che consentono l'accesso ai servizi più evoluti quali il fax, il video e la trasmissione dati.

- **BTS** (**B**ase **T**ransceiver **S**tation): costituisce l'elemento radio del sistema e rappresenta l'interfaccia dell'utente mobile con la rete ed i suoi servizi. Può essere collegata, in base alla topologia scelta, sia all'autocommutatore (LE) che al controllore (CSS) e gestisce l'accesso dell'utente alla rete assegnando i canali di traffico e segnalazione quando richiesti.

- **CSS** (**C**ell **S**ite **S**witch): è utilizzato nei soli ambienti UMTS caratterizzati da un'elevata densità di traffico. E' collegato da un lato ad un certo numero di stazioni radiobase (BTS) e dall'altro ad un autocommutatore (LE) ed ha il compito di gestire localmente il traffico e le procedure di segnalazione quando si richiudono all'interno dell'area controllata dal CSS stesso. In tal modo si evita di caricare il livello gerarchico superiore con un insieme di compiti gravosi in termini di carico di elaborazione.

- **LE** (**L**ocal **E**xchange): svolge la funzione di autocommutatore e può essere collegato direttamente sia alle BTS che ai CSS, in fuzione dell'ambiente UMTS in cui è inserito.

- **MSDP** (**M**obility and **S**ervice **D**ata **P**oint): costituisce la base dati nella quale vengono registrate tutte le informazioni necessarie per la gestione degli utenti (profilo) e della loro mobilità. In accordo con la filosofia del resto della rete, le basi dati possono essere sia centralizzate che distribuite sul territorio, posizionandole nel primo caso insieme agli LE, nel secondo con i CSS. Anche in questa situazione le due scelte sono alternative per quanto riguarda l'impatto delle procedure di segnalazione, che in un caso coinvolgeranno tutti i livelli gerarchici della rete, e nell'altro si richiuderanno, molto spesso, localmente.

- **MSCP** (**M**obility and **S**ervice **C**ontrol **P**oint): possiede le funzionalità di controllore sia per quel che riguarda le procedure di gestione della mobilità che per ciò che concerne i servizi forniti all'utente. Può accedere direttamente alle basi dati per leggere, scrivere o cancellare le informazioni in esse contenute; in tutte le architetture può essere associato agli LE ed agli MSDP, mentre l'associazione con i CSS è consentita solo nel caso degli ambienti d'affari.

1.4 Architettura del sistema UMTS

Nel problema che stiamo affrontando vengono fornite, come dati di input, le locazioni delle BTS e le locazioni potenziali dei CSS ed LE, potenziali perché il fatto di piazzarvi un dispositivo o meno è una delle variabili decisionali del modello: sarà compito del progettista decidere quali locazioni saranno effettivamente attivate e quali no (naturalmente si sottolinea che ciò riguarda esclusivamente i CSS e gli LE, in quanto le BTS sono già fissate sia in collocazione che in numero).

Si considererà un'architettura UMTS del tipo "a stella" a tre livelli: il livello più alto, costituito dai dispositivi LE attivi (scelti nel dato insieme di locazioni di LE potenziali), un livello medio, rappresentato dai CSS attivi (anch'essi scelti nel dato insieme di locazioni di CSS potenziali), ed infine il livello più basso, composto dalle BTS (le foglie della stella).

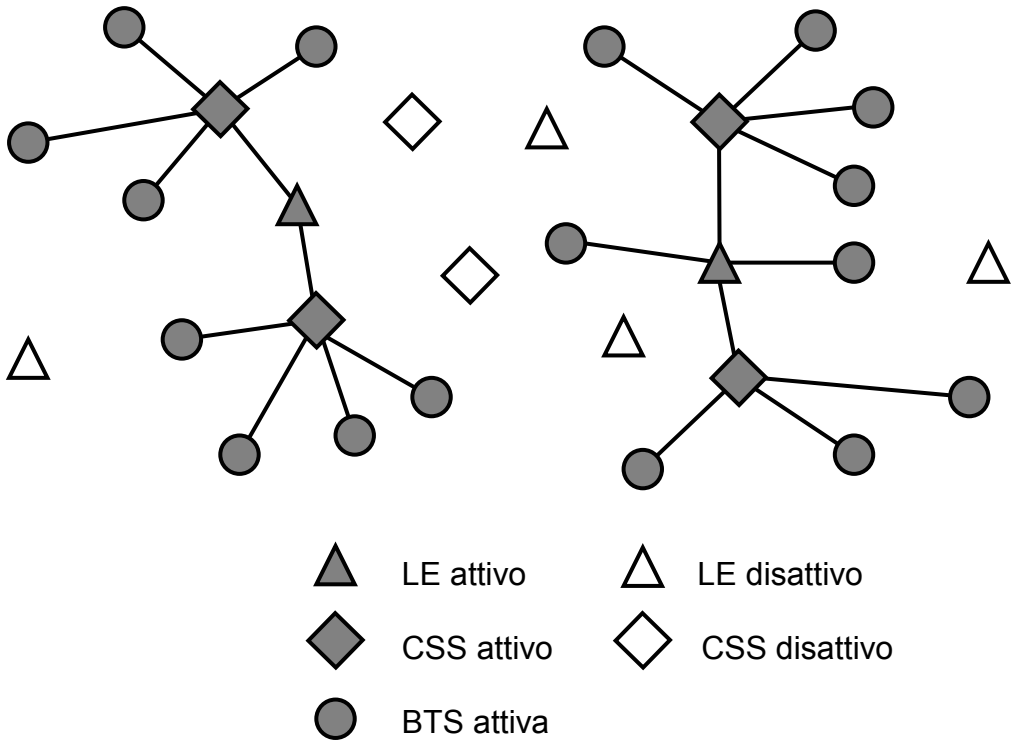

Figura 1.4.1 : Architettura UMTS a tre livelli di tipo stella

La Figura 1.4.1 mostra una situazione in cui 2 LE (su un totale di 6 potenziali) e 4 CSS (su un totale di 6 potenziali) sono attivati, e definisce un'architettura ammissibile di tipo stella per servire le 17 BTS date.

Si noti che il problema non può essere scomposto in due sottoproblemi indipendenti che consistono nell'assegnazione degli LE ai CSS e dei CSS alle BTS, in quanto la scelta dei CSS attivi e del loro carico di traffico crea uno stretto legame tra i due sottoproblemi.

Ciascuna BTS deve essere connessa al nucleo della rete o attraverso un singolo CSS attivo o direttamente attraverso un singolo LE attivo (per certe BTS pre-specificate è comunque obbligatoria la connessione verso uno solo dei due tipi di dispositivi di ordine superiore). Ogni BTS è caratterizzata dalla sua posizione geografica, la quantità di traffico che la attraversa, il numero di canali richiesti ed il suo tipo. Il posizionamento delle BTS è il risultato di un complesso processo di progettazione, che qui si assume dato, quindi il numero e la collocazione di tali dispositivi nel territorio si possono considerare fissati. In realtà, anche il numero e la posizione dei CSS e degli LE potenziali sono da considerarsi fissati già all'inizio del problema, ma essendo potenziali, nel loro caso c'è il grado di libertà di poter scegliere quali attivare e quali no. Nel prossimo paragrafo verrà illustrato il metodo usato in questa sede per posizionare le BTS ed i potenziali CSS ed LE, al fine di creare gli esempi di input sulla base dei quali testare gli algoritmi del problema. Si tratta di un metodo di creazione certamente approssimato, ma sufficiente per effettuare delle prove significative ed avere così un'idea del comportamento degli algoritmi presentati.

Il traffico trasportato ed il numero di canali uscenti da una BTS dipendono dal numero atteso medio di utenti serviti dalla cella. Più precisamente, il traffico è l'informazione totale trasmessa, ed il numero di canali è il numero di comunicazioni simultanee indipendenti, ciascuna supportata da un modulo di comunicazione (64 Kbit/s).

I canali che collegano le BTS al CSS/LE devono essere "impaccati" in moduli di data capacità (massimo numero di canali in un modulo). Nella gerarchia PCM (Pulse Code Modulation) ciascun modulo può raccogliere fino a 30 canali da 64 Kbps l'uno, garantendo così una capacità di 2 Mbps. Il tipo della BTS dipende dal tipo del dispositivo a cui è connessa.

I costi associati ad una BTS comprendono:

- il costo del dispositivo
- il costo effettivo di connessione dipendente dal CSS o LE a cui è collegata; il costo è assunto essere lineare nel numero di moduli usati (ma non nella distanza).

Ogni CSS è connesso alla rete attraverso un singolo LE, ed è caratterizzato dalla sua posizione geografica, dal tipo, dalla massima capacità di traffico, dal massimo numero di BTS collegabili e dal massimo numero di moduli in ingresso che possono essere supportati. I CSS possono

essere di due tipi differenti, chiamati “Simple” (tipo 1) o “Complex” (tipo 2), aventi diverse caratteristiche di carico massimo e di costo.

I costi associati ad un CSS comprendono:

- il costo del dispositivo, dipendente dal tipo e dalla locazione
- il costo di connessione, dipendente dall’LE a cui è collegato; il costo è assunto essere lineare nel numero di moduli usati (ma non nella distanza).

Ogni LE è caratterizzato dalla sua posizione geografica e dalla massima capacità di traffico ed il massimo numero di moduli in ingresso che possono essere supportati.

I costi associati ad un LE comprendono:

- il costo del dispositivo dipendente dalla locazione.

C’è da chiedersi come mai, se le BTS possono essere connesse direttamente agli LE, sia necessaria (o per lo meno conveniente) la presenza dei dispositivi di livello intermedio CSS (decentralizzazione delle funzioni di controllo a parte); un caso in cui tale presenza torna utile può essere messo in evidenza dall’esempio di Fig. 1.4.2. In questa situazione 4 BTS sono direttamente connesse ad un LE, che si suppone essere molto lontano; il costo della rete è dato dai costi delle BTS, i costi dei moduli di collegamento BTS-LE ed il costo dell’LE. Si supponga poi che i moduli siano piuttosto scarichi, cioè che il numero di canali che ciascuno di essi trasporta sia esiguo (come in effetti spesso accade). Un tale sottosfruttamento della capacità dei moduli, tenendo presente il loro costo al Km, è indubbiamente uno spreco di risorse.

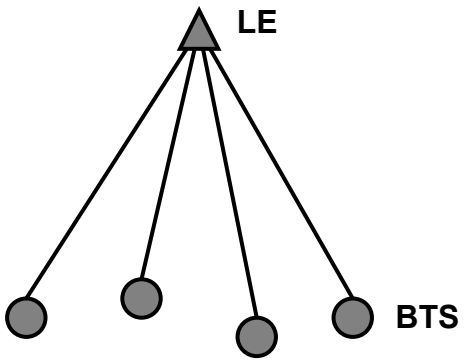

Figura 1.4.2 : esempio di rete di interconnessione senza CSS

Molto più conveniente è invece usare una configurazione di questo genere:

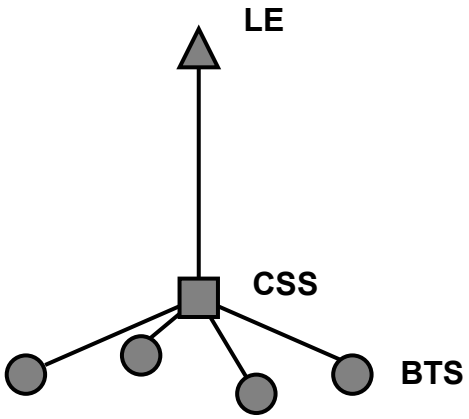

Figura 1.4.3 : esempio di rete di interconnessione con CSS

(dove si suppone che il CSS sia contiguo alle BTS). In tal modo, una volta giunti al CSS, i canali vengono impaccati in un unico modulo e trasportati all'LE attraverso di esso. Il costo per l'attivazione del CSS (elevato ma comunque sempre molto minore del costo per l'attivazione di un LE) è pertanto ripagato dal fatto che ora, invece che stendere per lunghe distanze 4 moduli poco sfruttati, se ne stende 1 solo sfruttato al meglio (ed i costi di connessione si riducono così di un fattore 4).

I vincoli di ammissibilità si possono suddividere in due categorie: vincoli di congruenza (i quali impongono che le connessioni debbano collegare tra loro solo dispositivi collocati) e "vincoli di limitazione" (i quali impongono che siano rispettati i limiti di carico massimo di tutti i dispositivi). Si potrebbe poi annoverare anche una terza categoria di vincoli, i cosiddetti "vincoli orografici", riguardanti cioè l'impossibilità fisica di connettere tra loro due dispositivi troppo lontani o divisi da ostacoli naturali quali montagne, fiumi, laghi, ecc.; tali vincoli si possono però incorporare molto semplicemente nel modello portando ad infinito i costi dei relativi collegamenti, o meglio ancora ponendo a zero le corrispondenti variabili binarie (descritte nel seguito).

I valori relativi alle massime capacità di carico (di traffico, moduli entranti e BTS afferenti) per i vari tipi di dispositivi (CSS simple, CSS complex ed LE) sono stati tratti da [2], e sono riassunti nella seguente tabella:

Tipo di dispositivo / Tipo di carico	CSS		LE
	simple	complex	
Max traffico controllabile (Erl)	350	600	3.000
Max n° moduli (2 Mbit/s) entranti	40	60	300
Max n° BTS collegabili	20	30	

Tabella 1.4.1 : valori delle max capacità di carico per i vari tipi di dispositivi

Nota: la casella corrispondente al massimo numero di BTS collegabili ad un LE è vuota in quanto non si pongono limitazioni in tal caso.

1.5 Generazione di istanze

Gli algoritmi che verranno in seguito proposti per la minimizzazione del costo della rete considerano come dati di ingresso informazioni quali le coordinate delle BTS ed il traffico associato a ciascuna di esse, nonchè le coordinate dei potenziali CSS ed LE (e quindi anche i costi di tutti i possibili collegamenti tra due dispositivi qualsiasi), in modo da lasciare come variabili decisionali solo quanti e quali CSS/LE attivare e come connetterli tra loro ed alle BTS.

Gli esempi di prova su cui gli algoritmi verranno testati sono stati creati da un “modulo generatore”, ossia un algoritmo piuttosto elementare avente il compito di generare in maniera pseudo-casuale le coordinate dei dispositivi.

Si illustreranno ora in breve le caratteristiche principali secondo cui opera tale modulo.

La regione di riferimento in cui si dispongono i dispositivi è quadrata (di dimensioni fissabili dall’utente tramite un opportuno fattore di scala) e suddivisa in tre zone quadrate concentriche A, B e C, che vogliono rappresentare rispettivamente l’area urbana, suburbana e rurale di un generico territorio geografico. Il quadrato interno A è interessato da una maggiore concentrazione dell’utenza e dunque anche da una quantità di traffico e da un numero di dispositivi maggiore; tale concentrazione diminuisce via via passando alla zona B ed infine alla C.

Il generatore sfrutta una serie di parametri (presenti in un apposito file di ingresso) quali il numero di utenti per Km^2 per regione, il raggio medio di cella, il traffico offerto per utente, il numero di BTS, di candidati CSS e di candidati LE, la massima distanza consentita per collegare due dispositivi ed il costo degli apparati e delle connessioni in base alla tecnologia utilizzata.

Il modulo generatore distribuisce in modo casuale le BTS in tutta l'area (distribuendone in quantità maggiore nelle zone più interne), i candidati CSS solo nelle regioni A e B ed i candidati LE solo nella regione A. A questo punto il modulo controlla che la configurazione di rete creata possa dar luogo a soluzioni ammissibili; se così non è, finchè i vincoli non sono soddisfatti vengono modificate le dimensioni delle regioni A, B e C, prima allargando A e poi eventualmente B (in questo processo le posizioni delle BTS vengono mantenute costanti, mentre quelle dei CSS e degli LE vengono rigenerate).

Il passo successivo è quello di calcolare il traffico associato a ciascuna cella ed il numero di canali necessario a supportarlo; poiché per creare un problema con un insieme soluzione sufficientemente ampio è necessario che il traffico totale non superi una certa soglia, vengono effettuati dei controlli in tal senso e, se la suddetta soglia viene superata, si provvede a decrementare il numero di utenti per regione fino a quando non si rientrerà nei limiti previsti.

In uscita il modulo fornisce le matrici dei costi di connessione fra tutte le coppie di dispositivi BTS-CSS, BTS-LE e CSS-LE.

1.6 Test bed

Con il termine "test bed" si vuole intendere un insieme di istanze atte a testare il comportamento degli algoritmi sviluppati; una saggia politica per la scelta di tali esempi, oltre a quella ovvia di considerarne un numero significativo, è di differenziarli il più possibile fra loro e di creare così un banco di prova sufficientemente ampio da studiare i suddetti algoritmi in svariate condizioni di funzionamento.

Nel presente lavoro sono stati considerati 13 esempi di input, suddivisi in 4 classi (con 4 esempi per classe tranne l'ultima, di dimensioni molto elevate, contenente 1 solo esempio). Gli esempi appartenenti ad una stessa classe sono caratterizzati da dimensioni di rete abbastanza simili fra loro. Passando da una classe alla successiva si ha invece un brusco incremento delle dimensioni: poiché ciò determina logicamente un notevole aumento della difficoltà di soluzione (a prescindere dall'efficacia e dalla bontà del metodo utilizzato), con tale passaggio si ammette anche un incremento del tempo limite di esecuzione destinato all'algoritmo.

Le classi 1, 2 e 3 contengono reti di input di piccole, medie e grandi dimensioni rispettivamente, la classe 4 contiene un'unica rete di grandissime dimensioni (ben maggiori di quelle considerate finora in letteratura); ciò in previsione di uno sviluppo che, in un futuro forse non troppo lontano, porterà a dover trattare reti di simile grandezza (ed anche in previsione di un confronto con nuove tecniche di soluzione che per allora saranno state sviluppate).

Viene qui di seguito illustrato l'elenco degli esempi scelti per il test bed (ciascuno caratterizzato dalla propria dimensione di rete), evidenziando la loro catalogazione nelle 4 classi (coi relativi tempi limite di esecuzione):

Classi	Esempi	Dimensioni			Tempi limite di esecuzione su PC Pentium II / 450MHz
		BTS	CSS	LE	
1	A	50	4	2	6 h
	B	55	5	3	
	C	60	5	4	
	D	65	6	4	
2	E	100	8	4	8 h
	F	105	9	5	
	G	110	9	6	
	H	115	10	6	
3	I	150	12	6	10 h
	L	155	13	7	
	M	160	13	8	
	N	165	14	8	
4	O	200	20	10	12 h

Tabella 1.6.1 : esempi costituenti il test bed

E' da notare che, aumentando il numero di BTS si aumenta il traffico gestito dalla rete, e dunque per supportare il nuovo flusso si è costretti ad aumentare sia il numero di moduli di collegamento che il numero di CSS/LE da attivare; questo si tradurrà quindi in un inevitabile incremento del costo complessivo della rete. Se, a parità di BTS, si aumenta il numero di CSS e di LE, il costo della rete tendenzialmente diminuisce, in quanto si dà alle BTS più possibilità di scelta per i collegamenti (se si suppone che i vecchi candidati CSS/LE restino dove sono e se ne aggiungano di nuovi in altre locazioni, alla peggio si evita di utilizzare i nuovi dispositivi e si sceglie la stessa configurazione di rete di prima, ottenendone lo stesso costo). Sull'onda di quanto appena detto verrebbe perciò la tentazione di ricoprire il territorio di candidati CSS ed LE, per attivare poi soltanto quelli che minimizzano il costo della rete; l'allocazione dei dispositivi non può però purtroppo sottostare a delle regole così arbitrarie (si pensi ad esempio ai problemi di tipo orografico, che rendono impossibile l'installazione di un dispositivo in mezzo ad un lago od in cima a una montagna).

Se si aumenta contemporaneamente il numero di tutti e tre i tipi di dispositivi, non si può dire a priori come i due diversi effetti sul costo complessivo della rete (aumento del costo per l'aumento di BTS da una parte, diminuzione del costo per l'aumento di CSS ed LE dall'altra) si bilanceranno: saranno quindi i risultati delle prove a fornire tale responso.

2. UN MODELLO DI PROGRAMMAZIONE LINEARE MIXED – INTEGER (MIP)

2.1 Caratteristiche del modello

Si introdurrà ora un modello matematico per il problema, basato sui seguenti dati di ingresso.

Si considera un insieme di n locazioni di BTS, un insieme di m potenziali locazioni di CSS di tipo 1 (simple) o 2 (complex) ed un insieme di p potenziali locazioni di LE.

La BTS situata nella locazione i produce un flusso di traffico t_i^{BTS} attraverso d_i^{BTS} canali di comunicazione. Tali canali sono raggruppati in "moduli" (cavi o microonde). Se Q è il massimo numero di canali che possono essere inseriti in un modulo, allora la BTS alla locazione i necessita di $e_i^{BTS} := \lceil d_i^{BTS}/Q \rceil$ moduli, dove $\lceil r \rceil = \min \{i \in N : i \geq r\}$ denota la parte intera superiore di un dato numero reale r. Vale la pena osservare che Q può in alcuni casi dipendere dalla locazione a cui un particolare modulo si connette.

Il CSS potenziale situato nella locazione j e di tipo $h \in \{1,2\}$ può supportare un flusso di traffico non superiore ad un dato limite $T_j^{CSS\text{-}h}$, un numero di moduli entranti non superiore ad $E_j^{CSS\text{-}h}$, ed un numero di BTS ad esso afferenti non maggiore di $N_j^{CSS\text{-}h}$.

L'LE potenziale situato nella locazione k può gestire un flusso di traffico non superiore ad un dato limite T_k^{LE} ed un numero di moduli entranti non superiore ad E_k^{LE}.

Una BTS può essere di tipo "basic" (il che significa che deve obbligatoriamente essere connessa ad un CSS), "isolated" (deve essere connessa direttamente ad un LE), o "free" (può essere connessa indifferentemente ad un CSS o ad un LE).

Il costo richiesto per installare un CSS di tipo h in locazione j è indicato con $f_j^{CSS\text{-}h}$, ed il costo per installare un LE in locazione k è indicato con f_k^{LE}. Il costo per attivare una BTS in locazione i e connetterla ad un CSS è $f_i^{BTS\text{-}CSS}$, mentre è $f_i^{BTS\text{-}LE}$ se la si vuole connettere direttamente ad un LE. Il costo per stendere un modulo dalla BTS in locazione i al CSS in locazione j è $c_{ij}^{BTS\text{-}CSS}$, per stenderlo dalla BTS i all'LE k è $c_{ik}^{BTS\text{-}LE}$ e per stenderlo dal CSS j all'LE k è $c_{jk}^{CSS\text{-}LE}$.

Alcune connessioni pre-specificate con moduli non sono possibili a causa dell'eccessiva distanza o di altre limitazioni tecniche.

Il problema consiste nello scegliere i CSS e gli LE che devono essere effettivamente installati ed il modo in cui connetterli fra loro e con le BTS attraverso moduli PCM, con l'obiettivo di supportare tutti i flussi di traffico che vanno dalle BTS fino agli LE, senza violare i vincoli di carico sui dispositivi e minimizzando la somma dei costi di installazione e di collegamento.

Il modello è basato sulle seguenti variabili decisionali binarie:

- y_j^{CSS-h} = 1 se e solo se un CSS di tipo h $\in$ {1,2} è attivato in locazione j
- y_k^{LE} = 1 se e solo se un LE è attivato in locazione k
- $x_{ij}^{BTS-CSS}$ = 1 se e solo se la BTS in locazione i è assegnata ad un CSS in locazione j
- x_{ik}^{BTS-LE} = 1 se e solo se la BTS in locazione i è assegnata ad un LE in locazione k
- x_{jk}^{CSS-LE} = 1 se e solo se un CSS in locazione j è assegnato ad un LE in locazione k

Il modello necessita anche della definizione delle seguenti variabili intere non negative:

- z_{jk}^{CSS-LE} ≡ numero di moduli da un CSS in locazione j ad un LE in locazione k

e delle seguenti variabili continue non negative:

- w_{jk}^{CSS-LE} ≡ flusso di traffico da un CSS in locazione j ad un LE in locazione k

2.2 Modello matematico

Con le definizioni appena date, il modello risulta:

minimise :

$$\sum_{j=1}^{m}\sum_{h=1,2} f_j^{CSS-h} y_j^{CSS-h} + \sum_{k=1}^{p} f_k^{LE} y_k^{LE} + \sum_{i=1}^{n}\sum_{j=1}^{m}\left(c_{ij}^{BTS-CSS} e_i^{BTS} + f_i^{BTS-CSS}\right) x_{ij}^{BTS-CSS} +$$

$$+ \sum_{i=1}^{n}\sum_{k=1}^{p}\left(c_{ik}^{BTS-LE} e_i^{BTS} + f_i^{BTS-LE}\right) x_{ik}^{BTS-LE} + \sum_{j=1}^{m}\sum_{k=1}^{p} c_{jk}^{CSS-LE} z_{jk}^{CSS-LE}$$

subject to :

(0) $$\sum_{j=1}^{m} x_{ij}^{BTS-CSS} + \sum_{k=1}^{p} x_{ik}^{BTS-LE} = 1 \qquad \text{for i = 1,…,n}$$

(1) $$\sum_{i=1}^{n} T_i^{BTS} x_{ij}^{BTS-CSS} \le \sum_{h=1,2} T_j^{CSS-h} y_j^{CSS-h} \qquad \text{for j = 1,…,m}$$

(2) $\sum_{i=1}^{n} x_{ij}^{BTS-CSS} \le \sum_{h=1,2} N_j^{CSS-h} y_j^{CSS-h}$ for j = 1,…,m

(3) $\sum_{i=1}^{n} e_i^{BTS} x_{ij}^{BTS-CSS} \le \sum_{h=1,2} E_j^{CSS-h} y_j^{CSS-h}$ for j = 1,…,m

(4) $\sum_{i=1}^{n} d_i^{BTS} x_{ij}^{BTS-CSS} \le Q \sum_{k=1}^{p} z_{jk}^{CSS-LE}$ for j = 1,…,m

(5) $z_{jk}^{CSS-LE} \le M_{jk} x_{jk}^{CSS-LE}$ for j = 1,…,m; k = 1,…,p

(6) $\sum_{j=1}^{m} w_{jk}^{CSS-LE} + \sum_{i=1}^{n} T_i^{BTS} x_{ik}^{BTS-LE} \le T_k^{LE} y_k^{LE}$ for k = 1,…,p

(7) $\sum_{i=1}^{n} T_i^{BTS} x_{ij}^{BTS-CSS} = \sum_{k=1}^{p} w_{jk}^{CSS-LE}$ for j = 1,…,m

(8) $w_{jk}^{CSS-LE} \le F_{jk} x_{jk}^{CSS-LE}$ for j =1,…m; k = 1,…,p

(9) $\sum_{j=1}^{m} z_{jk}^{CSS-LE} + \sum_{i=1}^{n} e_i^{BTS} x_{ik}^{BTS-LE} \le E_k^{LE} y_k^{LE}$ for k = 1,…,p

(10) $\sum_{h=1,2} y_j^{CSS-h} \le 1$ for j = 1,…,m

(11) $\sum_{k=1}^{p} x_{jk}^{CSS-LE} = \sum_{h=1,2} y_j^{CSS-h}$ for j = 1,…,m

$y_j^{CSS-h} \in \{0,1\}$ for j = 1,…,m; h = 1,2

$y_k^{LE} \in \{0,1\}$ for k = 1,…,p

$x_{ij}^{BTS-CSS} \in \{0,1\}$ for i = 1,…,n; j = 1,…,m

$x_{ik}^{BTS-LE} \in \{0,1\}$ for i = 1,…,n; k = 1,…,p

$x_{jk}^{CSS-LE} \in \{0,1\}$ for j = 1,…,m; k = 1,…,p

$z_{jk}^{CSS-LE} \ge 0$ and integer for j = 1,…,m; k = 1,…,p

$w_{jk}^{CSS-LE} \ge 0$ for j = 1,…,m; k = 1,…,p

I vincoli (0) forzano ciascuna BTS ad essere connessa o ad un CSS o ad un LE (comunque ad 1 ed un solo dispositivo di ordine superiore).

I vincoli (1), (2) e (3) impongono il limite rispettivamente sul flusso di traffico, sul numero di BTS afferenti e sul numero di moduli entranti per un dato CSS.

I vincoli (4) sono relazioni di congruenza, le quali impongono in pratica che il numero di moduli uscenti da un dato CSS sia almeno quello minimo in cui i canali entranti possono essere raggruppati.

I vincoli (5) portano a 0 il numero di moduli $z_{jk}^{CSS\text{-}LE}$ qualora $x_{jk}^{CSS\text{-}LE}$ sia 0 (ossia non esista collegamento tra CSS j ed LE k), e ne fissano un limite superiore M_{jk} qualora $x_{jk}^{CSS\text{-}LE}$ sia 1.

I vincoli (6) servono ad imporre il limite sul flusso di traffico gestibile da un LE.

I vincoli (7) esprimono il fatto che il traffico entrante in un CSS deve essere pari al traffico uscente.

I vincoli (8) portano a 0 il flusso di traffico $w_{jk}^{CSS\text{-}LE}$ qualora $x_{jk}^{CSS\text{-}LE}$ sia 0, e ne fissano un limite superiore F_{jk} qualora $x_{jk}^{CSS\text{-}LE}$ sia 1.

I vincoli (9) limitano il numero di moduli entranti in un dato LE.

I vincoli (10) impongono che non più di una CSS per locazione possa essere attivata.

I vincoli (11) impongono che, se un CSS è collocato, sia connesso ad 1 ed un solo LE (e, se non è collocato, non sia connesso a nessun LE).

Chiaramente, tutte le variabili associate a connessioni non ammissibili (perché troppo lunghe o fra BTS basic ed LE o BTS isolated e CSS) devono essere fissate a 0.

2.3 Miglioramento del modello

Poiché l'algoritmo descritto nell'articolo [1] usa il metodo branch and cut per risolvere il problema, si sono aggiunti ai vincoli già esistenti degli altri insiemi di disuguaglianze, ridondanti per il modello originale in cui è imposta la condizione di interezza sulle variabili, ma sicuramente utili durante il processo di soluzione quando la suddetta condizione di interezza viene rilassata.

I vincoli aggiuntivi sono esposti qui di seguito:

(12) $x_{ij}^{BTS-CSS} \le \sum_{h=1,2} y_j^{CSS-h}$ for i = 1,…,n; j = 1,…,m

(13) $x_{ik}^{BTS-LE} \le y_k^{LE}$ for i = 1,…,n; k = 1,…p

(14) $x_{jk}^{CSS-LE} \le y_k^{LE}$ for j = 1,…,m; k = 1,…,p

(15) $\sum_{h=1,2} y_j^{CSS-h} \le \sum_{k=1}^{p} z_{jk}^{CSS-LE}$ for j = 1,…,m

(16) $\sum_{k=1}^{p} y_k^{LE} \ge 1$

I vincoli (12) e (13) impongono rispettivamente che, se la BTS i è connessa al CSS j / all'LE k, allora il CSS j / l'LE k venga attivato.

I vincoli (14) stabiliscono che, se il CSS j è connesso all'LE k, allora l'LE k venga attivato.

I vincoli (15) forzano ciascun CSS attivo ad avere almeno 1 modulo in uscita.

Il vincolo (16) impone che vi sia almeno un LE collocato.

2.4 Risoluzione del modello

E' noto che il problema considerato è molto difficile da risolvere all'ottimalità (problema cosiddetto NP-hard), anche usando tecniche allo stato dell'arte. Ciò nonostante, nell'articolo [1] si procede all'esposizione di un metodo esatto, il quale può essere usato come metodo euristico semplicemente interrompendolo prima della convergenza.

Lo scheletro su cui è costruito l'algoritmo è quello classico per problemi MIP, ossia il branch and cut, dove la fase di risoluzione dei problemi LP relativi ai nodi dell'albero è affidata al programma commerciale Cplex 5.0 [14], mentre la fase di gestione e sviluppo dell'albero è assegnata alla versione 3.0 del programma MINTO [20].

Il metodo non utilizza i tagli internamente generati da MINTO (che vengono infatti disattivati), ma implementa la fase di taglio (cutting) in maniera esplicita, come segue: i vincoli (0) sono usati staticamente, quindi sono presenti nel modello LP che, in corrispondenza di un generico nodo dell'albero, viene costruito e risolto per primo. A questo punto viene usata la procedura di separazione per verificare se la soluzione corrente (presumibilmente frazionaria) soddisfa gli insiemi di vincoli (1), (4) e (7). Se

tali vincoli non vengono violati, allora la separazione è applicata agli insiemi (12), (13) e (14), e se tutti tali vincoli risultano soddisfatti, allora si passa all'esame del soddisfacimento dei tagli (2), (3), (5), (6), (8), (9), (10), (11), (15) e (16). La sequenza di separazione è interrotta non appena si trova una famiglia di tagli in cui almeno uno di essi viene violato. In tal caso, si risolve il modello LP costruito sulla base di tutti gli insiemi di vincoli considerati. Una volta ottenuta la nuova soluzione corrente, si ricomincerà ad applicare la procedura di separazione (si riparte dalla prima!!). Alla fine di questo processo, in corrispondenza del generico nodo che si sta analizzando si arriva così ad ottenere una soluzione che soddisfa tutti i vincoli (0)–(16). Se tale soluzione soddisfa anche i vincoli di interezza sulle variabili, allora si è trovata una soluzione ammissibile, e si confronta il suo costo con quello dell'attuale miglior soluzione ammissibile del problema (la cosiddetta soluzione incombente). In ogni caso non è necessario sviluppare l'albero oltre tale punto (il nodo in questione diventa quindi una foglia dell'albero decisionale). Se invece alcune variabili che dovrebbero essere intere risultano frazionarie, allora la soluzione ricavata non è ammissibile: se il suo costo è minore dell'attuale miglior soluzione ammissibile, l'algoritmo è costretto ad applicare il branching sul nodo ed a svilupparlo in due sottoproblemi. A ciascuno di questi ultimi verrà riapplicato il ragionamento appena descritto e così via, fino a che non si sarà oltrepassato un certo tempo limite complessivo o non si saranno elaborate tutte le foglie dell'albero.

Il metodo sopra esposto è in grado di risolvere con buone probabilità (in modo esatto ed entro un accettabile tempo computazionale) esempi di piccole dimensioni. Inoltre, risolvendo il rilassamento continuo del modello (ottenuto annullando i vincoli di interezza sulle variabili x, y e z), viene fornito un lower-bound (quindi una stima ottimistica) dell'effettivo costo minimo, cosa che risulta molto utile per valutare la qualità delle migliori soluzioni fornite da metodi euristici.

3. LA TABU SEARCH

3.1 Presentazione algoritmica

La Tabu Search è un metodo euristico (dunque non esatto) utilizzato per risolvere problemi matematici del tipo min f(x) soggetto a $x \in X$.

Il vantaggio legato a questo metodo è che si tratta di un algoritmo del tutto generale, applicabile a qualsiasi problema avente la forma appena descritta, e con l'attraente caratteristica di essere facilmente implementabile, in quanto basato su ragionamenti e considerazioni molto semplici.

Ecco come si procede: si parte da una soluzione iniziale ammissibile $x_o \in X$, e le si applica un algoritmo iterativo di discesa: data x_o, si considera il suo intorno $I(x_o)$ nello spazio soluzione X, dove con $I(x_o)$ si intende quell'insieme di soluzioni ammissibili (appartenenti ad X) vicine, secondo una qualche metrica definita in tale spazio, ad x_o.

A questo punto si sceglie la miglior soluzione x_1 dell'intorno in questione, ossia $x_1 = \arg\min f(x)$ con $x \in I(x_o)$, e si prende x_1 come nuova soluzione corrente (presumibilmente migliorativa rispetto ad x_o). Si ripete il ragionamento su x_1:se ne considera l'intorno $I(x_1)$ e si calcola $x_2 = \arg\min f(x)$ con $x \in I(x_1)$, facendola diventare la nuova soluzione corrente del problema, e così via.

Fin qui non c'è niente di nuovo rispetto ad un normale algoritmo iterativo di discesa: si ottiene una sequenza di soluzioni x_o, x_1, …, x_k, …, ognuna migliorativa rispetto alla precedente, finchè non si arriverà ad una soluzione x_n tale che nessuna soluzione dentro al suo intorno è in grado di migliorarla. Questo significa che si è arrivati ad un ottimo locale, e che in qualsiasi direzione ci si muova a partire da x_n, non si potrà far altro che peggiorare il valore di $f(x_n)$.

I comuni algoritmi di discesa si fermano qui, soddisfatti di aver raggiunto un ottimo locale; la Tabu Search si differenzia da essi proprio in questo punto, in quanto prosegue la ricerca con la speranza che, da qualche altra parte dello spazio soluzione, possa esistere un altro minimo locale di costo inferiore rispetto al primo.

Una volta che si è giunti all'ottimo locale x_n (non più migliorabile in un solo passo), pur di non rimanervi intrappolata la Tabu Search è anche disposta a visitare soluzioni peggiorative, e dunque in questo caso sceglie come nuova soluzione corrente $x_{n+1} = \arg\min f(x)$ con $x \in I(x_n)/\{x_n\}$, che fra tutte le soluzioni appartenenti ad $I(x_n)$ è evidentemente quella che peggiora di meno il costo di x_n.

In pratica si prosegue con la solita tecnica, ma ci si deve rendere conto che, così facendo, nasce il rischio di tornare in soluzioni già visitate e di cadere quindi in pericolosi cicli. Per evitare questa spiacevole situazione non si fa altro che tenere memoria delle soluzioni su cui si è già passati, una memoria cosiddetta "tabu" (da cui anche il nome del metodo): prima di scegliere $x_{i+1} = \arg\min f(x)$ con $x \in I(x_i)$ come nuova soluzione corrente, si

controlla che x_{i+1} non sia presente nella lista tabu; nel caso lo sia, vuol dire che x_{i+1} è già stata visitata in precedenza e di conseguenza deve essere scartata; la nuova candidata a diventare la soluzione corrente sarà $x_{i+1}' = \arg \min f(x)$ con $x \in I(x_i) \setminus \{x_{i+1}\}$, la si sottoporrà al controllo di presenza nella lista tabu e così via, finchè non si troverà una soluzione per cui tale controllo risulterà negativo; si potrà così procedere alla visita di quest'ultima, con la certezza di esserci capitati per la prima volta.

In pratica, per evitare sovraccarichi di memoria si preferisce mantenere nella tabu list soltanto le ultime soluzioni visitate: la speranza è che, dopo aver impedito l'accesso ad una soluzione per un certo numero di passi, l'algoritmo ve ne sia allontanato in misura tale da non rischiare più comunque di ricapitarci. Seguendo questa filosofia, la tabu list potrà così essere implementata molto semplicemente come una coda FIFO di lunghezza fissa, in cui la più vecchia soluzione esplorata presente in memoria (First In) è la prima destinata ad uscirne (First Out).

Addirittura, nel metodo Tabu Search applicato da [2] ad un sistema di rete di interconnessione, si considera troppo oneroso memorizzare anche solo le ultime soluzioni visitate: ciò che si fa è allora tener traccia delle ultime mosse effettuate, intendendo con mossa un insieme di operazioni elementari da compiere per passare da una soluzione ad un'altra.

Tornando all'algoritmo generale, un fatto degno di nota è che può tornare utile nel corso dell'esplorazione, di tanto in tanto, diversificare la ricerca: se dopo un certo numero di passi non si è ancora riusciti a migliorare il costo dell'ultimo minimo locale trovato, significa che tale zona dell'insieme soluzione non è tanto promettente da meritare ulteriore tempo d'indagine, per cui si sceglie una soluzione completamente diversa dall'ultima considerata (fase di "Diversificazione"), e da lì si ricomincia la ricerca iterativa con la tecnica degli intorni (fase di "Intensificazione"), sperando in essere capitati in una regione migliore della precedente.

3.2 Applicazione della Tabu Search alla rete di interconnessione

Verrà ora esposto a grandi linee il modo con cui la Tabu Search è stata applicata in [2] ed adattata al problema della minimizzazione del costo di una rete di interconnessione.

Come prima cosa, una bozza di soluzione iniziale (anche non ammissibile) viene costruita tramite una semplice procedura "greedy", la quale alloca ciascuna BTS al CSS o all'LE che ne minimizza il costo di collegamento, senza tener conto dell'eventuale violazione di vincoli sui limiti di carico dei dispositivi. Fatto questo, viene applicata una procedura di riallocazione per cercare di ridurre il grado di non ammissibilità della soluzione; più precisamente, se viene violato ad esempio il vincolo sul massimo carico di traffico in corrispondenza di un certo CSS o LE, si comincia a riallocare verso

altri dispositivi le BTS ad esso afferenti, a partire da quella con traffico maggiore e via via a scalare, fino a quando il dispositivo in questione non rientrerà nei limiti di traffico previsti. Una simile procedura di riallocazione viene applicata anche a tutti i CSS/LE eccessivamente caricati di moduli ed a tutti i CSS a cui afferiscono troppe BTS.

Una volta sistemati i collegamenti delle BTS coi dispositivi di ordine superiore, si pensa ai collegamenti fra CSS ed LE; anche qui, prima si allocano i CSS agli LE che ne minimizzano i costi di collegamento, dopodichè si pensa ad eventuali riallocazioni di CSS nel caso si superassero in qualche LE i limiti di carico previsti.

Alla fine di questo processo, dopo alcuni passaggi di costruzione ed aggiustamento, si ha tipicamente a disposizione una soluzione iniziale ammissibile del problema, la quale verrà usata come punto di partenza per l'esplorazione dello spazio soluzione.

Durante la ricerca effettuata con la Tabu Search, ogni soluzione viene valutata tenendo in conto il suo reale costo complessivo più eventuali penalità (non lineari) associate ai dispositivi in cui non sono rispettati i relativi vincoli di carico. Più tali penalità sono elevate e più il costo della corrispondente soluzione non ammissibile aumenta, rendendo più improbabile la sua scelta da parte dell'algoritmo. In teoria si dovrebbe fissare il valore delle penalità ad infinito, in modo che non vengano mai prese in considerazione soluzioni non ammissibili, ed in effetti questo è ciò che si fa per la maggior parte della procedura. In alcune fasi dell'esplorazione, per diversificare la ricerca, può però tornare utile abbassare il valore delle penalità ed analizzare zone non ammissibili dello spazio soluzione. Una volta compiuto un certo numero di passi, si considera la miglior soluzione (evidentemente non ammissibile) trovata durante tale fase e la si prende come nuovo punto di partenza a cui applicare l'algoritmo iterativo, stavolta con le penalità fissate ad infinito (al fine di visitare soltanto soluzioni ammissibili).

Per quanto riguarda un singolo passo dell'algoritmo iterativo, per passare da una soluzione alla successiva si sceglie la più conveniente fra 7 mosse possibili, che ora si andranno velocemente ad elencare:

1) Disattivazione di un CSS collocato, scelto fra i 7 meno utilizzati, con conseguente riassegnazione (al minimo costo complessivo) delle BTS ad esso afferenti.

2) Disattivazione di un LE collocato, scelto fra i 3 meno utilizzati, con conseguente riassegnazione (al minimo costo complessivo) delle BTS e dei CSS ad esso afferenti.

3) Attivazione di un nuovo CSS complex, scelto a caso fra 7 non collocati, con conseguente riallocazione di alcune BTS (a partire da quelle che causano un sovraccarico ai dispositivi a cui afferiscono).

4) Attivazione di un nuovo LE, scelto a caso fra 3 non collocati, con conseguente riallocazione di alcune BTS ed alcuni CSS (a partire da quelli che causano un sovraccarico ai dispositivi a cui afferiscono).

5) Cambio del tipo di un CSS (ovviamente collocato), ossia sostituzione di un CSS simple con uno complex (nel caso il CSS sia troppo caricato) o viceversa (nel caso contrario), sostituzione possibilmente seguita da una riallocazione di BTS.

6) Riassegnazione di una BTS attualmente afferente ad uno dei 5 dispositivi più utilizzati della rete.

7) Scambio dei dispositivi di destinazione fra due BTS.

Come già accennato in precedenza, la Tabu Search alterna una “fase di esplorazione”, caratterizzata da basse penalità per la non ammissibilità, ad una “fase di intensificazione”, in cui le penalità sono invece poste a valori molto elevati (teoricamente infiniti). Qualora, durante la fase di intensificazione, dopo 20 mosse non si sia riusciti a migliorare il costo dell’ultimo ottimo locale incontrato, non ritenendo più appetibile la zona dello spazio soluzione in cui ci si trova, si effettua una “diversificazione”, che consiste nella costruzione di una rete di interconnessione la più diversa possibile da quella attuale: si scambiano i CSS e gli LE attivi con quelli disattivi e si riallocano alcune BTS in maniera del tutto simile a quanto fatto durante la procedura di inizializzazione. A questo punto, una volta ottenuta la rete per così dire “complementare”, comincia la fase di esplorazione che dura un numero predefinito di mosse e, visto il basso valore delle penalità, consente l’attraversamento anche di soluzioni non ammissibili. Alla fine di tale fase si prende il miglior ottimo locale da essa trovato e lo si usa come punto di partenza per una nuova fase di intensificazione (penalità ridefinite a valori molto elevati).

L’intero processo termina quando viene raggiunto un determinato numero massimo di mosse o un predefinito tempo limite di esecuzione. La soluzione finale è così il miglior minimo locale ammissibile incontrato durante tutta la ricerca.

3.3 Esempi di test e risultati

E' stato applicato il test bed di input del Par. 1.6 alla Tabu Search trattata in [2] ed illustrata nel paragrafo precedente; qui di seguito si espongono i risultati ottenuti.

Input : test bed descritto al Par. 1.6

Metodo : Tabu Search

Tipo hardware : Pentium II 450 MHz, 128 MB RAM

Sistema Operativo : Windows 98

Parametri di ricerca di maggior rilevanza

Tempo limite = 6^h ; 8^h ; 10^h ; 12^h rispettivamente per le classi A, B, C, D

Costo penalità in esplorazione : 2.500

Costo penalità in intensificazione : 500.000

Classi	Esempi	Dimensioni			Costo miglior soluzione ammissibile
		BTS	CSS	LE	
1	A	50	4	2	10.760.529
	B	55	5	3	11.619.817
	C	60	5	4	12.395.078
	D	65	6	4	13.445.309
2	E	100	8	4	20.006.714
	F	105	9	5	22.151.922
	G	110	9	6	23.100.774
	H	115	10	6	23.999.151
3	I	150	12	6	30.139.211
	L	155	13	7	31.152.784
	M	160	13	8	32.070.411
	N	165	14	8	32.896.514
4	O	200	20	10	39.241.845

Tabella 3.3.1 : costo delle migliori soluzioni ammissibili fornite dalla Tabu Search in corrispondenza del test bed

Visto che questo è il primo metodo analizzato, i valori riportati nella tabella sono per ora puramente indicativi (acquisteranno significato solo dopo che si saranno confrontati con i valori ottenuti dai prossimi metodi), ma forniscono comunque lo spunto per alcune osservazioni.

Si nota innanzitutto che, all'aumentare delle dimensioni della rete in input, il costo della miglior soluzione fornita dalla Tabu Search aumenta; tale aumento è graduale nel passaggio da un esempio ad un altro all'interno di una stessa classe, mentre è invece molto brusco nel passaggio da una classe ad un'altra (come d'altronde è nei due casi l'aumento del numero di dispositivi). Si era osservato al Par. 1.6 che un incremento del numero di BTS causa un innalzamento del costo della rete, mentre effetto contrario provoca un incremento del numero di potenziali CSS/LE, e ci si chiedeva quale potesse essere il risultato della combinazione dei due effetti sull'andamento complessivo del costo, qualora cioè sia BTS che potenziali CSS/LE crescessero contemporaneamente in numero (si tenga comunque presente che fondamentale importanza in questo discorso rivestono i tassi di crescita dei tre tipi di apparati). Ebbene, dai risultati della tabella si evince che, ai tassi di crescita di BTS, CSS ed LE dati dal test bed, l'incremento di costo per l'aumento di BTS prevale sulla sua diminuzione per l'aumento di potenziali CSS/LE (il costo complessivo della rete, infatti, aumenta).

Si espone ora la situazione di carico dei dispositivi in corrispondenza delle soluzioni ottime (per ragioni di spazio si presentano solo i casi relativi agli esempi di input della classe 3).

Esempio I (150 BTS, 12 CSS, 6 LE)

CSS	Traffico (Erl)	Moduli entranti	n° BTS
1 Complex	599	23	23
2 Complex	598	23	23
4 Complex	592	24	24
6 Complex	591	25	25
9 Complex	600	21	21

LE	Traffico (Erl)	Moduli entranti
1	2.285	36
4	1.871	33

Esempio L (155 BTS, 13 CSS, 7 LE)

CSS	Traffico (Erl)	Moduli entranti	n° BTS
4 Complex	587	26	26
5 Complex	596	24	24
7 Complex	596	23	23
9 Complex	599	21	21
10 Complex	598	22	22

LE	Traffico (Erl)	Moduli entranti
2	2.741	48
3	1.584	26

Tabelle 3.3.2 : situazioni di carico dei dispositivi nelle soluzioni ottime relative alla classe 3 (esempi I, L) del test bed; nelle tabelle sono presenti soltanto i dati relativi ai dispositivi collocati

Esempio M (160 BTS, 13 CSS, 8 LE)

CSS	Traffico (Erl)	Moduli entranti	n° BTS
0 Complex	596	20	20
4 Complex	600	26	26
5 Complex	600	22	22
6 Complex	598	27	27
10 Complex	598	22	22

LE	Traffico (Erl)	Moduli entranti
2	1.825	33
7	2.676	47

Esempio N (165 BTS, 14 CSS, 8 LE)

CSS	Traffico (Erl)	Moduli entranti	n° BTS
0 Complex	574	27	27
1 Complex	599	21	21
2 Complex	600	23	23
7 Complex	587	26	26
10 Complex	599	17	17
12 Complex	597	21	21

LE	Traffico (Erl)	Moduli entranti
0	2.190	34
5	2.344	38

Tabelle 3.3.3 : situazioni di carico dei dispositivi nelle soluzioni ottime relative alla classe 3 (esempi M, N) del test bed; nelle tabelle sono presenti soltanto i dati relativi ai dispositivi collocati

Ciò che si osserva è che la Tabu Search tende sempre a costruire delle reti bilanciate ed a sfruttare al massimo tutti i dispositivi collocati, caricandoli fin quasi al limite delle loro capacità (i valori delle massime capacità di carico sono riportati in tabella 1.4.1 a pag. 14). Questo è del resto abbastanza comprensibile: visto che i costi di attivazione dei CSS e degli LE sono molto elevati è bene che, una volta deciso di attivarne uno, lo si usi al 100% delle sue potenzialità. Una rete sbilanciata, ossia con dispositivi collocati utilizzati solo in piccola parte rispetto alle loro possibilità, è sicuramente una rete poco vantaggiosa dal punto di vista economico.

Si vuole fornire anche un'immagine visiva di quella che può essere la struttura finale di una rete di interconnessione, una volta stabiliti i dispositivi da attivare ed i collegamenti con cui connetterli. Per ragioni di chiarezza grafica si è scelta la visualizzazione della rete ottima corrispondente ad un esempio di dimensioni contenute, ossia l'esempio A:

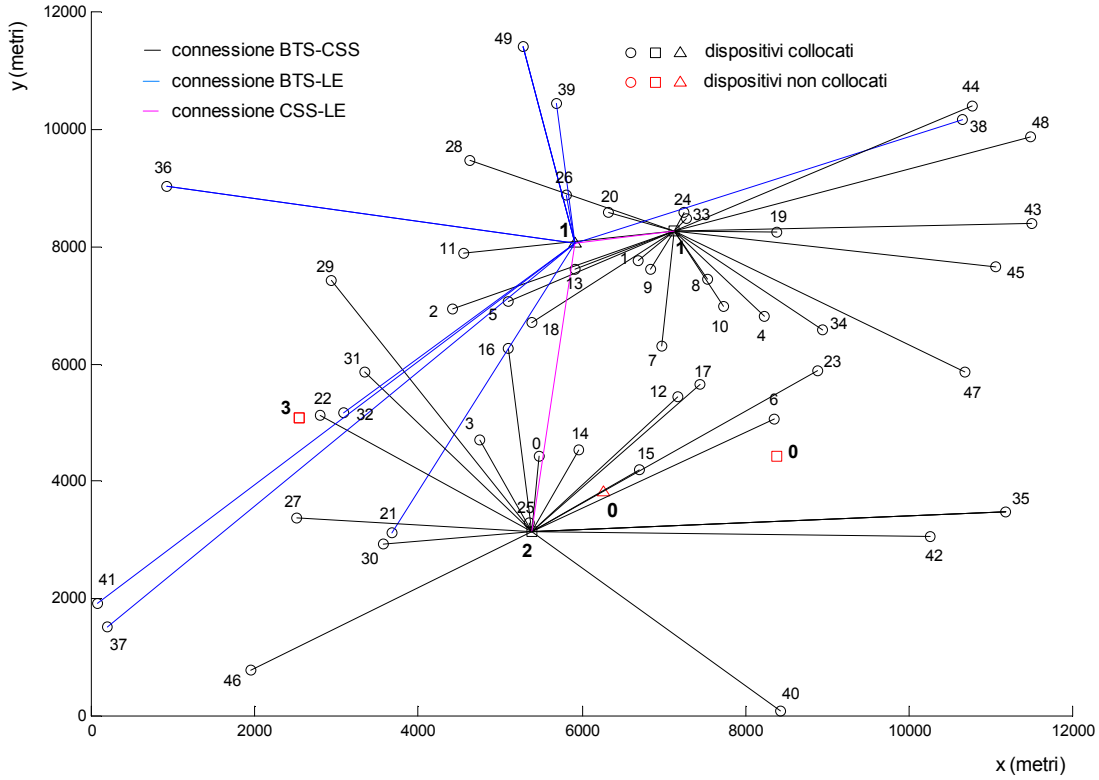

Figura 3.3.1 : grafico corrispondente alla rete di interconnessione ottima relativa all'esempio A (metodo Tabu Search)

La figura sottolinea come la presunta tendenza da parte di ciascun dispositivo a connettersi al suo "superiore" più vicino sia in realtà spesso non verificata (anche perché altrimenti la soluzione del problema diventerebbe a dir poco banale: sarebbe infatti sufficiente l'algoritmo greedy che ricava, tramite le suddette considerazioni di minima distanza, la soluzione iniziale per la Tabu Search, mentre dalle simulazioni è emerso quanto tale soluzione possa essere migliorabile).

I motivi per cui l'algoritmo può scegliere di non assegnare un dispositivo al suo superiore più vicino sono essenzialmente tre:

1) incompatibilità di tipo: non si può assegnare una BTS isolated ad un CSS, o una BTS basic ad un LE;
2) sottosfruttamento del dispositivo superiore: non è conveniente connettere ad esempio una BTS al CSS più vicino qualora quest'ultimo fosse poi destinato a servire solo quella BTS e poche altre (i costi di attivazione sono molto elevati);

3) sovrasfruttamento del dispositivo superiore: è proibito connettere un dispositivo al suo superiore più vicino qualora quest'ultimo, con tale connessione, superasse i limiti di carico previsti.

Uno svantaggio del metodo Tabu Search appena analizzato può essere la scelta del valore da assegnare al costo CP di una singola penalità durante la fase definita "di esplorazione", ossia quella fase parzialmente rilassata dai vincoli di carico in cui è permessa all'algoritmo anche la visita di soluzioni non ammissibili. Cambiare il valore di CP ha infatti a volte ripercussioni non indifferenti sulla soluzione ottima e sul suo costo, e purtroppo tale dipendenza non segue regole analitiche. Abbassare ad esempio il valore di CP significa permettere all'algoritmo di esplorare, nelle fasi in cui è consentito, soluzioni con un maggior grado di non ammissibilità (cioè soluzioni in cui i vincoli di carico sono più palesemente violati); ciò ha come effetto quello di far scegliere all'algoritmo semplicemente un'altra strada, ed il fatto che quest'ultima conduca ad una soluzione finale migliore o peggiore della precedente è del tutto imprevedibile. Nel corso delle simulazioni svolte nel presente lavoro si è sempre posto CP = 2.500 (per poi tramutarlo in 500.000 durante le fasi di intensificazione), in quanto si è sperimentalmente osservato che tale valore conduce mediamente ad una migliore soluzione ammissibile finale rispetto ad altre possibili strategie (anche di tipo oscillatorio).

4. IL PROGRAMMA ILOG - CPLEX

4.1 Presentazione e caratteristiche di funzionamento

Il software commerciale (ILOG-) Cplex è uno strumento dedicato alla risoluzione di problemi di programmazione lineare (intera e non) della forma:

$$\begin{array}{ll} \text{max / min} & c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \\ \\ \text{subject to} & a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \sim b_1 \\ & a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \sim b_2 \\ & \qquad \ldots\ldots\ldots \\ & a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \sim b_m \\ \\ \text{bounds} & l_1 \le x_1 \le u_1 \\ & l_2 \le x_2 \le u_2 \\ & \quad \ldots\ldots \\ & l_n \le x_n \le u_n \end{array}$$

dove $\sim$ può essere $\le$, $\ge$, o $=$, ed i limiti superiori u_i ed inferiori l_i possono essere positivi infiniti, negativi infiniti od un qualsiasi numero reale.

I dati forniti come input a Cplex sono i coefficienti della funzione obiettivo (c_1, …, c_n), i coefficienti dei vincoli (a_{11}, …, a_{mn}), i termini noti a secondo membro dei vincoli (b_1,…, b_m) e i limiti superiori (u_1,…, u_n) e inferiori (l_1,…, l_n) delle variabili.

Come output Cplex restituisce i valori delle variabili in corrispondenza di una soluzione ottima $x^* = (x_1^*,\ldots, x_n^*)$ ed il valore della funzione obiettivo $f(x^*)$.

Cplex può anche risolvere:

- Problemi di Mixed Integer Programming (MIP), dove alcune o tutte le variabili sono sottoposte all'ulteriore vincolo di assumere solo valori interi.

- Problemi di Quadratic Programming (QP), dove la funzione obiettivo può anche contenere termini quadratici.

- Problemi cosiddetti Network Flow, un caso speciale di LP che Cplex può risolvere molto più velocemente sfruttando la struttura del problema.

Cplex può essere utilizzato in tre forme differenti:

- Il Cplex Interactive Optimizer è un programma eseguibile che legge un problema in modo interattivo o da file (in certi formati standard), risolve il problema e mostra la soluzione in modo interattivo o in file di testo.
- La Concert Tecnology è un insieme di librerie che permettono al programmatore di includere gli ottimizzatori Cplex in un'applicazione C++.
- La Cplex Callable Library è una libreria C che permette al programmatore di includere gli ottimizzatori Cplex in applicazioni scritte in C, Visual Basic, Java e Fortran.

Nel caso di questa tesi Cplex sarà utilizzato in ambito Concert Tecnology (verrà infatti inserito all'interno di programmi C++) per risolvere problemi di tipo MIP.

4.2 I parametri di settaggio fondamentali

Cplex offre un metodo esatto per la soluzione del problema MIP che gli viene sottoposto e, se si avesse il tempo di aspettare il completamento dell'esecuzione dell'algoritmo, non ci sarebbe bisogno di definire od introdurre alcun parametro.

Purtroppo però il MIP è un problema NP-hard, e per istanze di grandi dimensioni il tempo di calcolo della soluzione ottima da parte di Cplex è spesso inaccettabile. Ecco allora che nasce la necessità di interrompere Cplex prima della sua naturale conclusione, ed ecco che in tal caso diventa conveniente considerare alcuni parametri che hanno il compito di gestire al meglio questa "ricerca abbreviata".

Si presenteranno di seguito soltanto i parametri che sono stati settati all'interno dei programmi C++ utilizzati per la tesi (inutile dire che l'insieme completo dei parametri che Cplex mette a disposizione dell'utente è molto più vasto):

- TiLim: setta il tempo massimo di esecuzione, trascorso il quale l'algoritmo termina pur senza garanzia di ottimalità.
- NodeLim: setta il massimo numero di nodi da esplorare prima che l'algoritmo termini (pur senza aver raggiunto l'ottimalità).
- TreLim: setta la massima quantità di memoria (in MB) utilizzabile dall'algoritmo branch and cut per la costruzione dell'albero.

- NodeFileInd: è un parametro il cui valore di settaggio decide come procedere una volta raggiunta la massima quantità di memoria utilizzabile stabilita da TreLim: se è settato a 0 l'algoritmo termina, altrimenti si prosegue memorizzando alcuni nodi in alcuni file di salvataggio.

- MIPEmphasis: è un parametro binario; se è settato a 0 dà la precedenza all'ottimalità piuttosto che all'ammissibilità (ossia cerca di trovare il prima possibile la soluzione ottima, senza preoccuparsi di trovare buone soluzioni ammissibili intermedie); se invece è settato a 1 enfatizza l'ammissibilità rispetto all'ottimalità (il che vuol dire che, nella ricerca dell'ottimo, tenta di trovare più soluzioni ammissibili intermedie possibile).

- CutUp: è un parametro di cutoff per i nodi di costo superiore, nel senso che Cplex ignora tutti i nodi aventi un valore della funzione obiettivo superiore al valore di CutUp; naturalmente l'inserimento ed il settaggio di un simile parametro ha significato soltanto se si ha a che fare con un problema di minimizzazione (come in questo caso). Se durante l'esecuzione Cplex trova una soluzione intera di valore inferiore a quello inserito manualmente dall'utente, è chiaro che per il prosieguo dell'esplorazione verrà considerato un nuovo valore di cutoff, pari al costo di tale soluzione.

- IntSolLim: setta il numero di soluzioni intere da trovare prima di fermare l'esecuzione.

- EpAGap: setta una tolleranza assoluta sul gap tra il costo della miglior soluzione intera a disposizione e quello del miglior nodo rimanente (intendendo con nodi rimanenti quelli non ancora ramificati). Quando il modulo di tale gap cade al di sotto del valore assegnato ad EpAGap, il processo di ottimizzazione viene interrotto.

- EpGap: setta una tolleranza relativa sul gap tra il costo della miglior soluzione intera a disposizione e quello del miglior nodo rimanente. Quando il modulo relativo di tale gap cade al di sotto del valore assegnato ad EpGap, il processo di ottimizzazione viene interrotto.

4.3 Esempi di test e risultati

Si è applicato il programma Cplex (vers. 7.0) al modello generale (0)–(16) esposto al Par. 2.2, utilizzando come input il test bed del Par. 1.6.

Per ogni esempio del test bed è stata effettuata una doppia elaborazione, una con il parametro indicatore MIPEmphasis fissato a 0 e l'altra con lo stesso parametro fissato a 1, al fine di verificare quanto il settaggio del suddetto indicatore possa influire sulle prestazioni del programma in esame.

Si riportano qui di seguito i risultati ottenuti (confrontati con quelli ricavati in precedenza dalla Tabu Search).

Input : test bed descritto nel Par. 1.6

Metodo : Cplex 7.0

Tipo hardware : Pentium II 450 MHz, 128 MB RAM

Sistema Operativo : Windows 98

Parametri di ricerca di maggior rilevanza

Tempo limite = 6^h ; 8^h ; 10^h ; 12^h rispettivamente per le classi A, B, C, D

Indicatore dell'enfasi (MIPEmphasis) = 0 / 1 per ogni esempio di test

Tolleranza assoluta (EpAGap) = 10^{-9}

Tolleranza relativa (EpGap) = 10^{-9}

Memoria limite (TreLim) = 128 MB

Indicatore di gestione memoria in sovraccarico (NodeFileInd) = 1

Classi	Esempi	Costo miglior soluzione ammissibile		
		Cplex		Tabu Search
		MIPEmph = 0	MIPEmph = 1	
1	A (50, 4, 2)	10.761.484	**10.760.529**	**10.760.529**
	B (55, 5, 3)	**11.619.817**	**11.619.817**	**11.619.817**
	C (60, 5, 4)	12.400.694	12.396.436	**12.395.078**
	D (65, 6, 4)	**13.376.659**	**13.376.659**	13.445.309
2	E (100, 8, 4)	**19.734.790**	**19.734.790**	20.006.714
	F (105, 9, 5)	21.956.718	**21.952.231**	22.151.922
	G (110, 9, 6)	22.975.773	**22.971.543**	23.100.774
	H (115, 10, 6)	24.007.215	**23.996.482**	23.999.151
3	I (150, 12, 6)	30.286.351	30.142.363	**30.139.211**
	L (155, 13, 7)	31.560.692	**31.098.048**	31.152.784
	M (160, 13, 8)	32.460.110	**32.045.632**	32.070.411
	N (165, 14, 8)	33.218.924	**32.866.855**	32.896.514
4	O (200, 20, 10)	41.452.951	39.282.420	**39.241.845**

Tabella 4.3.1 : costo delle migliori soluzioni ammissibili fornite da Cplex con MIPEmphasis = 0 / 1 e confronto con Tabu Search (il costo minimo di ogni esempio è evidenziato in grassetto)

Per quanto riguarda il confronto fra le due serie di simulazioni effettuate con Cplex (distinte fra loro, come si è già detto, dal valore associato al parametro MIPEmphasis), si nota che per le prime due classi di esempi non vi sono grosse disparità di comportamento (i costi finali o sono uguali o differiscono solo di qualche migliaio a favore del caso MIPEmphasis = 1).

Per le classi 3 e 4 risulta invece chiaro come i risultati ottenuti con MIPEmphasis = 1 siano di gran lunga migliori di quelli ricavati con MIPEmphasis = 0 (con riduzioni di costo dell'ordine di centinaia di migliaia).

Il motivo per cui, negli esempi di input di grosse dimensioni, Cplex offra delle soluzioni finali così poco appetibili qualora MIPEmphasis sia settato a 0 risiede nel fatto che, in tal caso, la priorità del programma è rivolta non tanto a trovare una sequenza di soluzioni intermedie ammissibili via via migliorative, quanto piuttosto ad arrivare alla soluzione ottima nel più breve tempo possibile. Ora, poiché gli esempi delle classi 3 e 4 sono di dimensioni tali da non poter sperare di raggiungere l'ottimo in tempi ragionevoli, Cplex si deve accontentare della migliore fra le poche soluzioni ammissibili incontrate durante l'esplorazione. Nelle prove è accaduto sovente che, dopo aver trovato nei primi minuti di simulazione una soluzione ammissibile di scarsa qualità, nel corso delle ore successive Cplex non sia più riuscito a migliorarla.

Al contrario, in ciascuna delle prove contraddistinte da MIPEmphasis = 1 viene fornita a Cplex la possibilità di visitare, durante l'esplorazione del branching tree, un numero molto più elevato di soluzioni ammissibili, ed è quindi verosimile che l'ottima fra queste sia migliore della corrispondente ottenuta con MIPEmphasis = 0.

D'ora in avanti, quindi, quando ci si riferirà alle simulazioni effettuate con Cplex, si intenderà implicitamente la serie di simulazioni che, fra le due svolte, offre i risultati migliori, ossia quella con il parametro MIPEmphasis settato a 1.

Per quanto riguarda il confronto fra la serie di simulazioni effettuata con Cplex (caso MIPEmphasis = 1) e quella svolta con la Tabu Search, si osserva come nella maggior parte degli esempi Cplex offra i risultati migliori (talvolta in modo molto significativo, come negli esempi E, F e G). Non si può però legare il miglioramento delle prestazioni del primo metodo rispetto al secondo all'aumento delle dimensioni degli esempi in ingresso; non si può cioè affermare che il divario di comportamento fra Cplex e la Tabu Search si faccia più accentuato via via che si considerano reti di dimensioni crescenti: all'interno di ogni classe si ha infatti almeno un esempio in cui la Tabu Search si avvicina o addirittura migliora (come negli esempi I ed O) i risultati forniti da Cplex.

Nei prossimi capitoli si studierà il progetto di metodi di esplorazione alternativi, i quali utilizzeranno Cplex in maniera più oculata di quanto visto finora: essi non lo applicheranno infatti direttamente al modello generale, ma bensì a modelli (derivati da quest'ultimo) più restrittivi. In altre parole, invece di studiare tutto l'insieme delle soluzioni in un colpo solo, si proverà a studiarlo "per intorni". La filosofia è quindi la stessa su cui è basata la Tabu Search, ma con la grossa differenza che ora ciascun intorno verrà espresso non come studio della migliore tra alcune mosse specifiche, ma bensì come insieme di vincoli da analizzare tramite Cplex.

5. METODO DI RICERCA VARIABLE NEIGHBOURHOOD SEARCH (VNS)

5.1 Introduzione

Come già accennato, i metodi di ricerca locale per l'ottimizzazione combinatoriale procedono effettuando delle esplorazioni locali successive a partire da una soluzione iniziale, esplorazioni che conducono ad una sequenza di soluzioni migliorative della funzione obiettivo, fino a che non viene incontrato un ottimo locale. In pratica, ad ogni iterazione viene ottenuta una soluzione migliorativa x' nell'intorno N(x) della soluzione corrente x, ed il processo si ripete finché non si arriverà ad un'iterazione in cui nel relativo intorno non è più possibile un decremento del costo (e ci si troverà per l'appunto in un ottimo locale).

Negli anni recenti sono state proposte numerose tecniche metaeuristiche che estendono in varie direzioni questo schema, ed evitano di restare intrappolate in ottimi locali aventi un insoddisfacente valore della funzione obiettivo. Queste metaeuristiche (tra cui si possono citare Tabu Search, Simulated Annealing, Genetic Search, GRASP, Variable Depth Search, Adaptative Multi-Start) sono riuscite ad ottenere risultati sorprendenti in molti contesti pratici.

Una tecnica relativamente nuova e quindi non ancora esplorata a fondo è la cosiddetta VNS (Variable Neighbourhood Search), introdotta e discussa in [3]. Contrariamente ad altre metaeuristiche basate su metodi di ricerca locale, la VNS non segue una traiettoria ma esplora intorni a distanza via via crescente dall'attuale miglior soluzione ammissibile, e si muove da quest'ultima ad una nuova se e solo se si è evidenziato un miglioramento della funzione obiettivo.

5.2 Algoritmo generale VNS

Si denoti con N_k un insieme finito di strutture di intorni pre-selezionate ($k = 1, \ldots, k_{max}$), e con $N_k(x)$ l'insieme di soluzioni nel k-esimo intorno di x (le euristiche classiche di ricerca locale solitamente usano una sola struttura di intorno, ossia pongono $k_{max} = 1$).

L'euristica base VNS comprende i seguenti passi:

Inizializzazione:

seleziona l'insieme di strutture di intorni N_k , k = 1,…,K_{max}, che sarà usato nella ricerca; trova una soluzione iniziale x; scegli una condizione di termine esplorazione.

Ripeti i seguenti passi finchè non si è incontrata la condizione di termine:

(1) fissa k := 1

(2) finchè $k \leq k_{max}$, ripeti i seguenti passi:

(a) diversificazione (shaking): genera casualmente un punto x' appartenente al k-esimo intorno di x (x' $\in$ $N_k(x)$);

(b) intensificazione (local research): applica un qualche metodo di ricerca locale con x' come soluzione iniziale; denota con x'' l'ottimo locale così ottenuto

(c) spostamento eventuale: se l'ottimo locale appena trovato è migliore di quello incombente x, considera x'' come nuovo ottimo incombente (x := x'') e prosegui la ricerca con N_1 (k := 1); altrimenti, fissa k := k+1

(3) anche se x'' è peggiore di x, poiché si è nella situazione in cui $k > k_{max}$, assegna comunque x := x'' e da lì prosegui la ricerca

Un'interpretazione grafica di tale metodo, applicato ad una funzione di variabile scalare, può essere la seguente:

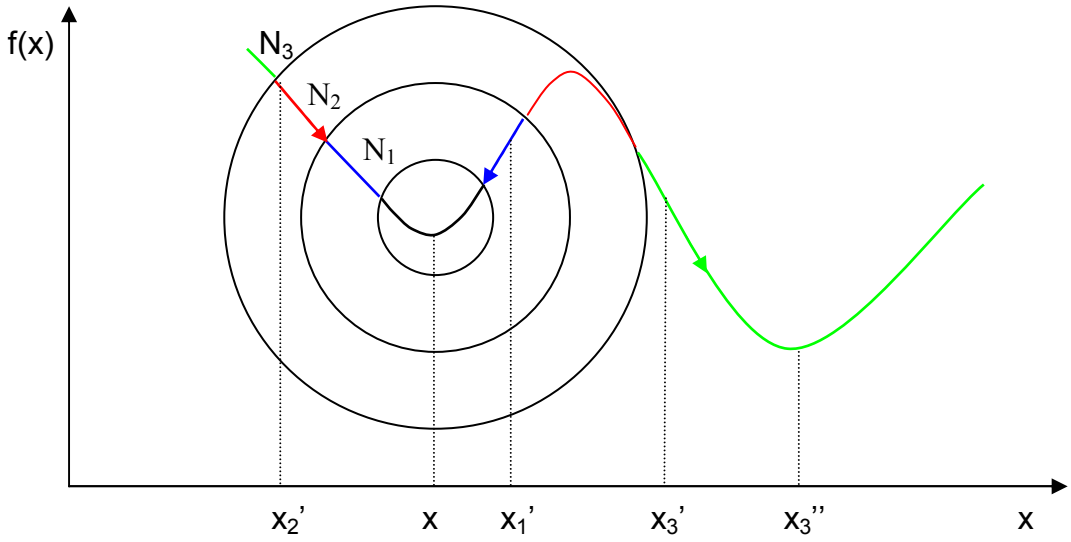

Figura 5.2.1 : Interpretazione grafica metodo VNS

Si parte dall'ottimo locale x: si considera l'intorno N_1 e si sceglie una soluzione a caso x_1' al suo interno, dopodichè si applica l'algoritmo iterativo di discesa; ciò che succede è che si torna in x, ossia $x_1'' \equiv x$.

Si allontana allora la regione di indagine e si passa a considerare l'intorno N_2, di cui si sceglie una soluzione a caso x_2'; anche in questo caso, applicando a x_2' l'algoritmo di discesa, si torna in x, ossia $x_2'' \equiv x$.

Si ripete ancora una volta il ragionamento: si sceglie una soluzione a caso x_3' appartenente all'intorno N_3, ancor più distante da x rispetto ai precedenti; stavolta, applicando l'algoritmo di discesa a x_3', si arriva ad un ottimo locale $x_3'' \neq x$, e pure migliore.

Ciò che si fa a questo punto è di prendere x_3'' come nuova soluzione incombente (cioè come nuova x) e rieseguire gli stessi passi visti prima.

La condizione di termine dell'esecuzione può essere un tempo limite, un numero massimo di iterazioni, oppure un numero massimo di iterazioni tra due miglioramenti.

5.3 Estensioni del VNS

Un'estensione nota del VNS è il VND (Variable Neighbourhood Descent), in cui viene applicata la tecnica degli intorni variabili anche alla fase di ricerca locale. Il metodo si basa sulla considerazione che un ottimo locale all'interno di un intorno non è necessariamente un ottimo locale per un altro intorno; il VNS, durante la ricerca locale, applica un unico tipo di intorno N_1'(x) alla soluzione corrente x; se dentro ad N_1'(x) non si trova una soluzione migliorativa, si dichiara che x è un ottimo locale e si interrompe l'algoritmo iterativo di discesa. Nel VND, invece, anche se in N_1'(x) non ci sono soluzioni migliori di x, non si interrompe la ricerca locale ma si considera un altro intorno N_2'(x), e se necessario un altro N_3'(x), fino a che non si trova una soluzione che migliora x o non si raggiunge il numero massimo di intorni; in questo secondo caso è allora dichiarata la fine della ricerca locale, ed il suo risultato è per l'appunto l'ottimo locale x.

I passi del VND riguardanti la fase di ricerca locale (detta anche “fase di intensificazione”) sono i seguenti:

Inizializzazione:

seleziona l’insieme di strutture di intorni N_k’, k = 1,…,K_{max}’, che sarà usato nella ricerca locale; considera come soluzione iniziale la soluzione casuale x appartenente all’intorno $N_i(x^*)$ (con x* soluzione incombente), fornita dalla fase di diversificazione.

Algoritmo iterativo:

(1) fissa k := 1

(2) mentre $k \leq k_{max}$’, ripeti i seguenti passi:

(a) esplorazione dell’intorno: trova la miglior soluzione $x' \in N_k'(x)$;

(b) spostamento locale eventuale: se la soluzione così ottenuta x’ è migliore di x, ponila come nuovo ottimo locale provvisorio (x:= x’) e da lì prosegui la ricerca locale con N_1’ (k := 1); altrimenti, fissa k := k+1.

(3) x è l’ottimo locale.

Altra estensione del VNS è il RVNS (Reduced Variable Neighbourhood Search), che si contraddistingue dalla versione base nei seguenti punti:

- quando durante la diversificazione c’è da scegliere la soluzione casuale x’ nell’intorno $N_k(x)$ della soluzione incombente x (o di un ottimo locale che ne ha preso le veci, secondo il passo 3 dell’algoritmo VNS), il RVNS procede a tale scelta prendendo x’ come la miglior soluzione tra l scelte a caso (dove l è un parametro dell’algoritmo) all’interno di $N_k(x)$. La scelta di x’ ora non è più effettuata totalmente “alla cieca”: in qualche modo, si vuole infatti mettere l’algoritmo sulla buona strada, evitandogli il rischio di far partire l’intensificazione da una soluzione di costo eccessivo e di compiere così una ricerca locale lunga o, ancor peggio, dalla prematura convergenza.
- in entrambe le fasi di ricerca si introducono due parametri k_{min} e k_{step}, aventi la funzione di controllare il processo del cambio di intorno: in pratica, si sostituisce all’istruzione di assegnazione k:= 1 l’istruzione k:= k_{min} e all’istruzione d’incremento k:= k+1 la k:= k+k_{step}. Con l’introduzione di tali due parametri, l’implementazione delle fasi di intensificazione e di diversificazione può avvenire in maniera semplice e naturale: infatti, se k_{min} e k_{step} sono settati a dei valori elevati, la ricerca viene effettuata in regioni dello spazio soluzione lontane da quelle esplorate finora (fase di diversificazione), mentre se essi sono assegnati a dei valori contenuti, la ricerca si concentra nelle zone limitrofe (fase di intensificazione).

5.4 Applicazione del VNS e delle sue estensioni ad una rete di interconnessione

In questo paragrafo verrà illustrato il modo in cui l'algoritmo VNS (completo di entrambe le sue estensioni VND ed RVNS) è stato applicato al problema della minimizzazione del costo di una rete di interconnessione.

5.4.1 Espressioni e vincoli caratteristici del metodo

L'espressione (lineare nelle variabili decisionali del nostro modello) che introduce una metrica nello spazio delle soluzioni (e dà quindi significato matematico a concetti quali "distanza" o "lontananza" da una soluzione ammissibile (x*, y*) assegnata) è la seguente:

$$\Delta(x,y,x^*,y^*) := \sum_{j\in Y^*_{CSS}}\left[1-\sum_{h=1,2}\left(y_j^{CSS-h^*}\, y_j^{CSS-h}\right)\right]+\sum_{k\in Y^*_{LE}}\left(1-y_k^{LE}\right)+\sum_{j\notin Y^*_{CSS}}\sum_{h=1,2}y_j^{CSS-h}+$$

$$+\sum_{k\notin Y^*_{LE}}y_k^{LE}+\sum_{(i,j)\in X^*_{BTS-CSS}}\left(1-x_{ij}^{BTS-CSS}\right)+\sum_{(i,j)\notin X^*_{BTS-CSS}}x_{ij}^{BTS-CSS}+\sum_{(i,k)\in X^*_{BTS-LE}}\left(1-x_{ik}^{BTS-LE}\right)+$$

$$+\sum_{(i,k)\notin X^*_{BTS-LE}}x_{ik}^{BTS-LE}+\sum_{(j,k)\in X^*_{CSS-LE}}\left(1-x_{jk}^{CSS-LE}\right)+\sum_{(j,k)\notin X^*_{CSS-LE}}x_{jk}^{CSS-LE} \qquad (5.4.1.1)$$

dove l'asterisco si riferisce ad una particolare soluzione ammissibile fissata, e precisamente:

- $y_j^{CSS-h^*}$ è il valore di y_j^{CSS-h} in corrispondenza della soluzione x*, ed esprime il fatto se tale soluzione possiede o meno nella locazione j un CSS collocato di tipo h
- Y^*_{CSS} = { j ∈ {1,…,m} : $y_j^{CSS-1^*}$ = 1 o $y_j^{CSS-2^*}$ = 1 }, ossia è l'insieme di CSS collocati (con un dispositivo simple o complex, non ha importanza quale) nella soluzione x*
- Y^*_{LE} = { k ∈ {1,…,p}:$y_k^{LE^*}$ = 1 }, ossia è l'insieme di LE collocati nella soluzione x*
- $X^*_{BTS-CSS}$ = { (i,j): i ∈ {1,…,n}, j ∈ {1,…,m} e $x_{ij}^{BTS-CSS^*}$ = 1 }, ossia è l'insieme dei collegamenti attivi fra BTS e CSS nella soluzione x*
- X^*_{BTS-LE} = { (i,k): i ∈ {1,…,n}, k ∈ {1,…,p} e $x_{ik}^{BTS-LE^*}$ = 1 }, ossia è l'insieme dei collegamenti attivi fra BTS ed LE nella soluzione x*

- X^*_{CSS-LE} = { (j,k): j ∈ {1,…,m}, k ∈ {1,…,p} e $x_{jk}^{CSS-LE^*}$ = 1 },ossia è l'insieme dei collegamenti attivi fra CSS ed LE nella soluzione x*

Per quanto riguarda le variabili y_j^{CSS-h}, y_k^{LE}, $x_{ij}^{BTS-CSS}$, x_{ik}^{BTS-LE}, x_{jk}^{CSS-LE} ed il loro significato (peraltro abbastanza intuitivo), si rimanda al Par. 2.1.

Si andrà ora ad esaminare singolarmente i vari termini dell'espressione (5.4.1.1):

1. $$\sum_{j\in Y^*_{CSS}}\left[1-\sum_{h=1,2}\left(y_j^{CSS-h^*}\,y_j^{CSS-h}\right)\right] \equiv \sum_{j\in Y^*_{CSS}}\left(1-y_j^{CSS-1^*}\,y_j^{CSS-1}-y_j^{CSS-2^*}\,y_j^{CSS-2}\right) \qquad (5.4.1.2)$$

Si consideri un generico j ∈ Y_{CSS}*, ossia un generico CSS j che nella soluzione x* risulta collocato con un qualche tipo (1 o 2).

Per le variabili y_j^{CSS-1} e y_j^{CSS-2} si presentano le seguenti possibilità:

- $y_j^{CSS-1} = y_j^{CSS-1^*}$, $y_j^{CSS-2} = y_j^{CSS-2^*}$
  il che significa che nella nuova soluzione il CSS j risulta collocato e con lo stesso tipo rispetto a quello della soluzione x*.
  In questo caso si ha: $1-y_j^{CSS-1^*}\,y_j^{CSS-1}-y_j^{CSS-2^*}\,y_j^{CSS-2}=0$
- $y_j^{CSS-1} = -y_j^{CSS-1^*}+1$, $y_j^{CSS-2} = -y_j^{CSS-2^*}+1$
  il che significa che nella nuova soluzione il CSS j risulta collocato ma con tipo diverso che nella soluzione x*.
  In questo caso si ha: $1-y_j^{CSS-1^*}\,y_j^{CSS-1}-y_j^{CSS-2^*}\,y_j^{CSS-2}=1-0-0=1$
- $y_j^{CSS-1} = 0$, $y_j^{CSS-2} = 0$
  il che significa che nella nuova soluzione il CSS j non è collocato.
  In questo caso si ha: $1-y_j^{CSS-1^*}\,y_j^{CSS-1}-y_j^{CSS-2^*}\,y_j^{CSS-2}=1-0-0=1$

Il termine j-esimo dell'espressione (5.4.1.2) è uguale a 0 se e solo se nella nuova soluzione il CSS j-esimo è collocato e con lo stesso tipo rispetto a quello della soluzione x*.

Di conseguenza, l'espressione (5.4.1.2) conta quanti CSS della nuova soluzione, tra quelli che nella soluzione x* erano collocati, ora non lo sono più oppure lo sono ma con un tipo diverso rispetto ad x*.

2. $$\sum_{k\in Y^*_{LE}}\left(1-y_k^{LE}\right) \qquad (5.4.1.3)$$

Si consideri un generico k ∈ Y_{LE}*, ossia un generico LE k che nella soluzione x* risulta collocato.

Ora,se $y_k^{LE}=1$ (si colloca l'LE k nella nuova soluzione), allora $1-y_k^{LE}=0$, mentre se $y_k^{LE}=0$ (non si colloca l'LE k nella nuova soluzione), allora $1-y_k^{LE}=1$

Il termine k-esimo dell'espressione (5.4.1.3) è uguale a 0 se e solo se nella nuova soluzione l'LE k-esimo è collocato.

Di conseguenza, l'espressione (5.4.1.3) conta quanti LE della nuova soluzione, tra quelli che nella soluzione x* erano collocati, ora non lo sono più.

3. $$\sum_{j \notin Y^*_{CSS}} \sum_{h=1,2} y_j^{CSS-h} \equiv \sum_{j \notin Y^*_{CSS}} \left(y_j^{CSS-1} + y_j^{CSS-2} \right) \qquad (5.4.1.4)$$

Si consideri un generico j $\notin$ Y_{CSS}*, ossia un generico CSS j che nella soluzione x* risulta non collocato.

Ora, se nella nuova soluzione il CSS j continua a restare non collocato, si ha $y_j^{CSS-1} + y_j^{CSS-2} = 0$, altrimenti, se viene collocato con un qualche tipo (1 o 2), si ha $y_j^{CSS-1} + y_j^{CSS-2} = 1$.

L'espressione (5.4.1.4) conta perciò quanti CSS della nuova soluzione, tra quelli che nella soluzione x* non erano collocati, ora lo sono.

4. $$\sum_{k \notin Y^*_{LE}} y_k^{LE} \qquad (5.4.1.5)$$

Considerando un generico k $\notin$ Y_{LE}*, ossia un generico LE k che nella soluzione x* risulta non collocato si ha che, se nella nuova soluzione continua ad essere non collocato, il termine k-esimo dell'espressione (5.4.1.5) vale 0, altrimenti vale 1.

L'espressione (5.4.1.5) conta quindi quanti LE della nuova soluzione, tra quelli che nella soluzione x* non erano collocati, ora lo sono.

5. $$\sum_{(i,j) \in X^*_{BTS-CSS}} \left(1 - x_{ij}^{BTS-CSS} \right) \qquad (5.4.1.6)$$

Tale espressione conta quanti collegamenti tra BTS e CSS nella nuova soluzione, fra quelli che nella soluzione x* erano attivi, ora non lo sono più.

6. $$\sum_{(i,j) \notin X^*_{BTS-CSS}} x_{ij}^{BTS-CSS} \qquad (5.4.1.7)$$

Tale espressione conta quanti collegamenti tra BTS e CSS nella nuova soluzione, fra quelli che nella soluzione x* non erano attivi, ora lo sono.

7. $$\sum_{(i,k) \in X^*_{BTS-LE}} \left(1 - x_{ik}^{BTS-LE} \right) \qquad (5.4.1.8)$$

Si conta quanti collegamenti tra BTS ed LE nella nuova soluzione, fra quelli che nella soluzione x* erano attivi, ora non lo sono più.

8. $$\sum_{(i,k)\notin X^*_{BTS-LE}} x_{ik}^{BTS-LE} \qquad (5.4.1.9)$$

Si conta quanti collegamenti tra BTS ed LE nella nuova soluzione, fra quelli che nella soluzione x* non erano attivi, ora lo sono.

9. $$\sum_{(j,k)\in X^*_{CSS-LE}} \left(1 - x_{jk}^{CSS-LE}\right) \qquad (5.4.1.10)$$

Da tale espressione risulta il numero di collegamenti tra CSS ed LE, fra quelli attivi nella soluzione x*, che nella nuova soluzione non lo sono più.

10. $$\sum_{(j,k)\notin X^*_{CSS-LE}} x_{jk}^{CSS-LE} \qquad (5.4.1.11)$$

Da tale espressione risulta il numero di collegamenti tra CSS ed LE, fra quelli non attivi nella soluzione x*, che nella nuova soluzione lo sono.

In conclusione, l'espressione (5.4.1.1) esprime la distanza tra la nuova soluzione x ed una particolare soluzione ammissibile fissata x* come il numero di dispositivi e di collegamenti che cambiano di stato nel passaggio da una all'altra.

A questo punto è bene portare la (5.4.1.1) in un'altra forma, del tutto equivalente ma più adatta ai nostri scopi:

$$\Delta(x,y,x^*,y^*) = \sum_{j\in Y^*_{CSS}} \left[1 - \sum_{h=1,2}\left(y_j^{CSS-h^*} y_j^{CSS-h}\right)\right] + \sum_{k\in Y^*_{LE}}\left(1 - y_k^{LE}\right) + \sum_{j\notin Y^*_{CSS}}\sum_{h=1,2} y_j^{CSS-h} +$$

$$+ \sum_{k\notin Y^*_{LE}} y_k^{LE} + \sum_{j=1}^{m}\left[\sum_{i\in X^*_{BTS-CSS(j)}}\left(1 - x_{ij}^{BTS-CSS}\right) + \sum_{i\notin X^*_{BTS-CSS(j)}} x_{ij}^{BTS-CSS}\right] + \sum_{k=1}^{p}\left[\sum_{i\in X^*_{BTS-LE(k)}}\left(1 - x_{ik}^{BTS-LE}\right) +\right.$$

$$\left. + \sum_{i\notin X^*_{BTS-LE(k)}} x_{ik}^{BTS-LE} + \sum_{j\in X^*_{CSS-LE(k)}}\left(1 - x_{jk}^{CSS-LE}\right) + \sum_{j\notin X^*_{CSS-LE(k)}} x_{jk}^{CSS-LE}\right] \qquad (5.4.1.12)$$

dove:

- $X^*_{BTS-CSS(j)}$ = { (i,j), i ∈ {1,…,n} : $x_{ij}^{BTS-CSS^*}$ = 1 } è l'insieme dei collegamenti attivi fra le BTS ed il CSS j nella soluzione x*
- $X^*_{BTS-LE(k)}$ = { (i,k), i ∈ {1,…,n} : $x_{ik}^{BTS-LE^*}$ = 1 } è l'insieme dei collegamenti attivi fra le BTS e l'LE k nella soluzione x*
- $X^*_{CSS-LE(k)}$ = { (j,k), j ∈ {1,…,m} : $x_{jk}^{CSS-LE^*}$ = 1 } è l'insieme dei collegamenti attivi fra i CSS e l'LE k nella soluzione x*

Per esprimere gli spazi soluzione corrispondenti agli intorni del VNS verranno usati due tipi di vincolo, entrambi basati sull'espressione (5.4.1.1):

$$k_1 \leq \Delta(x, y, x^*, y^*) \leq k_2 \qquad (5.4.1.13)$$

per la fase di diversificazione (shaking), e

$$\Delta(x, y, x^*, y^*) = k' \qquad (5.4.1.14)$$

per la fase di intensificazione (local search).

La (5.4.1.13) impone che vi siano almeno k_1 e non più di k_2 cambiamenti sullo stato dei dispositivi e dei collegamenti nel passaggio dalla particolare soluzione ammissibile fissata x* alla nuova soluzione x.

La (5.4.1.14) impone invece che tali cambiamenti siano esattamente k'.

Visti gli scopi per cui vengono utilizzati i due vincoli (il primo per creare un insieme delle soluzioni vasto e lontano da x*, il secondo per crearne uno ristretto e limitrofo ad x*), è evidente che i valori assegnati ai parametri k_1, k_2 e k' saranno tali che k_1, k_2 >> k'.

Alla fine di ogni fase di ricerca sarà inoltre applicato al minimo locale scelto come punto di partenza per la nuova diversificazione un terzo tipo di vincolo (che andrà ad aggiungersi in maniera permanente al sistema), anch'esso basato sull'espressione (5.4.1.1):

$$\Delta(x, y, x^*, y^*) \geq 1 \qquad (5.4.1.15)$$

Si tratta di un vincolo di tipo "tabu", avente cioè la funzione di impedire all'algoritmo di riconsiderare in futuro il minimo locale in questione; c'è infatti il rischio che, essendo a volte l'algoritmo abilitato a passare per minimi locali peggiorativi, si finisca in un minimo già visitato e già scelto come soluzione di partenza per le successive fasi di diversificazione ed intensificazione, e si ripeta così una serie di esplorazioni già effettuate in precedenza.

C'è però da sottolineare il difetto che, così com'è espresso, il vincolo "tabu" è molto selettivo, ossia interdice all'algoritmo la futura visita del minimo, ma non quella di soluzioni ad esso molto vicine (differenti al limite anche solo per lo stato di un dispositivo o di un collegamento); se una di tali soluzioni venisse scelta come nuovo minimo locale di partenza per la successiva fase di diversificazione, non si sarebbe risolto nulla (anche non partendo esattamente dal minimo locale di prima, si finirebbe infatti comunque in regioni già esplorate in precedenza). D'altronde, si può per questo sempre decidere di modificare il coefficiente a secondo membro del vincolo e di portarlo da 1 a valori più elevati, in modo da escludere la visita, oltre che del minimo locale, anche di un suo intorno di dimensione opportuna.

5.4.2 Pseudocodice dell'algoritmo

Viene ora esposto qui di seguito lo pseudocodice utilizzato per implementare l'algoritmo. Si precisa che:

con ***best_sol*** si intende la soluzione (x, y, w, z) incombente (la miglior soluzione ammissibile corrente)

con ***start_sol*** si intende il minimo locale a partire dal quale si effettua la nuova ricerca, tramite diversificazione, di un altro minimo (preme puntualizzare che, per come è stato progettato l'algoritmo, nel corso dell'esecuzione *start_sol* può essere diverso da *best_sol*)

con ***new_local_sol*** si intende il nuovo minimo locale, ossia la soluzione ottenuta dalla ricerca locale appena svolta.

setta i parametri di Cplex per la ricerca generale;

costruisci e risolvi il modello (0)-(16) con Cplex, fermandoti alla prima soluzione ammissibile;

best_sol := prima soluzione ammissibile;

start_sol := *best _sol*;

$k_1 := k_{min}$; $k_2 := k_1 + k_{step}$;

finchè il tempo di esecuzione non supera il tempo massimo

{

aggiungi al modello corrente il vincolo di diversificazione $k_1 \leq \Delta(x, y, start_sol) \leq k_2$ intorno alla soluzione *start_sol*;

setta i parametri di Cplex per la diversificazione;

risolvi il modello con Cplex, fermandoti alla l-esima soluzione ammissibile (o allo scadere di un tempo limite predefinito);

commento algoritmo di ricerca locale;

$k' := k'_{min}$;

finchè non si è raggiunto l'intorno limite $k' = k'_{max}$ e il tempo di esecuzione non supera il tempo massimo

{

aggiungi al modello il vincolo $\Delta(x, y, current_sol) = k'$ settandone i parametri e risolvendolo con Cplex per l'intensificazione;

se hai trovato una soluzione migliorativa

current_sol := nuova soluzione corrente; $k' := k'_{min}$;

altrimenti

$k' := k' + k'_{step}$;

```
            rimuovi il vincolo di intensificazione dal modello;
        }

        rimuovi il vincolo di diversificazione dal modello;
        new_local_sol := current_sol;

        se costo(new_local_sol) < costo(best_sol)
        {
            best_sol := new_local_sol;
            start_sol := best_sol;
            aggiungi al modello il vincolo di tabu Δ(x, y, start_sol) ≥ 1
            intorno a start_sol;
            k1 := kmin;      k2 := k1 + kstep;
        }
        altrimenti
            se k1 < kmax
                k1 := k1 + kstep;      k2 := k1 + kstep;
            altrimenti
            {
                start_sol := new_local_sol;
                aggiungi al modello il vincolo di tabu Δ(x, y, start_sol) ≥ 1
                intorno a start_sol;
                k1 := kmin;      k2 := k1 + kstep;
            }
    }

    visualizza best_sol;
```

5.4.3 Commenti allo pseudocodice

La struttura algoritmica è quella classica del VNS: sulla base della miglior soluzione ammissibile x a disposizione al momento, si costruiscono degli intorni $N_i(x)$ a distanza via via crescente, e per ciascuno di essi si sceglie una soluzione a caso x_i' a cui si applica un algoritmo iterativo di discesa per arrivare ad un minimo locale x_i'': il processo di estensione degli intorni terminerà quando, applicando l'algoritmo di discesa ad una soluzione x_q' scelta casualmente dentro ad un intorno $N_q(x)$, si verrà condotti ad un minimo locale x_q'' di costo inferiore ad x; a quel punto, x_q'' diventerà la nuova soluzione incombente, e su di essa verrà riapplicato il tipo di ricerca appena esposto.

Supponendo che lo spazio soluzione del problema sia bidimensionale (un sottoinsieme di R^2), graficamente si può rappresentare il processo di estensione degli intorni di x e delle corrispondenti ricerche locali nel seguente modo:

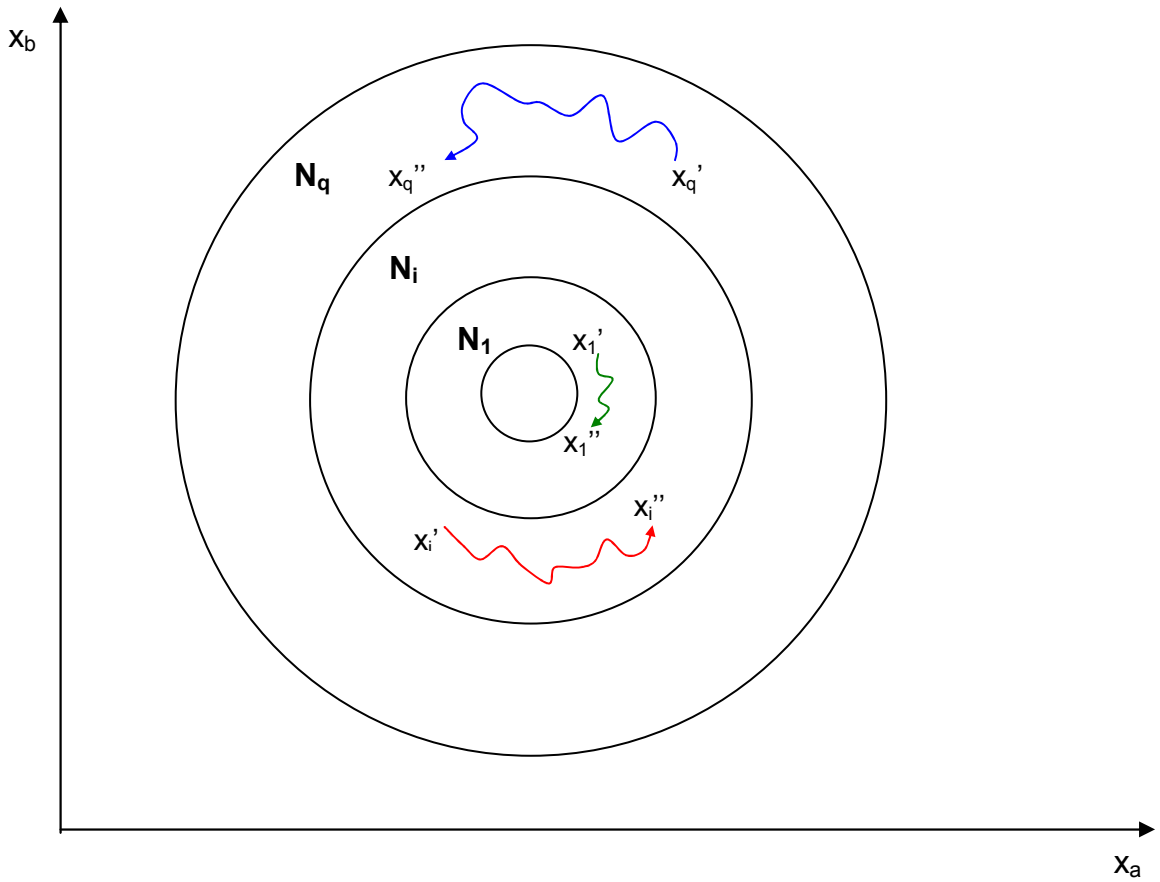

Figura 5.4.3.1 : Interpretazione grafica del metodo VNS

Come si può notare dalla figura, la traiettoria della generica ricerca locale i-esima (avente origine dal punto $x_i' \in N_i(x)$) rimane sempre racchiusa nel relativo $N_i(x)$, in quanto il vincolo che ne implementa l'obbligo di appartenenza viene mantenuto fino al raggiungimento del minimo locale x_i''. Ciò è garanzia del fatto che i vari x_i'' saranno distinti fra loro e da x, evitando in tal modo di ripetere traiettorie già analizzate (Fig. 5.4.3.2) o di ricadere nei paraggi della soluzione incombente (Fig. 5.4.3.3):

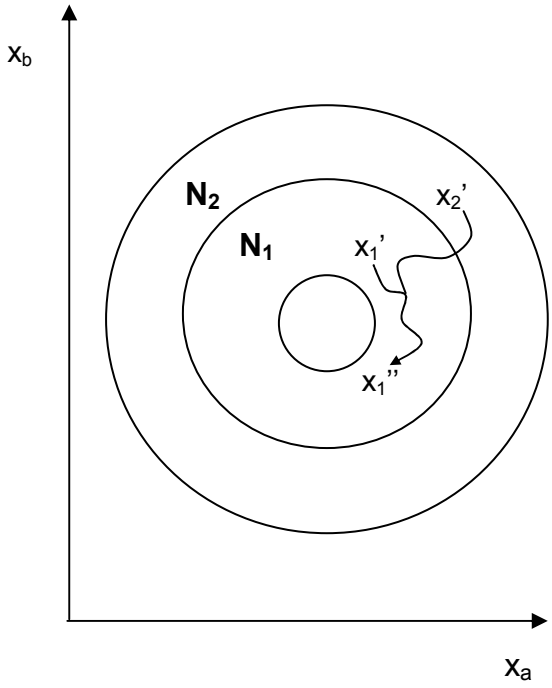

Figura 5.4.3.2

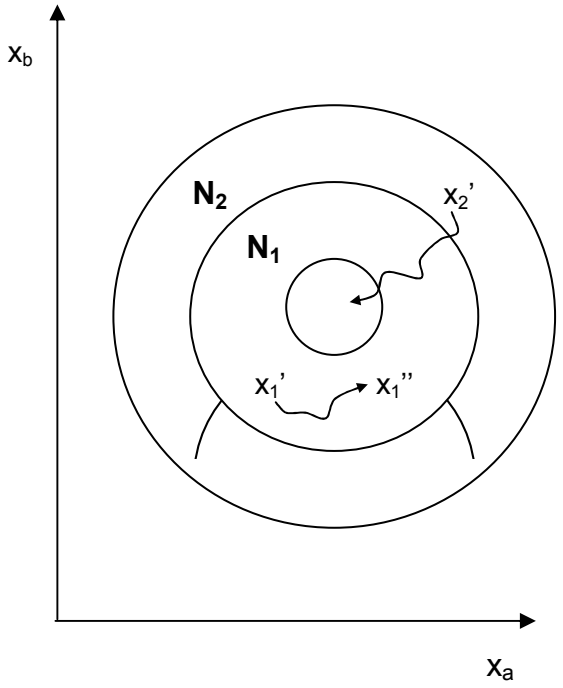

Figura 5.4.3.3

Possibili problemi derivanti dalla non appartenenza della traiettoria i-esima di ricerca locale all'intorno $N_i(x)$

Può anche capitare che, nonostante si considerino intorni $N_i(x)$ sempre più lontani da x, si ottengano dei minimi locali x_i'' che non riescono a migliorare il costo di x. Chiaramente questo processo di allontanamento deve avere un limite, non solo per questioni di tempo, ma anche perché tutte le soluzioni oltre una certa distanza da x sono non ammissibili: considerare un intorno molto lontano significa imporre un vincolo che obbliga la rete a cambiare lo stato ad un numero elevato di dispositivi e/o di collegamenti, insidiando così l'ammissibilità della rete stessa.

Ecco perché si impone un upper-bound al numero di intorni $N_i(x)$ costruibili sulla base di x: se, una volta analizzato $N_{max}(x)$, non si è ancora riusciti a trovare un ottimo locale che migliori il costo di x, si arresta tale fase di ricerca avente x come origine, si considera l'ultimo minimo locale trovato in ordine di tempo e, sulla base di questo, si ricomincia il processo di costruzione ed analisi degli intorni a distanza crescente.

Il fatto di permettere all'algoritmo di accettare un minimo locale peggiorativo nasconde però, come già detto, l'insidia di selezionare minimi già scelti in passato come soluzioni di partenza per nuove ricerche, e di ciclare così all'infinito. Ci si può salvaguardare da tale pericolo aggiungendo al modello, per ogni nuovo minimo locale di partenza (o "*start_sol*", come definito nello pseudocodice), il vincolo permanente (5.4.1.15), il quale impone l'esclusione di quella particolare configurazione di rete da tutte le ricerche future.

L'algoritmo sfrutta la variante VND perché applica il metodo di esplorazione per intorni variabili anche alla ricerca locale di discesa: si considera la generica soluzione corrente, e sulla base di questa si costruisce il vincolo (5.4.1.14) con k' = 2 (k'_{min} = 2) e lo si aggiunge al modello; se nell'intorno così creato non si trova una soluzione migliorativa, il valore di k' viene portato da 2 a 4 (k'_{step} = 2) e riscritto nel vincolo, in maniera tale da cambiare l'intorno di esplorazione ed analizzare le regioni successive. Il passo di incremento del parametro k' è 2 e non 1 perché il cambiamento più elementare che possa essere eseguito su una rete è il cambio di destinazione di un dispositivo, il che significa lo spegnimento di un collegamento e l'attivazione di un altro, e dunque il cambio di stato di 2 variabili della rete stessa.

Durante la ricerca locale viene anche effettuato un controllo sul tempo di esecuzione trascorso; ciò è necessario in quanto, vista la scarsa velocità di convergenza di tale ricerca (i miglioramenti da una soluzione all'altra possono infatti essere assai esigui), se il controllo venisse effettuato solo dopo il ritrovamento del minimo locale si correrebbe il rischio di oltrepassare abbondantemente il tempo limite complessivo fissato per l'esecuzione. In questo modo, invece, non appena il suddetto tempo limite viene superato, si esce all'istante dalla ricerca locale e si termina il programma.

La pedissequa applicazione della variante RVNS avrebbe voluto che il vincolo di intensificazione (5.4.1.14) avesse la stessa forma di quello di diversificazione (5.4.1.13), ossia $k_1' \leq \Delta(x, y, x^*, y^*) \leq k_2'$, naturalmente con k_1', k_2' << k_1, k_2. Questo avrebbe dunque comportato, in corrispondenza di una generica soluzione corrente durante la ricerca locale, la creazione e l'analisi sequenziale dei seguenti intorni (fino alla scoperta di una soluzione migliorativa o al raggiungimento dell'intorno limite k'_{max}):

$0 \leq \Delta(x, y, x^*, y^*) \leq 2$

$4 \leq \Delta(x, y, x^*, y^*) \leq 6$

$8 \leq \Delta(x, y, x^*, y^*) \leq 10$

…….

Tale analisi presenta però il difetto che ogni intorno contiene una notevole quantità di soluzioni non ammissibili (e pertanto non interessanti): nel generico intorno le uniche soluzioni ammissibili sono infatti quelle che soddisfano il vincolo con l'uguaglianza (al limite inferiore o superiore).

Sostituendo la sequenza d'analisi dei precedenti vincoli con la seguente

$\Delta(x, y, x^*, y^*) = 2$

$\Delta(x, y, x^*, y^*) = 4$

$\Delta(x, y, x^*, y^*) = 6$

…….

(il vincolo $\Delta(x, y, x^*, y^*) = 0$ è superfluo in quanto riproporrebbe la soluzione corrente), si restringe la ricerca pur senza escludere da questa alcuna soluzione ammissibile, evitando così inutili perdite di tempo.

Per quel che riguarda invece il vincolo $k_1 \leq \Delta(x, y, x^*, y^*) \leq k_2$ di diversificazione, ai parametri k_1, k_2 (limiti inferiore e superiore del vincolo) e k_{step} (passo di incremento di k_1 e k_2) sono assegnati valori dell'ordine delle decine; questo perché, trovandosi nella fase cosiddetta di "shaking", l'algoritmo deve avere la possibilità di effettuare delle grosse variazioni nella configurazione della rete, e quindi anche di cambiare la collocazione dei CSS e degli LE. Infatti, se la rete corrente è ben bilanciata, ad ogni dispositivo afferisce tipicamente un significativo numero di dispositivi di livello inferiore; ad un'eventuale deallocazione dovranno perciò seguire numerose riassegnazioni di collegamenti, e se i valori di k_1 e k_2 fossero bassi (sull'ordine delle unità), ciò non sarebbe consentito.

E' consigliabile (come suggerisce anche l'RVNS) non arrestare la diversificazione alla prima soluzione ammissibile incontrata, ma proseguire fino a che non si incontra la l-esima migliorativa o non è scaduto un determinato tempo limite. In tal modo, l'algoritmo viene avvantaggiato nel senso che può far cominciare la ricerca locale già da una buona soluzione; si tratta di un vantaggio sicuramente non trascurabile, visto che la convergenza della suddetta ricerca è piuttosto lenta: data la ridotta estensione (e distanza dalla soluzione corrente) degli intorni esplorati durante tale fase, i miglioramenti di costo relativi ad ogni passo sono in genere molto contenuti.

5.5 Esempi di test e risultati

Si è applicato il metodo VNS al test bed di esempi riportato al Par. 2.2 e già utilizzato come banco di prova per i precedenti metodi ai Par. 3.3 e 4.4; prima di esporre i risultati ottenuti, si presentano i parametri utilizzati nelle varie fasi di ricerca (ricerca iniziale basata sul modello (0) – (16) dell'articolo [1], diversificazione ed intensificazione), i valori ad essi assegnati a seconda della classe di test, ed una breve giustificazione sul loro uso.

Parametri dell'algoritmo principale

- Tempo limite totale di esecuzione = 6^h ; 8^h ; 10^h ; 12^h (su PC Pentium II / 450 MHz) rispettivamente per le classi 1, 2, 3, 4: non appena l'algoritmo effettua un controllo sul tempo trascorso e si accorge che si è oltrepassato il tempo limite totale, l'esecuzione del programma termina.
- Massimo numero di diversificazioni = 1.000: l'algoritmo conta il numero di soluzioni distinte utilizzate come punti di partenza per le fasi di diversificazione (ossia il numero di volte che l'identificatore *start_sol* viene reinizializzato ad una nuova soluzione). Non appena tale numero supera il numero massimo, l'esecuzione del programma termina.
- Limite massimo di memoria = 128 MB
- Indicatore di gestione memoria in sovraccarico = 1: non appena la dimensione dell'albero supera il limite di memoria stabilito, i nodi successivi vengono memorizzati in file di salvataggio.

Parametri per l'iniziale ricerca generale

- Tempo limite = 60' (su PC Pentium II / 450 MHz)
- Massimo numero di nodi = illimitato (5.000.000)
- Massimo numero di soluzioni ammissibili migliorative = 1: non appena viene oltrepassato il tempo limite o viene esplorato il massimo numero di nodi oppure viene trovato il massimo numero di soluzioni ammissibili migliorative, Cplex termina la propria ricerca.
- MIPEmphasis = 1: durante la ricerca, Cplex dà la precedenza all'ammissibilità più che all'ottimalità.

Parametri per la diversificazione

- Tempo limite = 30' (su PC Pentium II / 450 MHz)
- Massimo numero di nodi = illimitato (5.000.000)
- Massimo numero di soluzioni ammissibili migliorative = 7: come prima, non appena uno di tali limiti viene raggiunto, Cplex termina la propria ricerca.
- MIPEmphasis = 1: durante la ricerca, visto che gli insiemi soluzione esplorati nel corso di tale fase sono ancora troppo estesi per poter ragionevolmente pensare di ottenerne l'ottimo, si preferisce accontentarsi di una buona soluzione ammissibile.

- $k_{min} = 50$, $k_{max} = 200$, $k_{step} = 50$: ciò significa che, data una certa soluzione di partenza *start_sol*, il primo intorno di diversificazione avrà come estremi $k_1 = 50$ e $k_2 = 100$ ($50 \leq \Delta(x, y, x^*, y^*) \leq 100$); se il minimo locale trovato con la conseguente intensificazione non migliora la soluzione incombente, si passerà all'intorno di diversificazione con estremi $k_1 = 100$ e $k_2 = 150$, e così via, fino a che non si troverà un minimo locale migliorativo o non si giungerà all'intorno limite, costituito da $k_1 = 200$ e $k_2 = 250$.

Parametri per l'intensificazione

- Tempo limite = 3' ; 5' ; 7' ; 10' (su PC Pentium II / 450 MHz) rispettivamente per le classi 1, 2, 3, 4.
- Massimo numero di nodi = illimitato (5.000.000): anche qui, il primo limite ad essere oltrepassato causerà l'arresto della ricerca.
- MIPEmphasis = 0
- EpAGap = 10^{-9}
- EpGap = 10^{-9}
- CutUp = costo soluzione corrente: visto che gli intorni esplorati nel corso di tale fase sono di estensione limitata, si può a buon diritto impostare la ricerca all'ottenimento dell'ottimo (addirittura dell'ottimo assoluto dell'intorno, e non di una soluzione ammissibile prossima al lower bound; in virtù di questo fatto, le due tolleranze sul gap sono settate praticamente a zero). Si è quindi utilizzato il parametro di CutUp e lo si è fissato al costo della soluzione corrente: dato che lo scopo del singolo passo della ricerca locale è trovare una soluzione migliorativa rispetto alla corrente, evitare la ramificazione dei nodi dell'albero aventi un costo superiore costituisce senza dubbio un prezioso guadagno di tempo.
- $k'_{min} = 2$, $k'_{max} = 6$, $k'_{step} = 2$: ciò significa che, data una certa soluzione corrente, se ne esaminerà prima l'intorno caratterizzato da $k' = 2$ ($\Delta(x, y, x^*, y^*) = 2$), poi, in mancanza di una soluzione migliorativa rispetto alla corrente, si incrementerà k' a 4, ed infine lo si porterà eventualmente a 6.

In tutte le varie fasi di esplorazione, il parametro relativo al massimo numero di nodi (oltre cui far terminare a Cplex la ricerca corrispondente) è stato posto ad un valore molto elevato (5.000.000); questo perché si è voluta regolare la durata delle ricerche esclusivamente sulla base dei tempi limite. In alcuni casi, d'altra parte, potrebbe essere conveniente fissare il massimo numero di nodi (anziché il tempo limite) in quanto è ragionevole pensare che, dopo l'esplorazione di un certo numero di nodi dell'albero, Cplex fornisca una soluzione soddisfacente, o comunque non abbia più senso proseguire oltre.

Sono stati così introdotti entrambi i tipi di parametri onde fornire all'utente la possibilità di scegliere, per ogni fase di ricerca, quello da lui ritenuto più opportuno.

Ecco i risultati (preceduti da un riassunto dei parametri di simulazione più significativi e dei valori ad essi associati):

Input : test bed descritto al Par. 1.6

Metodo : Variable Neighbourhood Search (VNS)

Tipo hardware : Pentium II 450 MHz, 128 MB RAM

Sistema Operativo : Windows 98

Parametri di ricerca di maggior rilevanza

Algoritmo principale
- Tempo limite tot di esecuzione = 6^h ; 8^h ; 10^h ; 12^h
- Memoria limite (TreLim) = 128 MB
- Indicatore gestione memoria (NodeFileInd) = 1

Iniziale ricerca generale
- Tempo limite (TiLim) = 60'
- Max n° soluzioni migliorative (IntSolLim) = 1
- Indicatore dell'enfasi (MIPEmphasis) = 1

Diversificazione (shaking)
- Tempo limite (TiLim) = 30'
- Max n° soluzioni migliorative (IntSolLim) = 7
- Indicatore dell'enfasi (MIPEmphasis) = 1
- $k_{min} = 50$, $k_{max} = 200$, $k_{step} = 50$

Intensificazione (local research)
- Tempo limite (TiLim) = 3' ; 5' ; 7' ; 10'
- Indicatore dell'enfasi (MIPEmphasis) = 0
- Tolleranza assoluta (EpAgap) = 10^{-9}
- Tolleranza relativa (EpGap) = 10^{-9}
- Limite superiore di cutoff (CutUp) = costo sol. corrente
- $k'_{min} = 2$, $k'_{max} = 6$, $k'_{step} = 2$

Classi	Esempi	Costo miglior soluzione ammissibile		
		VNS	Cplex	Tabu Search
1	A (50, 4, 2)	**10.760.529**	**10.760.529**	**10.760.529**
	B (55, 5, 3)	**11.619.817**	**11.619.817**	**11.619.817**
	C (60, 5, 4)	**12.393.173**	12.396.436	12.395.078
	D (65, 6, 4)	**13.376.659**	**13.376.659**	13.445.309
2	E (100, 8, 4)	19.740.675	**19.734.790**	20.006.714
	F (105, 9, 5)	**21.951.866**	21.952.231	22.151.922
	G (110, 9, 6)	22.973.105	**22.971.543**	23.100.774
	H (115, 10, 6)	**23.979.111**	23.996.482	23.999.151
3	I (150, 12, 6)	**30.137.934**	30.142.363	30.139.211
	L (155, 13, 7)	31.137.306	**31.098.048**	31.152.784
	M (160, 13, 8)	**32.035.223**	32.045.632	32.070.411
	N (165, 14, 8)	**32.856.421**	32.866.855	32.896.514
4	O (200, 20, 10)	39.247.134	39.282.420	**39.241.845**

Tabella 5.5.1 : costo delle migliori soluzioni ammissibili fornite dal metodo VNS e confronto con le soluzioni di Cplex e Tabu Search a parità di tempo di esecuzione (il costo migliore di ogni esempio è evidenziato in grassetto)

La tabella evidenzia come le soluzioni ottenute col metodo VNS, a confronto con quelle offerte da Cplex e dalla Tabu Search, siano molto competitive: il VNS si rivela infatti il metodo migliore in buona parte degli esempi e, anche quando ciò non accade, la soluzione da esso fornita si colloca a ridosso della migliore di turno. Le prestazioni di Cplex e del VNS sono comunque molto simili, in quanto le differenze fra le rispettive migliori soluzioni ottenute nei vari esempi sono dell'ordine di poche migliaia; gli unici casi in cui si manifestano dei divari più consistenti sono l'esempio L della classe 3 (a favore di Cplex) e l'esempio O della classe 4 (a favore del VNS). Entrambi tali metodi sono complessivamente preferibili (ed in maniera piuttosto netta) alla Tabu Search, la quale alterna buoni risultati ad altri più scadenti (soprattutto nella classe 2).

Si espone ora una serie di tabelle ed una di grafici (entrambe relative ai soli esempi della classe 3 per ragioni di spazio) in cui si mette in evidenza il comportamento del metodo VNS all'interno della singola simulazione (risultati intermedi più significativi ed andamento complessivo della funzione costo):

Esempio I

n°	Diversif	Intensif	Tempo
1	30.277.979	30.183.507	1h 33' 33"
2	30.259.135	30.236.900	2h 26' 51"
3	30.358.838	30.178.683	4h 11' 58"
4	30.270.785	**30.137.934**	5h 55' 15"
5	30.215.773	30.201.513	6h 26' 16"
6	30.227.581	30.196.246	8h 20' 12"
7	30.165.955	30.149.957	9h 36' 08"
8	30.299.100	30.269.687	10h 02' 38"

Esempio L

n°	Diversif	Intensif	Tempo
1	31.253.430	31.223.522	1h 21' 53"
2	31.198.951	31.169.120	3h 02' 16"
3	31.188.944	31.160.537	4h 19' 59"
4	31.208.310	31.156.652	4h 51' 43"
5	31.162.984	**31.137.306**	6h 19' 49"
6	31.222.304	31.157.794	7h 13' 14"
7	31.282.209	31.137.314	8h 00' 54"
8	31.140.915	31.140.702	8h 47' 06"
9	31.353.238	31.149.398	10h 00' 18"

Esempio M

n°	Diversif	Intensif	Tempo
1	32.181.268	32.095.919	1h 41' 57"
2	32.118.923	**32.035.223**	7h 10' 13"
3	32.038.919	32.038.919	7h 16' 43"
4	32.203.301	32.059.303	9h 43' 15"
5	32.217.229	32.040.247	10h 00' 42"

Esempio N

n°	Diversif	Intensif	Tempo
1	32.964.705	32.865.371	2h 31' 24"
2	32.894.428	32.874.708	3h 51' 44"
3	32.966.522	32.872.574	5h 28' 35"
4	33.016.944	32.862.897	7h 20' 46"
5	32.892.708	**32.856.421**	8h 12' 13"
6	32.892.089	32.866.498	10h 00' 51"

Tabelle 5.5.2 : risultati intermedi (esiti delle varie fasi di diversificazione ed intensificazione) delle simulazioni, svolte col metodo VNS, relativamente agli esempi della classe 3 (in ogni tabella è evidenziato in grassetto il costo minimo); i tempi sono relativi ad un PC Pentium II / 450 MHz

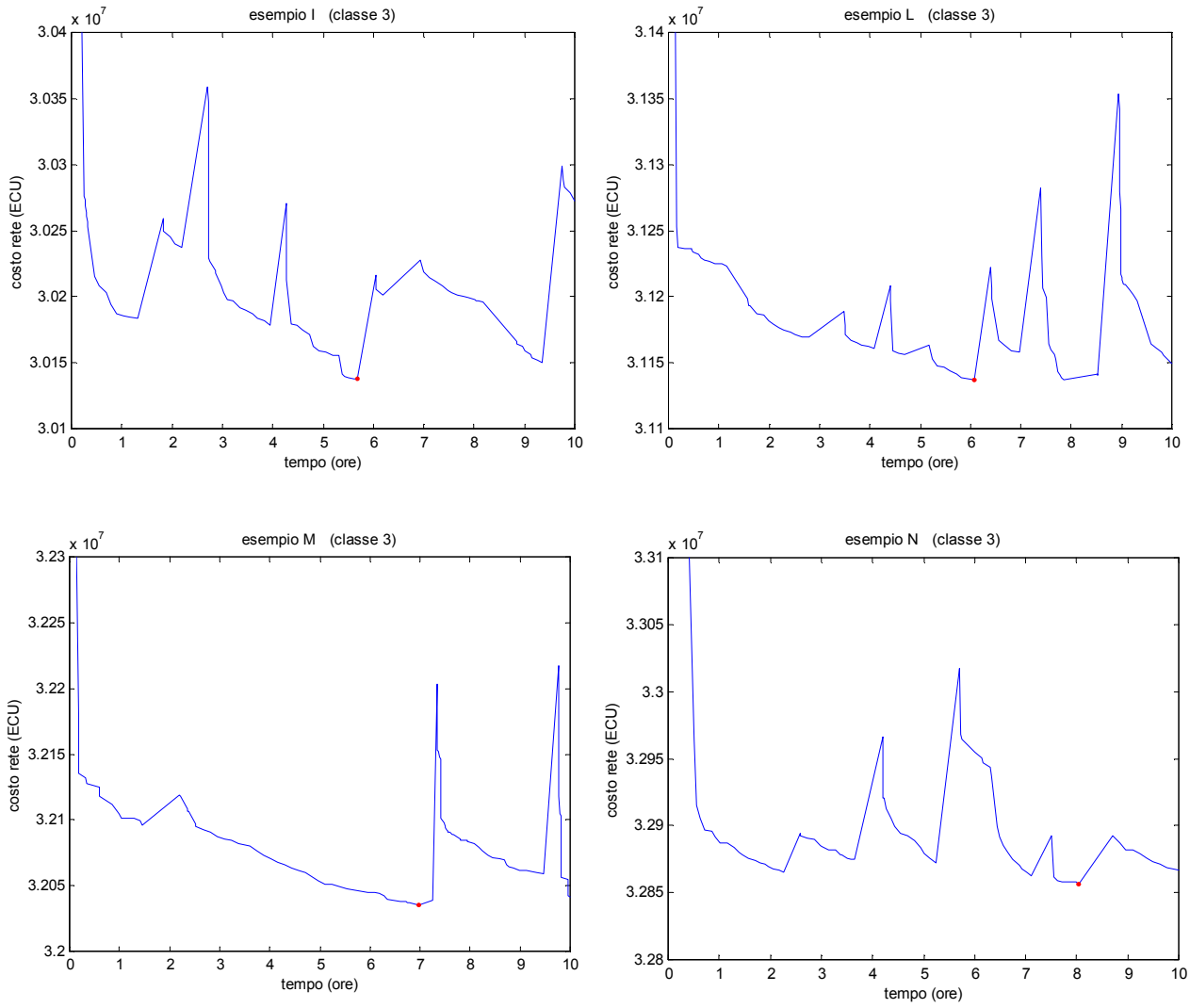

Figure 5.5.1 : andamenti complessivi delle funzioni costo delle simulazioni, svolte col metodo VNS, relativamente agli esempi della classe 3 (in ogni grafico è posto in risalto il punto corrispondente al costo minimo)

I bruschi peggioramenti del costo che si notano nelle figure corrispondono ai passaggi dalle fasi di intensificazione a quelle di diversificazione, mentre le discese (che sono in genere piuttosto lente, soprattutto dopo i primi minuti) corrispondono alle ricerche locali (ciascuna delle quali conclusa, logicamente, dal proprio minimo locale).

Dalle tabelle e dai grafici si può dedurre come, nel corso di ogni simulazione, vi sia più di una fase di intensificazione che conduce ad un minimo locale di costo paragonabile a quello minimo; questo denota una buona capacità di convergenza del metodo, e dunque una ridotta influenza della componente di casualità sulla qualità del costo della soluzione finale (non tutti i minimi locali appartengono comunque al medesimo livello di costo: parte delle intensificazioni converge infatti prematuramente, e conduce a soluzioni di scarsa qualità.

Val forse la pena far notare alcuni particolari interessanti emersi dalle prove.

Nel corso di una generica ricerca locale, raramente (più che altro in partenza) si riesce a diminuire il costo della soluzione corrente con k' = 2; molto più frequente è invece il miglioramento ottenuto con k' = 4; ciò significa, in termini di configurazione di rete, che per migliorare la soluzione corrente bisogna spesso riassegnare almeno due dispositivi; questo è in sintonia con quanto accadeva nella Tabu Search, dove la mossa più frequentemente utilizzata era non tanto lo spostamento di una BTS, quanto lo scambio delle destinazioni di due BTS (caso particolare della riassegnazione di due dispositivi).

Sempre nel corso della generica ricerca locale, si è osservato che le collocazioni dei CSS e degli LE rimangono rigorosamente inalterate; ciò era prevedibile per quanto detto prima: dato che i vincoli di volta in volta aggiunti impongono piccole variazioni strutturali della rete corrente, è del tutto improbabile che venga allocato o deallocato un dispositivo. Se venisse allocato, infatti, gli si potrebbero far afferire pochissimi dispositivi, creando così una rete sbilanciata il cui costo sicuramente non migliorerebbe quello corrente. Se invece il dispositivo in questione venisse deallocato, vorrebbe dire che è quasi scarico (i vincoli di ricerca locale non consentirebbero altrimenti tutte le riassegnazioni dei dispositivi ad esso connessi); imbattersi però in una rete sbilanciata (e dunque dal costo elevato) durante una fase di ricerca locale è assai difficile, in quanto tale ricerca parte già da una buona soluzione iniziale, e nel seguito ottiene un progressivo miglioramento del costo.

Ora, visto che durante la ricerca locale le collocazioni dei dispositivi restano immutate, un'idea (che sarà sfruttata nel metodo esposto nel prossimo capitolo) potrebbe essere quella di inserire dei vincoli che impongono le collocazioni suddette: tali vincoli velocizzerebbero infatti le esplorazioni, in quanto fisserebbero già in partenza i valori di parecchie variabili del modello; precisamente, essi imporrebbero in modo diretto i valori alle $y_j^{CSS\text{-}h}$ (j = 1,…,m; h = 1,2) ed alle y_k^{LE} (k = 1,…,p), nonché forzerebbero indirettamente al valore 0 le x, w e z relative a collegamenti che partono da (o arrivano in) dispositivi le cui y sono fissate a 0 (se un dispositivo non è collocato, è ovvio che non può essere connesso al resto della rete).

L'idea di fissare le collocazioni di CSS ed LE si rivelerà tanto efficace da meritare una fase di ricerca tutta per sé, ossia una fase durante la quale gli unici vincoli che si aggiungeranno al modello principale (oltre naturalmente ai vincoli permanenti di tabu) saranno quelli che imporranno la collocazione dei suddetti dispositivi. Si vedrà infatti che l'ambiziosa convinzione di trovare con un'unica esplorazione la migliore soluzione relativa ad una particolare configurazione di apparati sarà spesso verificata. Nel malaugurato caso in cui tale ricerca non dovesse dare esito positivo, si ricorrerà invece ad una ricerca locale "a miglioramenti progressivi" molto simile a quella già illustrata (anche se relativa a vincoli diversi), durante la quale si continuerà a mantenere le $y_j^{CSS\text{-}h}$ e le y_k^{LE} fissate ai valori stabiliti.

6. METODO DI RICERCA DIVERSIFICATION-REFINING-TIGHT REFINING

6.1 Considerazioni generali

Quando si vuole risolvere un problema MIP di dimensioni elevate, la tecnica dell'approccio brutale (in cui si prende il modello così com'è e si cerca di risolverlo in un colpo solo tramite un algoritmo come il branch and cut) non è a volte la più consigliabile. Si deve tenere poi conto che la difficoltà di risoluzione aumenta esponenzialmente all'aumentare delle dimensioni del problema, e che nel caso qui studiato di una rete di interconnessione le dimensioni aumentano a loro volta quadraticamente all'aumentare dei dispositivi della rete: se n è il numero totale di dispositivi, il numero di tutti i possibili collegamenti fra di essi (e dunque anche delle variabili x che ne attestano le attivazioni) è infatti $O(n^2)$.

Invece che un unico, gigantesco studio dell'insieme soluzione preso nella sua interezza, è spesso preferibile una serie di studi più mirati, concentrati ciascuno solo su di una piccola regione (sperabilmente interessante) dell'insieme suddetto. Nel capitolo precedente è stata illustrata la tecnica euristica VNS, la quale, data una generica soluzione corrente, aggiungeva al modello di partenza nuovi vincoli tali che l'insieme delle soluzioni si riduceva ad un piccolo intorno, facilmente esplorabile, della soluzione in questione. Nel caso si fosse trovata nell'intorno una soluzione migliorativa, questa sarebbe diventata la nuova soluzione corrente, e su di essa si sarebbero costruiti i nuovi vincoli (da aggiungere al modello generale) per studiarne il relativo intorno.

Un'altra strada per rendere più maneggevole un problema MIP è quella di ridurne le dimensioni, fissando parte delle sue variabili ad una determinata configurazione di valori. E' ovvio che, così facendo, si risolve solamente un sottoproblema, e che quindi sarà necessario ripetere l'operazione di assegnazione (e soluzione del corrispondente sottosistema) per svariate configurazioni di valori. L'imposizione ad una variabile di un valore determinato è un vincolo molto stringente, ed in tal modo riduce notevolmente l'estensione del relativo insieme di soluzioni, semplificandone così lo studio. Non bisogna però cadere nell'errore di fissare troppe variabili e rendere perciò eccessivamente selettiva ciascuna singola esplorazione, in quanto poi altrimenti bisognerebbe effettuarne un numero eccessivo (se si aumenta il numero di variabili da fissare, si aumentano esponenzialmente le possibili configurazioni di valori per l'assegnazione).

Il nodo cruciale della questione è quindi trovare una metodologia con cui scegliere le variabili da fissare e le configurazioni di valori a cui fissarle. Sicuramente, le variabili che più intuitivamente si prestano a far parte dell'insieme da assegnare sono quelle binarie; sulla loro scelta specifica (ossia quante e quali considerare) influirà comunque in modo fondamentale il tipo di struttura del problema. Nel caso di una rete di interconnessione, la corrispondente struttura suggerisce di "congelare" a dei valori determinati le

variabili binarie relative all'allocazione o meno dei dispositivi potenziali CSS ed LE; in tal modo, il sottoproblema così creato consisterà soltanto nel trovare i collegamenti migliori con cui connettere i vari apparati (i collegamenti saranno infatti l'unico grado di libertà su cui poter agire).

6.2 Applicazione del DRT ad una rete di interconnessione

Il metodo che si vuole ora sviluppare si basa sulla seguente idea: data una generica rete di interconnessione, considerarne varie sue configurazioni (in cui le collocazioni dei dispositivi sono rigorosamente fissate), e per ciascuna di esse collegare i suddetti dispositivi in modo da rispettare i vincoli strutturali e di carico e minimizzarne il costo complessivo. Fra tutte le soluzioni così trovate, si sceglierà naturalmente quella di costo minimo. Con la collocazione forzata degli apparati, si ottiene una notevole semplificazione del modello: vengono infatti fissate non solo (in modo diretto) le variabili $y_j^{CSS\text{-}h}$ (j = 1,...,m; h = 1,2) e y_k^{LE} (k = 1,…,p), ma anche (in modo indiretto ed al valore 0) le variabili x, w e z relative a collegamenti che partono da (o arrivano in) dispositivi le cui y sono fissate a 0. Eliminando tutte queste variabili (in quanto trasformate in semplici coefficienti), si ha perciò una cospicua riduzione del numero di gradi di libertà del sistema di partenza.

6.2.1 Vincoli con coefficienti big-M

Una vantaggiosa conseguenza (oltre a quelle già descritte) che deriva dal fissare a valori determinati le variabili riguardanti la collocazione dei dispositivi è la limitazione dell'effetto negativo dei cosiddetti "vincoli con coefficienti big-M": si tratta di vincoli nei quali una variabile binaria è moltiplicata per un coefficiente di valore molto elevato, e la cui presenza in un modello MIP crea non poche difficoltà di soluzione all'algoritmo branch and cut.

Si osserva che nel modello (0)–(16) proposto in [1] ed adottato nel presente lavoro (Par. 2.2, pag. 18), i vincoli con coefficienti big-M sono (1), (2), (3), (6) e (9), vale a dire i vincoli che impongono la massima capacità di carico (traffico, moduli entranti e BTS afferenti) per ciascuno dei CSS ed LE.

Si consideri ad esempio i vincoli di tipo (1), (2) e (3) relativi ad un generico CSS j, e per semplicità si supponga che il CSS in questione possa essere solamente di tipo simple (il discorso si può facilmente estendere anche al caso generale).

Si avrà così:

(1) $$\sum_{i=1}^{n} T_i^{BTS} x_{ij}^{BTS-CSS} \le T_j^{CSS-1} y_j^{CSS-1} \qquad T_j^{CSS-1} = 350$$

(2) $$\sum_{i=1}^{n} x_{ij}^{BTS-CSS} \le N_j^{CSS-1} y_j^{CSS-1} \qquad N_j^{CSS-1} = 20$$

(3) $$\sum_{i=1}^{n} e_i^{BTS} x_{ij}^{BTS-CSS} \le E_j^{CSS-1} y_j^{CSS-1} \qquad E_j^{CSS-1} = 40$$

Può capitare che, a causa dell'aggiunta di altri gruppi di vincoli al modello di partenza (al fine di limitare l'estensione dell'insieme di esplorazione), il numero di BTS afferenti al CSS j (e quindi anche il traffico ed il numero di moduli entranti) sia sicuramente non superiore ad un valore ben più piccolo del limite massimo tollerabile dal dispositivo.

Ora, è chiaro che l'algoritmo branch and cut, avendo il compito di minimizzare il costo della rete senza considerare i vincoli di interezza sulle variabili, attiverà soltanto quella "frazione" del CSS j che gli sarà strettamente necessaria per sostenere il carico in ingresso (in tal modo, anche il costo di attivazione sarà infatti solo una frazione del costo totale). Supponendo, per fare un esempio numerico, che il numero di BTS afferenti non possa essere superiore a 5, il numero di moduli entranti non più alto di 8 ed il traffico non maggiore di 70 Erlang,sicuramente l'algoritmo assegnerà alla variabile y_j^{CSS-1} un valore non superiore ad ¼ (tanto basta infatti per soddisfare i vincoli (1), (2) e (3)). La soluzione a cui si perverrà sarà dunque non ammissibile, in quanto la y_j^{CSS-1} in essa contenuta risulterà frazionaria, così come risulteranno frazionarie tutte le y relative a dispositivi le cui capacità di carico saranno sfruttate solo in parte. L'algoritmo sarà di conseguenza costretto ad effettuare svariate operazioni di branching ed a sviluppare l'albero molto in profondità prima di poter pervenire a soluzioni soddisfacenti i vincoli di interezza.

Una tecnica certamente utile per contrastare gli effetti dei big-M è quella di inserire nel modello gruppi di vincoli ridondanti quando le condizioni di interezza sono imposte, ma che costituiscono dei tagli assai efficaci durante il processo di soluzione, quando le suddette condizioni vengono rilassate.

Nel caso della variabile y_j^{CSS-1} e dei valori numerici esposti nell'esempio prima presentato, tali vincoli saranno:

$$\sum_{i=1}^{n} T_i^{BTS} x_{ij}^{BTS-CSS} \le 70\, y_j^{CSS-1} \qquad \sum_{i=1}^{n} x_{ij}^{BTS-CSS} \le 5\, y_j^{CSS-1} \qquad \sum_{i=1}^{n} e_i^{BTS} x_{ij}^{BTS-CSS} \le 8\, y_j^{CSS-1}$$

e rappresenteranno un sostanziale rafforzamento dei corrispondenti (1), (2) e (3).

I suddetti vincoli dovranno essere generati per ogni variabile y di collocazione, e dovranno naturalmente essere aggiornati ogniqualvolta sostituzioni o modifiche di altri vincoli cambieranno il numero di BTS, di

moduli e di traffico oltre cui il singolo dispositivo sicuramente non sarà caricato nella situazione corrente.

Nel caso del metodo DRT nella fase di tight refining, l'imposizione alle y di valori determinati rende comunque i vincoli big-M del tutto innocui (essendo y_j^{CSS-1} già fissata a 0 o a 1, non v'è rischio che l'algoritmo le assegni il minimo valore frazionario soddisfacente le correnti esigenze di carico), e dunque lo sviluppo dei vincoli atti a smorzarne gli effetti nocivi diventa in tal modo evidentemente superfluo.

6.2.2 Espressioni e vincoli caratteristici del metodo

Si elencheranno ora le espressioni ed i vincoli caratteristici del DRT, associati a delle brevi spiegazioni sul loro significato.

L'espressione chiave su cui si incentra tutto il metodo è la seguente:

$$\Delta'(y, y^*) := \sum_{j \in Y^*_{CSS}} \left[1 - \sum_{h=1,2} \left(y_j^{CSS-h^*} y_j^{CSS-h} \right) \right] + \sum_{k \in Y^*_{LE}} \left(1 - y_k^{LE} \right) + \sum_{j \notin Y^*_{CSS}} \sum_{h=1,2} y_j^{CSS-h} + \sum_{k \notin Y^*_{LE}} y_k^{LE} \qquad (6.2.2.1)$$

dove l'asterisco indica che il simbolo sottostante si riferisce ad una particolare soluzione ammissibile fissata, e precisamente:

- $y_j^{CSS-h^*}$ è il valore di y_j^{CSS-h} in corrispondenza della soluzione x*, ed esprime il fatto se tale soluzione prevede o meno un CSS di tipo h collocato nella locazione j
- Y^*_{CSS} = { j ∈ {1,…,m} : $y_j^{CSS-1^*}$ = 1 o $y_j^{CSS-2^*}$ = 1 } è l'insieme dei CSS collocati (con un dispositivo simple o complex, non ha importanza quale) nella soluzione x*
- Y^*_{LE} = { k ∈ {1,…,p} : $y_k^{LE^*}$ = 1 } è l'insieme degli LE collocati nella soluzione x*

Il termine Δ'(y, y*) esprime quindi il numero di dispositivi che, nella nuova soluzione, cambiano di stato (tipo o collocazione) rispetto alla particolare soluzione ammissibile fissata x*.

Il vincolo

$$\Delta'(y, y^*) = 0 \qquad (6.2.2.2)$$

impone che la nuova soluzione mantenga i dispositivi collocati esattamente nelle stesse posizioni (e con lo stesso tipo) della soluzione x*.

Se si considera il modello con i vincoli di interezza imposti sulle variabili, il (6.2.2.2) è equivalente ai seguenti:

$$\begin{cases} y_j^{CSS-h} = y_j^{CSS-h^*} \;,\; j = 1,...,m \;;\; h = 1,2 \\ y_k^{LE} = y_k^{LE^*} \;,\; k = 1,...,p \end{cases} \qquad (6.2.2.3)$$

C'è da chiedersi però se non convenga utilizzare l'insieme (6.2.2.3) al posto dell'unico vincolo (6.2.2.2): potrebbe infatti accadere che, con il rilassamento continuo delle variabili intere, i (6.2.2.3) risultino più stringenti, e tornino dunque più utili durante il processo di esplorazione del branching tree.

In realtà, è facile verificare che il vincolo (6.2.2.2) è equivalente ai (6.2.2.3) anche nel caso del rilassamento continuo.

Vi è poi un secondo gruppo di vincoli, il quale verrà utilizzato dall'algoritmo (congiuntamente con il (6.2.2.2)) nel caso la ricerca in un'unica esecuzione dell'ottimo a dispositivi fissati risultasse ancora troppo complessa, e fosse quindi consigliabile procedere con uno studio (a passi successivi) di intorni più ristretti.

Tali vincoli sono i seguenti:

$$\left.\begin{array}{ll} \forall j \in Y_{CSS}^* & \sum_{i \in X^*_{BTS-CSS(j)}} \left(1 - x_{ij}^{BTS-CSS}\right) + \sum_{i \notin X^*_{BTS-CSS(j)}} x_{ij}^{BTS-CSS} \le k_{BC} \\ \forall k \in Y_{LE}^* & \sum_{i \in X^*_{BTS-LE(k)}} \left(1 - x_{ik}^{BTS-LE}\right) + \sum_{i \notin X^*_{BTS-LE(k)}} x_{ik}^{BTS-LE} \le k_{BL} \\ \forall k \in Y_{LE}^* & \sum_{j \in X^*_{CSS-LE(k)}} \left(1 - x_{jk}^{CSS-LE}\right) + \sum_{j \notin X^*_{CSS-LE(k)}} x_{jk}^{CSS-LE} \le k_{CL} \end{array}\right\} \qquad (6.2.2.4)$$

Sempre tenendo conto dei ragionamenti svolti al Par. 5.4.1 e che i primi membri dei (6.2.2.4) sono inclusi nell'espressione (5.4.1.1), si può concludere che:

1. il primo membro del primo gruppo di vincoli esprime, per ciascun CSS j collocato nella soluzione x* (e collocato anche nella nuova soluzione, in virtù del (6.2.2.2)), il numero di BTS che prima afferivano al CSS j e nella nuova soluzione hanno cambiato destinazione, più il numero di BTS che prima afferivano ad un altro dispositivo e che ora si connettono invece al

CSS j: il vincolo impone in pratica che, rispetto alla soluzione x*, non si disconnettano dal CSS j né gli si aggiungano più di k_{BC} BTS.

2. Il primo membro del secondo gruppo di vincoli esprime, per ciascun LE k collocato nella soluzione x* (e collocato anche nella nuova soluzione, sempre in virtù del (6.2.2.2)), il numero di BTS che prima afferivano all'LE k e nella nuova soluzione hanno cambiato destinazione, più il numero di BTS che prima afferivano ad un altro dispositivo e che ora si connettono invece all'LE k: il vincolo impone così che, rispetto alla soluzione x*, non si disconnettano dall'LE k né gli si aggiungano più di k_{BL} BTS.
3. Il primo membro del terzo gruppo di vincoli esprime, per ciascun LE k collocato nella soluzione x* (e collocato anche nella nuova soluzione, grazie alla presenza del (6.2.2.2)), il numero di CSS che prima afferivano all'LE k e nella nuova soluzione hanno cambiato destinazione, più il numero di CSS che prima afferivano ad un altro dispositivo e che ora si connettono invece all'LE k: il vincolo impone perciò che, rispetto alla soluzione x*, non si disconnettano dall'LE k né gli si aggiungano più di k_{CL} CSS.

La contemporanea imposizione di (6.2.2.2) e dei (6.2.2.4) costringe la nuova soluzione non solo a scegliere la stessa configurazione di dispositivi presente in x*, ma anche a costruire la struttura dei collegamenti molto simile a quella in x*. Lo studio di un tale insieme di soluzioni, i cui gradi di libertà risultano così drasticamente ridotti, non dovrebbe a questo punto più dare problemi.

Una volta terminata l'analisi dell'insieme soluzione relativo ad una data configurazione fissata di dispositivi, è necessario che quest'ultima non sia più presa in considerazione dall'algoritmo per le esplorazioni successive. Il vincolo "tabu" destinato a questo scopo è il seguente:

$$\Delta'(y, y^*) \geq 1 \qquad (6.2.2.5)$$

Esso impone che, rispetto alla configurazione appena studiata, la nuova soluzione possieda almeno un dispositivo con stato diverso (nel tipo o nella collocazione).

Per diversificare la ricerca e trovare dunque una nuova configurazione di dispositivi su cui operare l'analisi, si utilizza inoltre il seguente vincolo:

$$k_1 \leq \Delta'(y, y^*) \leq k_2 \qquad (6.2.2.6)$$

che impone che la nuova soluzione dovrà avere non meno di k_1 e non più di k_2 dispositivi con stato diverso (nel tipo o nella collocazione) rispetto alla configurazione precedentemente analizzata.

Si passa ora ad esporre in maniera dettagliata l'algoritmo vero e proprio. Si supponga di avere a disposizione una buona soluzione, (per es. il risultato della diversificazione di una precedente fase di ricerca). Ora, la speranza è che si possa migliorare sensibilmente il costo della suddetta soluzione lasciando inalterate le collocazioni dei dispositivi e semplicemente ridefinendo i collegamenti fra questi ultimi.

Tale fase è stata chiamata "Refining" (raffinamento), e viene implementata con l'aggiunta del vincolo (6.2.2.2) al modello principale; si fornisce il sistema così ottenuto (di dimensione notevolmente ridotta rispetto a quello di partenza) in ingresso a Cplex, e ci si aspetta che questi restituisca in tempi rapidi l'ottimo corrispondente, ossia la soluzione che, tenendo i dispositivi fissati in quelle determinate locazioni, minimizza il costo complessivo.

Se anche in questo caso si presentassero delle difficoltà nell'esplorazione e Cplex non riuscisse ad arrivare all'ottimo (o magari neanche a migliorare la soluzione di partenza in tempi ragionevoli), si applica il cosiddetto "Tight refining", un algoritmo locale di discesa che ad ogni passo studia un piccolo intorno della soluzione corrente: per tutta la durata del tight refining il vincolo (6.2.2.2) viene mantenuto all'interno del modello (dunque si continua ad imporre la collocazione dei dispositivi nelle posizioni fissate). Nel generico passo, si considera la soluzione corrente e, sulla base di quest'ultima, si costruiscono i vincoli (6.2.2.4), i quali impongono che a ciascun CSS collocato non si disconnettano o aggiungano più di k_{BC} BTS, e che a ciascun LE collocato non si disconnettano o aggiungano più di k_{BL} BTS e k_{CL} CSS. Si inizia con i parametri k_{BC}, k_{BL} e k_{CL} fissati ad un valore minimo k'_{min}, in modo da restringere il più possibile l'intorno di analisi. Se dentro a tale intorno Cplex non riesce a trovare una soluzione migliorativa, si espande il campo d'azione della ricerca aumentando il valore dei suddetti parametri, si effettua l'esplorazione del nuovo (e più vasto) intorno creato e così via, fino a che non si troverà una soluzione migliorativa o non si raggiungerà un limite massimo k'_{max} di variazioni per k_{BC}, k_{BL} e k_{CL} (non si può infatti dare all'intorno la possibilità di espandersi all'infinito, perché all'aumentare della sua estensione cresce anche la difficoltà di esplorazione). Nel caso in cui si sia pervenuti ad una soluzione migliorativa, la si pone come nuova soluzione corrente, si eliminano i vecchi vincoli di tight refining e si costruiscono i nuovi vincoli sulla base di quest'ultima, riportando i parametri k_{BC}, k_{BL} e k_{CL} al valore minimo k'_{min} (si ricomincia cioè l'esplorazione dall'intorno di dimensioni minime). Nel secondo caso, ossia se k_{BC}, k_{BL} e k_{CL} sono giunti a valere k'_{max} senza che Cplex sia mai riuscito a trovare una soluzione migliore della corrente, si dichiara conclusa tale fase di ricerca, si eliminano gli ultimi vincoli (6.2.2.4) ed il vincolo (6.2.2.2) di refining dal modello, e si considera la soluzione corrente come la soluzione di costo minimo relativamente all'attuale configurazione fissata di dispositivi.

Una volta concluso il tight refining, si aggiunge al modello (in maniera permanente) il vincolo tabu (6.2.2.5), al fine di non correre il rischio di riesaminare in futuro la configurazione di dispositivi appena analizzata (il vincolo impone, infatti, che rispetto a tale configurazione vi sia almeno un dispositivo avente stato diverso, nel tipo o nella collocazione). Si applica quindi la diversificazione, e si impone (tramite la temporanea aggiunta del vincolo (6.2.2.6)) di trovare una soluzione con una configurazione di

dispositivi avente non meno di k_1 e non più di k_2 stati diversi rispetto alla precedente. Scaduto un assegnato tempo limite, si considera la migliore soluzione a cui si è pervenuti, si "congelano" i dispositivi alle collocazioni di tale soluzione (tramite la costruzione del vincolo (6.2.2.2) sulla base di quest'ultima) e si ripetono le fasi di refining e tight refining sopra esposte.

6.2.3 Pseudocodice dell'algoritmo

Si esporrà ora in maniera più formale l'algoritmo prima descritto, riportando lo pseudocodice utilizzato per implementarlo. Si precisa che con ***best_sol*** e con ***current_sol*** si intendono rispettivamente la soluzione incombente (la miglior soluzione ammissibile via via a disposizione nel corso della ricerca) e la soluzione corrente.

setta i parametri di Cplex per la ricerca generale (modello (0) - (16));

risolvi il modello con Cplex, fermandoti alla prima soluzione ammissibile;

current_sol := prima soluzione ammissibile;

best_sol := *current_sol*;

finchè il tempo di esecuzione non supera il tempo massimo

{

aggiungi al modello il vincolo tabu $\Delta'(y, current_sol) \geq 1$ intorno alla soluzione *current_sol*, al fine di non riesaminare più l'attuale configurazione di dispositivi collocati;

aggiungi al modello il vincolo di diversif. $k_1 \leq \Delta'(y, current_sol) \leq k_2$ intono alla soluzione *current_sol*;

setta i parametri di Cplex per la diversificazione;

risolvi il modello con Cplex, fermandoti allo scadere di un tempo limite predefinito;

current_sol := soluzione ammissibile fornita da Cplex;

rimuovi il vincolo di diversificazione dal modello;

se il tempo di esecuzione supera il tempo max **allora esci** dal ciclo;

commento Refining;

aggiungi al modello il vincolo di refining $\Delta'(y, current_sol) = 0$ intorno alla soluzione *current_sol*, fissando i dispositivi collocati alle rispettive posizioni;

setta i parametri di Cplex per il refining (MIPEmphasis := 0; CutUp: = *best_sol*);

risolvi il modello con Cplex, fermandoti allo scadere di un tempo limite predefinito;

se Cplex non è riuscito a trovare una soluzione ammissibile

poni MIPEmphasis := 1 e risolvi il modello con Cplex, fermandoti allo scadere del tempo limite precedente;

se Cplex ha trovato una soluzione ammissibile

{

current_sol := soluzione ammissibile fornita da Cplex;

se costo(*current_sol*) < costo(*best_sol*)

best_sol := *current_sol;*

}

altrimenti

se non esistono soluzioni migliori di *best_sol*

allora rimuovi il vincolo di refining (6.2.2.2) dal modello e

salta alla fine del ciclo per l'iterazione successiva;

se Cplex ha trovato la soluzione ottima

allora rimuovi il vincolo di refining (6.2.2.2) dal modello e

salta alla fine del ciclo per l'iterazione successiva;

se il tempo di esecuzione supera il tempo max **allora esci** dal ciclo;

commento Tight refining;

k' := K'_{min};

finchè non si è raggiunto l'intorno limite k' = k'_{max} e il tempo di esecuzione non supera il tempo massimo

{

aggiungi al modello i vincoli di tight refining (6.2.2.4) intorno alla soluzione corrente e usando k' (k_{BC}, k_{BL}, k_{CL} := k');

setta i parametri di Cplex per il tight refining;

risolvi il modello con Cplex;

se Cplex ha trovato una soluzione migliorativa

allora considera tale soluzione come nuova soluzione corrente e poni k' := k'_{min};

altrimenti poni k' := k' + 1;

```
            rimuovi i vincoli di tight refining (6.2.2.4) dal modello;
        }

        rimuovi il vincolo di refining (6.2.2.2) dal modello;
        se costo(current_sol) < costo(best_sol)
            best_sol := current_sol;
    }

    visualizza best_sol;
```

6.2.4 Commenti allo pseudocodice

Vi sono alcune caratteristiche dello pseudocodice su cui vale forse la pena di soffermare l'attenzione.

Innanzitutto si nota come in diversi punti del programma (esattamente alla fine della diversificazione, alla fine del refining e ad ogni passo del tight refining) si effettui il controllo del tempo trascorso; una simile precauzione è obbligatoria, se non si vuole correre il rischio di oltrepassare in misura eccessiva il tempo limite totale previsto per l'esecuzione.

Per quanto riguarda la fase di refining, preme sottolineare due aspetti: il primo è che ciascun modello relativo a tale fase viene inizialmente processato da Cplex con il parametro MIPEmphasis posto a 0; se la ricerca non fornisce alcuna soluzione ammissibile entro il tempo limite stabilito, allora l'analisi del sistema viene ripetuta con MIPEmphasis fissato a 1, fornendo maggiori probabilità di incontrare, se non l'ottimo, almeno una buona soluzione ammissibile (che la fase di tight refining provvederà poi a migliorare). Il secondo aspetto è che il limite superiore CutUp viene posto al valore della migliore soluzione ammissibile disponibile; ciò permette al programma, qualora il lower bound del sistema diventasse superiore al valore di CutUp, di terminare l'esecuzione di Cplex e di saltare anche il successivo tight refining, passando direttamente all'analisi di un'altra configurazione di dispositivi collocati. In pratica, non appena ci si accorge che l'attuale configurazione non offre soluzioni ammissibili migliori di quella incombente, la si abbandona senza ulteriori perdite di tempo.

Per quel che riguarda invece la fase di tight refining, si possono fare le seguenti considerazioni. Può accadere che, data una certa configurazione di dispositivi, il refining non riesca nemmeno ad ottenere una soluzione ammissibile, né con MIPEmphasis a 0 né con lo stesso parametro posto a 1. In questo caso, se non si vuole abbandonare la ricerca all'interno della suddetta configurazione (cosa che equivarrebbe a sprecare il tempo dedicato

alla precedente diversificazione ed alle due esecuzioni di refining), l'utilizzo di tale fase di ricerca diventa essenziale: si considera la soluzione corrente (fornita direttamente dalla fase di diversificazione, nonché unica soluzione ammissibile al momento conosciuta relativamente a quelle date collocazioni di dispositivi) e le si applica l'algoritmo locale di discesa già descritto. Anche se non si perverrà all'ottimo della configurazione in questione, tipicamente si riuscirà comunque a migliorare sensibilmente il costo della soluzione di partenza.

Alcune curiosità relative all'esecuzione di un singolo passo del tight refining: il limite superiore CutUp viene tarato sul valore della soluzione corrente e non della miglior soluzione ammissibile conosciuta (come invece accade nella fase di refining): questo perché durante il refining si esplora in un colpo solo tutto l'insieme soluzione relativo alla data configurazione di dispositivi collocati, mentre un generico passo del tight refining ne esplora solo una piccola regione: se nell'intorno della soluzione corrente non esiste una soluzione migliore di quella incombente, non è da escludere che tale miglioramento si possa comunque ottenere con lo studio di un intorno nel corso di un passo successivo. Se però durante la singola iterazione non si riesce a migliorare nemmeno la soluzione corrente, si interrompe allora l'algoritmo di discesa e si dichiara conclusa l'analisi della configurazione di dispositivi assegnata.

Ci si potrebbe poi chiedere come mai, nel tentativo di migliorare la soluzione corrente, si considerino via via intorni di estensioni crescenti e non invece direttamente l'intorno di massima estensione: il problema è che, a mano a mano che l'estensione dell'intorno aumenta, la difficoltà della sua esplorazione cresce di pari passo. Sarebbe dunque opportuno evitare l'esame degli intorni di dimensioni eccessive, a meno che non vi si sia obbligati dal fatto che negli intorni più ristretti non si è trovata una soluzione migliorativa.

Un'ultima considerazione merita il vincolo tabu (6.2.2.5), il quale comporta l'esclusione da ogni esplorazione futura della configurazione di dispositivi collocati a cui è riferito. Ciò si traduce nel fatto che si rende inaccessibile per le ricerche a venire esattamente la regione appena analizzata: poter delimitare in maniera così netta e precisa l'insieme di soluzioni da interdire costituisce un notevole vantaggio, in quanto evita di ricadere in zone già esplorate e, all'opposto, di escludere zone non ancora analizzate (si può dire che la ricerca dell'algoritmo all'interno dell'insieme soluzione complessivo avviene in modo "partizionato", ossia analizzando regioni completamente disgiunte l'una dall'altra). Nel caso del VNS, il settaggio del parametro a secondo membro del vincolo tabu (5.4.1.15) che stabilisce l'estensione dell'insieme da interdire costituiva invece un problema: in quel metodo, non essendo le ricerche effettuate così ben circoscrivibili, sorgeva infatti la difficoltà di una scelta di compromesso per un intorno tabu non troppo esteso ma nemmeno troppo limitato.

6.3 Esempi di test e risultati

Il metodo DRT è stato applicato al test bed di istanze riportato al Par. 2.2 e già utilizzato come banco di prova per i precedenti metodi ai Par. 3.3, 4.4 e 5.5. Prima di esporre i risultati ottenuti, si presentano i parametri utilizzati nelle varie fasi di ricerca (ricerca iniziale basata sul modello (0) – (16) dell'articolo [1], diversificazione, refining e tight refining), i valori ad essi assegnati (a seconda della classe di test) ed una breve giustificazione sul loro uso.

Parametri dell'algoritmo principale

- Tempo limite totale di esecuzione = 6^h ; 8^h ; 10^h ; 12^h (su PC Pentium II / 450 MHz) rispettivamente per le classi 1, 2, 3, 4 (come per le prove precedenti).
- Massimo numero di diversificazioni = 1.000: l'algoritmo tiene un contatore sul numero di diversificazioni effettuate durante l'esecuzione; non appena tale numero supera il limite massimo previsto, il programma termina.
- Limite massimo di memoria = 128 MB
- Indicatore di gestione memoria in sovraccarico = 1: non appena la dimensione dell'albero decisionale in una singola esecuzione di Cplex supera il limite di memoria stabilito, i nodi successivi vengono memorizzati in file di salvataggio.

Parametri per l'iniziale ricerca generale

- Tempo limite = 60' (su PC Pentium II / 450 MHz)
- Massimo numero di nodi = illimitato (5.000.000)
- Massimo numero di soluzioni ammissibili migliorative = 1: non appena viene oltrepassato il tempo limite o viene esplorato il massimo numero di nodi (oppure viene trovato il massimo numero di soluzioni ammissibili migliorative), Cplex termina la propria ricerca.
- MIPEmphasis = 1: durante la ricerca, Cplex dà la precedenza all'ammissibilità più che all'ottimalità.

Parametri per la diversificazione

- Tempo limite = 30’ (su PC Pentium II / 450 MHz)
- Massimo numero di nodi = illimitato (5.000.000)
- Massimo numero di soluzioni ammissibili migliorative = 7: non appena uno di tali limiti viene raggiunto, Cplex termina l’esecuzione in atto.
- MIPEmphasis = 1: essendo gli insiemi soluzione relativi a tale fase di ricerca ancora piuttosto ostici da analizzare, invece di mirare all’ottimo si preferisce ottenere una buona soluzione ammissibile.
- $k_{min} = 1$, $k_{step} = 2$: ciò significa che, data una certa soluzione corrente da diversificare, si cercherà innanzitutto una soluzione appartenente all’intorno di diversif. con estremi $k_1 = 1$ e $k_2 = 3$ ($1 \leq \Delta'(y, y^*) \leq 3$, ossia una soluzione avente non meno di 1 e non più di 3 dispositivi con stato diverso rispetto alla corrente); se la ricerca nel suddetto intorno risulterà vana, si passerà ad esaminare l’intorno di diversificazione con estremi $k_1 = 4$ e $k_2 = 6$, e così via, fino a che una di tali ricerche non metterà a disposizione una soluzione ammissibile (la cui configurazione di dispositivi collocati sarà dunque scelta per le successive fasi di refining e tight refining).

Parametri per il refining

- Tempo limite = 5’ ; 10’ ; 20’ ; 30’ (su PC Pentium II / 450 MHz) rispettivamente per le classi 1, 2, 3, 4.
- Massimo numero di nodi = illimitato (5.000.000)
- MIPEmphasis = 0 / 1 (la seconda assegnazione solo in caso di fallimento della prima).
- CutUp = costo soluzione incombente (miglior soluzione disponibile).

  Visto che durante il refining un numero notevole di variabili è fissato e che dunque la dimensione del modello è sensibilmente ridotta, è lecito impostare la ricerca all’ottenimento dell’ottimo. Presentandosi tuttavia il rischio che l’insieme soluzione si riveli comunque arduo da esplorare, e che la ricerca dia esito negativo per il ritrovamento non solo dell’ottimo, ma anche di una qualsiasi soluzione ammissibile intermedia (in quanto non fra le attuali priorità di Cplex), spesso è utile rieseguire il refining ripetendo la ricerca con MIPEmphasis settato a 1; in tal modo si cercherà di ottenere perlomeno una soluzione di costo accettabile, sulla quale si potrà poi applicare il tight refining per un’eventuale ulteriore miglioramento. Del fatto infine che il CutUp sia fissato al costo della soluzione incombente si è già parlato nel paragrafo precedente: lo scopo è quello di interrompere la ricerca non appena ci si accorga che, con la configurazione dei dispositivi collocati in esame, non v’è alcuna speranza di trovare una soluzione migliore dell’ottima attualmente conosciuta.

Parametri per il tight refining

- Tempo limite = 3' (su PC Pentium II / 450 MHz)
- Massimo numero di nodi = illimitato (5.000.000)
- MIPEmphasis = 0
- EpAGap = 10^{-9}
- EpGap = 10^{-9}
- CutUp = costo soluzione corrente

  I primi due parametri sono i consueti limiti che definiscono il termine della singola esecuzione. I parametri dell'enfasi e della tolleranza sul gap sono settati in maniera tale da orientare Cplex, nella ricerca relativa a ciascun intorno, al raggiungimento dell'ottimo (addirittura dell'ottimo assoluto dell'intorno, e non di una soluzione ammissibile prossima al lower bound). Ciò è consentito dal fatto che le dimensioni dell'intorno sono molto contenute: non solo le collocazioni dei dispositivi sono rigidamente fissate, ma si hanno pure limitate possibilità di scelta nelle riassegnazioni dei collegamenti. Se invece i parametri della tolleranza sul gap non fossero così stringenti, si potrebbe incorrere nella spiacevole situazione in cui, essendovi una sequenza di soluzioni aventi costi progressivamente tendenti all'ottimo assoluto dell'intorno, si effettui una serie di ricerche dove ad ogni passo ci si arresta dopo aver migliorato in modo quasi impercettibile la soluzione corrente. Con EpAGap ed EpGap settati praticamente a zero, questo rischio è perciò scongiurato. Il CutUp è infine assegnato al costo della soluzione corrente in quanto, visto che lo scopo del singolo passo di tale fase di ricerca è proprio quello di migliorare il suddetto costo, si evita la ramificazione dei nodi dell'albero (relativo all'intorno che si sta esplorando) conducenti a soluzioni sicuramente peggiorative.

- $k'_{min} = 1$, $k'_{max} = 4$: data una certa soluzione corrente, se ne esaminerà prima l'intorno di minima estensione, corrispondente a $k_{BC} = k_{BL} = k_{CL} = 1$ (vincoli (6.2.2.4)). In mancanza di soluzioni migliorative, si estenderà via via l'intorno di analisi incrementando di 1 il valore dei tre parametri, fino a che non si otterrà una soluzione di costo inferiore della corrente o non si giungerà alla situazione limite in cui $k_{BC} = k_{BL} = k_{CL} = 4$ (caso in cui si dichiarerà la fine del tight refining ed il raggiungimento del minimo locale).

Come nel metodo VNS, in tutte le varie fasi di esplorazione il parametro relativo al massimo numero di nodi è stato posto ad un valore molto elevato al fine di regolare la durata delle ricerche esclusivamente sulla base dei limiti temporali. L'utente ha comunque facoltà di operare la scelta da lui ritenuta più opportuna, regolando a proprio piacimento i parametri in questione all'interno di un apposito file di input.

I risultati ottenuti (preceduti da un riassunto dei parametri di simulazione più significativi e dei valori ad essi associati) sono i seguenti:

Input : test bed descritto al Par. 1.6

Metodo : Diversification – Refining – Tight refining (DRT)

Tipo hardware : Pentium II 450 MHz, 128 MB RAM

Sistema Operativo : Windows 98

Parametri di ricerca di maggior rilevanza

Algoritmo principale
- Tempo limite tot di esecuzione = 6^h ; 8^h ; 10^h ; 12^h
- Memoria limite (TreLim) = 128 MB
- Indicatore gestione memoria (NodeFileInd) = 1

Iniziale ricerca generale
- Tempo limite (TiLim) = 60'
- Max n° soluzioni migliorative (IntSolLim) = 1
- Indicatore dell'enfasi (MIPEmphasis) = 1

Diversificazione
- Tempo limite (TiLim) = 30'
- Indicatore dell'enfasi (MIPEmphasis) = 1
- $k_{min} = 1$, $k_{step} = 2$

Refining
- Tempo limite (TiLim) = 5' ; 10' ; 20' ; 30'
- Indicatore dell'enfasi (MIPEmphasis) = 0 / 1
- Limite superiore di cutoff (CutUp) = costo sol. incombente

Tight refining
- Tempo limite (TiLim) = 3'
- Indicatore dell'enfasi (MIPEmphasis) = 0
- Tolleranza assoluta (EpAgap) = 10^{-9}
- Tolleranza relativa (EpGap) = 10^{-9}
- Limite superiore di cutoff (CutUp) = costo sol. corrente
- $k'_{min} = 1$, $k'_{max} = 4$

Classi	Esempi	Costo miglior soluzione ammissibile			
		DRT	VNS	Cplex	TS
1	A (50, 4, 2)	**10.760.529**	**10.760.529**	**10.760.529**	**10.760.529**
	B (55, 5, 3)	**11.619.817**	**11.619.817**	**11.619.817**	**11.619.817**
	C (60, 5, 4)	12.398.907	**12.393.173**	12.396.436	12.395.078
	D (65, 6, 4)	**13.376.659**	**13.376.659**	**13.376.659**	13.445.309
2	E (100, 8, 4)	**19.734.790**	19.740.675	**19.734.790**	20.006.714
	F (105, 9, 5)	**21.950.121**	21.951.866	21.952.231	22.151.922
	G (110, 9, 6)	**22.971.252**	22.973.105	22.971.543	23.100.774
	H (115, 10, 6)	**23.977.171**	23.979.111	23.996.482	23.999.151
3	I (150, 12, 6)	**30.088.388**	30.137.934	30.142.363	30.139.211
	L (155, 13, 7)	**31.096.388**	31.137.306	31.098.048	31.152.784
	M (160, 13, 8)	**31.997.972**	32.035.223	32.045.632	32.070.411
	N (165, 14, 8)	**32.833.226**	32.856.421	32.866.855	32.896.514
4	O (200, 20, 10)	**39.182.672**	39.247.134	39.282.420	39.241.845

Tabella 6.3.1: costo delle migliori soluzioni ammissibili fornite dal metodo DRT e confronto con le soluzioni di VNS, Cplex e TS a parità di tempo di esecuzione (il costo minimo di ogni esempio è evidenziato in grassetto)

La tabella evidenzia come il metodo DRT, fra i 4 posti a confronto, sia quello che riesce quasi sempre a trovare la soluzione di costo minimo. Nelle classi 1 e 2 le prestazioni offerte dalle varie tecniche sono piuttosto simili: trattandosi di esempi di dimensioni contenute, le metodologie sono tutte in grado di fornire l'ottimo assoluto o quasi (a parte qualche difficoltà evidenziata dalla Tabu Search nella classe 2).

Passando però ad esempi di dimensioni maggiori (vale a dire a quelli delle classi 3 e 4), la maggiore efficacia del DRT risulta evidente, in quanto esso riesce ad abbassare sistematicamente di alcune decine di migliaia i costi delle corrispondenti migliori soluzioni fornite dagli altri metodi (fa eccezione l'esempio L in cui Cplex riesce a trovare una soluzione paragonabile).

Per una maggiore chiarezza espositiva, sono stati riportati gli istogrammi relativi al gap dei valori delle soluzioni delle tecniche Tabu Search, Cplex e VNS rispetto ai risultati ottenuti col DRT:

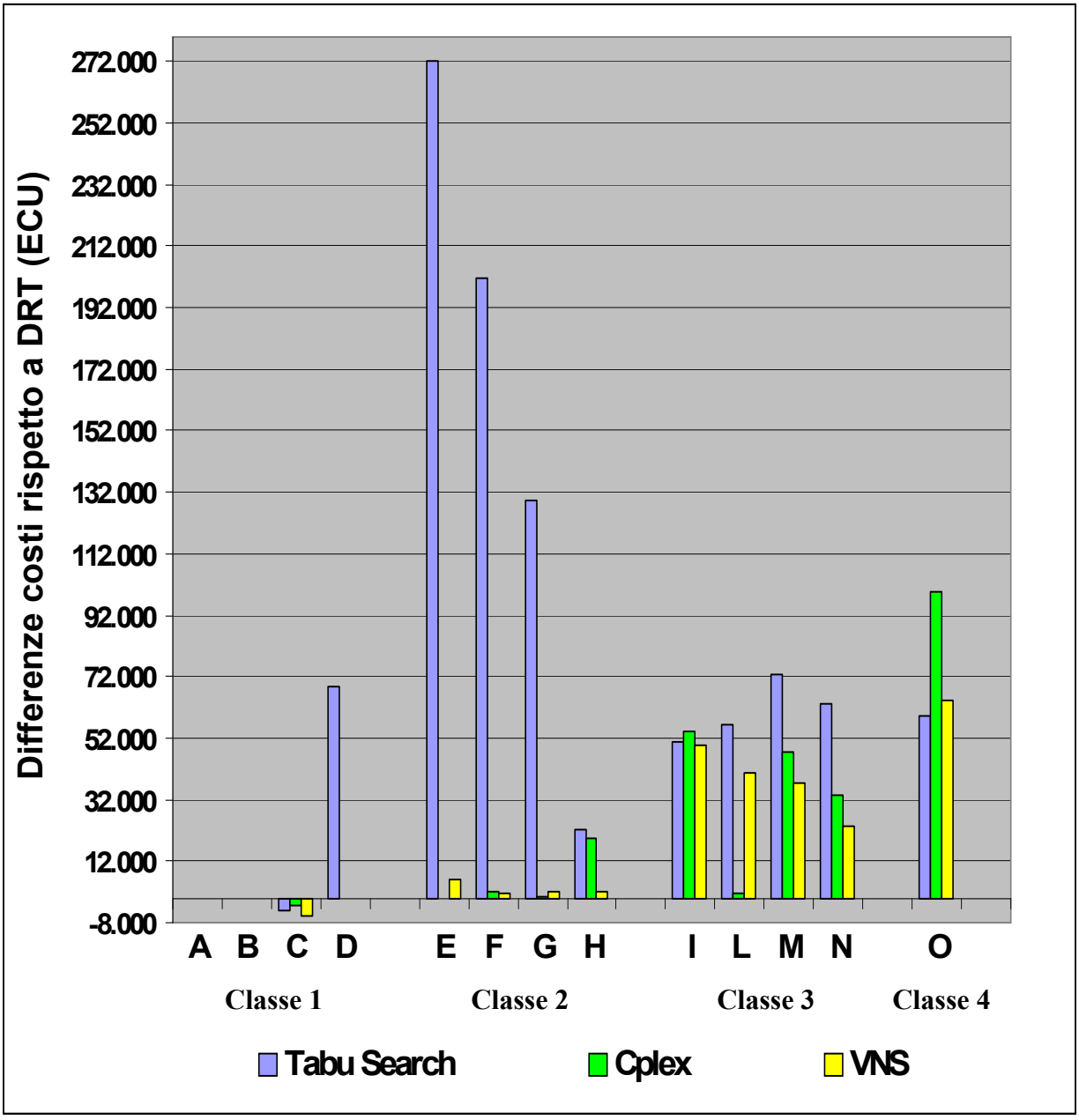

Figura 6.3.1 :differenze di costo di Tabu Search, Cplex e VNS rispetto a DRT

Il grafico ribadisce come nelle prime due classi di esempi non vi siano grosse discordanze di comportamento fra i vari metodi (Tabu Search a parte, la quale si mantiene su livelli di costo decisamente più elevati). Nelle classi 3 e 4 le prestazioni della Tabu Search si uniformano a quelle di Cplex e VNS, ma sono tutte e tre visibilmente peggiori di quelle offerte dal DRT.

Si espone ora una serie di tabelle (relativamente ai soli esempi della classe 3 per ragioni di spazio) in cui si mettono in evidenza i risultati intermedi più significativi ottenuti dal metodo DRT nel corso della singola simulazione:

Esempio I

n°	Diversif	Refining + tight ref
1	30.133.147	30.089.431
2	30.220.061	—
3	30.171.714	—
4	30.144.482	—
5	30.202.173	—
6	30.332.888	30.089.028
7	30.166.267	—
8	30.195.927	—
9	30.165.650	—
10	30.171.267	30.088.635
11	30.146.934	—
12	30.117.035	—
13	30.202.015	—
14	30.241.933	—
15	30.148.108	**30.088.388**
16	30.234.064	—
17	30.140.928	—
18	30.146.896	—
19	30.252.558	—
20	30.176.954	—

Esempio L

n°	Diversif	Refining + tight ref
1	31.151.497	31.147.081
2	31.210.036	31.097.332
3	31.297.249	—
4	31.174.964	31.096.420
5	31.171.982	—
6	31.233.765	—
7	31.169.304	—
8	31.262.967	**31.096.388**
9	31.259.331	—
10	31.253.053	—
11	31.215.877	—
12	31.294.843	—
13	31.209.661	—
14	31.252.576	—
15	31.241.981	—
16	31.249.725	—
17	31.162.847	—
18	31.244.993	—

Tabelle 6.3.2 : risultati intermedi (esiti delle varie fasi di diversificazione, refining e tight refining) delle simulazioni, svolte col metodo DRT, relativamente agli esempi I, L della classe 3 (in ogni tabella è evidenziato in grassetto il costo minimo)

Esempio M

n°	Diversif	Refining + tight ref
1	32.077.123	32.000.778
2	32.110.624	~~
3	32.159.568	~~
4	32.120.698	**31.997.972**
5	32.091.768	~~
6	32.190.660	~~
7	32.288.974	~~
8	32.047.588	~~
9	32.220.031	~~
10	32.172.170	~~
11	32.067.672	~~
12	32.148.885	~~
13	32.139.525	~~
14	32.082.109	~~
15	32.147.473	~~
16	32.080.166	~~
17	32.181.509	~~
18	32.132.152	~~
19	32.139.332	~~

Esempio N

n°	Diversif	Refining + tight ref
1	32.877.576	**32.833.226**
2	32.910.997	32.837.070
3	32.920.690	~~
4	32.947.736	32.842.152
5	32.949.082	~~
6	32.953.508	~~
7	32.918.873	32.843.169
8	32.913.395	~~

Tabelle 6.3.3 : risultati intermedi (esiti delle varie fasi di diversificazione, refining e tight refining) delle simulazioni, svolte col metodo DRT, relativamente agli esempi M, N della classe 3 (in ogni tabella è evidenziato in grassetto il costo minimo)

Le caselle barrate stanno ad indicare le fasi di refining non portate a compimento per il fatto che, nel corso delle corrispondenti ricerche, si è dimostrato che non avrebbero in ogni caso condotto a soluzioni migliorative rispetto all'ottima conosciuta.

La differenza che emerge da un confronto con le tabelle 5.5.2 relative al metodo VNS è che ora per ogni simulazione si compie un maggior numero di diversificazioni, pur essendo la durata di queste ultime pari a 30' in entrambi i metodi, e pur essendo la stessa per entrambi anche la durata complessiva, pari a 10^h, delle esecuzioni. Ciò implica necessariamente che il tempo totale dedicato dal DRT alle fasi di refining e di tight refining è molto minore del tempo totale dedicato dal VNS alle fasi di intensificazione: mentre infatti nel VNS ciascuna intensificazione deve essere condotta fino in fondo (ossia fino all'ottenimento di un minimo locale) in quanto non è dato di sapere in anticipo se tale ricerca riuscirà a migliorare o meno la soluzione incombente, nel DRT ciascun refining può essere abbandonato non appena il lower bound raggiunge il limite CutUp. A conferma di quanto appena asserito, si nota che

la stragrande maggioranza delle fasi di refining eseguite negli esempi I, L, M ed N viene interrotta in questo modo, ed in genere è destinata a durare solo pochi secondi. Inutile dire che tale proprietà pone il DRT in una situazione di netto vantaggio rispetto al VNS, in quanto gli consente di eseguire un ben maggior numero di esplorazioni rispetto a quest'ultimo.

Un'ultima osservazione: si nota che nell'esempio N la soluzione migliore è stata trovata durante la prima fase di refining, ma che nonostante questo alcune fasi di refining e tight refining successive abbiano sfruttato tutto il tempo a loro disposizione per poi offrire delle soluzioni di costo peggiore. Ciò sembra in contraddizione con quanto espresso in precedenza. In realtà, può succedere che tutto il tempo d'analisi destinato ad un dato refining non basti né per trovare una soluzione migliore della incombente né per affermare che una simile soluzione, nell'attuale insieme di ricerca, non esiste (il fatto che il lower bound si mantenga al di sotto del limite CutUp non consente infatti di abbandonare l'esplorazione). Allo scadere del tempo limite per il refining, visto che si nutre ancora la speranza di migliorare l'ottimo corrente, inizia l'esecuzione del tight refining, lo scopo del cui singolo passo è di abbassare il costo non più della soluzione incombente, ma bensì di quella corrente. In queste condizioni è chiaro che soltanto dopo il compimento dell'ultimo passo della ricerca locale si potrà sapere se la speranza di migliorare l'ottimo sarà stata coronata da successo oppure no.

Conclusioni e prospettive future

Nella presente tesi è stato studiato il problema dell'ottimizzazione di una rete di interconnessione relativa a sistemi radiomobili di terza generazione.

Dopo essersi soffermati sulla descrizione matematica del modello atto a rappresentare la rete e dopo aver illustrato l'euristica Tabu Search quale tecnica attualmente in uso per l'ottenimento della soluzione, si sono proposti tre metodi di ricerca alternativi, tutti basati sull'utilizzo del programma Cplex (versione 7.0). Il primo metodo, denominato semplicemente Cplex, è consistito nella banale applicazione del programma omonimo al modello del problema. Il secondo, chiamato VNS, ha operato una ricerca ad intorni variabili durante la quale, per lo studio di ciascun intorno, ci si è appunto serviti di Cplex. Il terzo metodo, detto DRT, ha effettuato un'analisi in cui, di volta in volta, venivano fissate le collocazioni dei dispositivi e, tramite Cplex, si cercava di ottimizzarne i collegamenti.

Messi alla prova da un test bed costituito da 13 istanze di input di varie dimensioni, tutti e tre i metodi presentati hanno esibito prestazioni superiori alla Tabu Search. In particolare l'ultimo metodo, vale a dire il DRT, si è dimostrato in assoluto il più efficiente, soprattutto se applicato a reti con un elevato numero di dispositivi (situazione tipica che verrà a presentarsi nel prossimo futuro).

Il soddisfacente comportamento del DRT e l'idea del tutto generale che sta alla base del suo funzionamento hanno fornito lo spunto per un'ulteriore evoluzione del lavoro, già tuttora in fase di studio: si tratta dell'implementazione di un programma che estenda il metodo DRT alla risoluzione di un generico problema MIP il cui modello contenga variabili di tipo binario di due tipi ("x" ed "y") strutturalmente diversi.

La naturalezza con cui il metodo DRT si presta a tale generalizzazione e l'efficacia da esso dimostrata nell'affrontare un problema difficile come la minimizzazione del costo di una rete di interconnessione alimentano le speranze che il proseguimento del lavoro in questa direzione possa riservare sviluppi interessanti.

APPENDICE A

Presentazione dei file di ingresso

Nella presente appendice è riportato il prospetto dei file di ingresso relativi ai due programmi C++ che implementano i metodi VNS e DRT.

Prima della loro esposizione, se ne riporta l'elenco con una breve descrizione per ciascuno di essi.

_cplexinfo.dat: è utilizzato sia dal VNS che dal DRT, ed è il file che contiene le informazioni generali sulla rete in ingresso, quali il numero di dispositivi, le massime distanze consentite per i collegamenti, i vincoli di capacità su CSS ed LE, i costi degli apparati e delle connessioni, il fattore di scala dell'area geografica e la tecnologia usata per la trasmissione (cavi o microonde).

_coord.dat: è utilizzato sia dal VNS che dal DRT, ed è il file che contiene le coordinate spaziali delle BTS e dei potenziali CSS ed LE.

_BTStraffic.dat: è utilizzato sia dal VNS che dal DRT, ed è il file che contiene l'informazione sul tipo e sul traffico associato ad ogni BTS: il tipo di una BTS può essere basic (0), isolated (1) o free (2), ed indica rispettivamente una connessione obbligatoria ad un (qualsiasi) CSS, una connessione obbligatoria ad un (qualsiasi) LE, oppure una connessione libera ad un (qualsiasi) CSS od LE. Il tipo è da non confondersi con la categoria, la quale indica semplicemente il tipo di dispositivo a cui la BTS è connessa in una data soluzione (il tipo e la categoria comunque coincidono se la BTS non è free).

_cplexpar_VNS.dat: è utilizzato solo dal VNS, ed è il file che contiene il settaggio dei parametri che gestiscono l'uso di Cplex all'interno delle diverse fasi di ricerca (iniziale ricerca generale, diversificazione, intensificazione). Tali parametri decidono il tempo limite di esecuzione complessivo e delle varie fasi, le dimensioni ed i limiti degli intorni di esplorazione (di diversificazione ed intensificazione) e se la ricerca all'interno della singola fase deve essere improntata all'ottimalità o all'ammissibilità.

_cplexpar_DRT.dat: è utilizzato solo dal DRT, ed è il file che contiene il settaggio dei parametri che gestiscono l'uso di Cplex all'interno delle diverse fasi di ricerca (iniziale ricerca generale, diversificazione, refining, tight refining). Come nel _cplexpar_VNS.dat per il caso VNS, tali parametri stabiliscono come deve essere effettuata la ricerca all'interno delle singole fasi (tempi limite, dimensioni degli intorni, precedenza all'ottimalità o all'ammissibilità, ecc.).

Nota: le istanze di file di ingresso esposte nel seguito si riferiscono ad una rete di interconnessione costituita da 40 BTS, 4 CSS e 2 LE.

```
/*****************************************************************/
/*  _cplexinfo.dat - general information in input                */
/*****************************************************************/

#     Number of BTS                                    :      40
#     Number of CSS                                    :       4
#     Number of LE                                     :       2

/*****************************************************************/

#   scenario urban  (0 - 12 Km)

     Maximum distance allowed between BTS-CSS  (m)    :    6000
     Maximum distance allowed between BTS-LE   (m)    :    7500
     Maximum distance allowed between CSS-LE   (m)    :    5000

#   scenario suburban  (12 - 20 Km)

     Maximum distance allowed between BTS-CSS  (m)    :   12000
     Maximum distance allowed between BTS-LE   (m)    :   14000
     Maximum distance allowed between CSS-LE   (m)    :   10000

#   scenario rural  (20 - 60 Km)

     Maximum distance allowed between BTS-CSS  (m)    :   35000
     Maximum distance allowed between BTS-LE   (m)    :   40000
     Maximum distance allowed between CSS-LE   (m)    :   25000

     Number of channels in an elementary module
         * of a cable                                 :      30
         * of MW  ( 0-15 Km)  ( 4x2Mbps)              :     120
         * of MW  ( 0-15 Km)  (16x2Mbps)              :     480
         * of MW  (15-70 Km)  (64x2Mbps)              :    1920

/*****************************************************************/
/*  CONSTRAINTS CONCERNING THE CAPACITY OF CSSs & LEs            */
/*****************************************************************/

     Simple CSS   * Handled traffic  (Erl)            :     350
                  * Linkable BTSs                     :      20
                  * 2Mbit/s PCM modules IN/OUT        :      40

     Complex CSS  * Handled traffic  (Erl)            :     600
                  * Linkable BTSs                     :      30
                  * 2Mbit/s PCM modules IN/OUT        :      60

     LE           * Handled traffic  (Erl)            :    3000
                  * 2Mbit/s PCM modules IN/OUT        :     300

/*****************************************************************/
/*  COST OF LINKS & DEVICES                                      */
/*****************************************************************/

     CABLE (x module=2Mbps) INSIDE CITIES
        Connection fee per two ends      (ECU)       :    1454
      Fixed monthly charge per two ends  (ECU)       :     764
      Variable monthly charge per 100m   (ECU)       :       2.9

     CABLE (x module=2Mbps) ON LONG DISTANCES (>20 Km)
        Connection fee per two ends      (ECU)       :    2254
      Fixed monthly charge per two ends  (ECU)       :     764
```

```
      Variable monthly charge depending on
      the distance between the two ends
                  * from   0 to   35Km  (ECU/Km)     :       52
                  * from  35 to   70Km  (ECU/Km)     :       40
                  * from  70 to  150Km  (ECU/Km)     :       12
                  * from 150 and above  (ECU/Km)     :        7.2

     MICROWAVE
          * from  0 to 15Km  (ECU)  ( 4x2Mbps)       :    24243
          * from  0 to 15Km  (ECU)  (16x2Mbps)       :    29092
          * from 15 to 70Km  (ECU)  (64x2Mbps)       :   334000

     basic BTS     (ECU)                            :   100000
     isolated BTS  (ECU)                            :   150000
     simple CSS    (ECU)                            :   450000
     complex CSS   (ECU)                            :   600000
     LE            (ECU)                            :  1400000

     MICROWAVE      (0)   OR   CABLE      (1)        :        1

/*****************************************************************/

#     Loss Probability in the Erlang-B               :        0.02

#     Scale Factor                                   :       12
```

```
/*****************************************************************/
/* _coord.dat - information about devices' coordinates           */
/*****************************************************************/


******* BTS *******

BTS[0]:  5472 4421
BTS[1]:  6683 7770
BTS[2]:  4418 6928
BTS[3]:  4754 4712
BTS[4]:  8234 6822
BTS[5]:  5103 7067
BTS[6]:  8350 5072
BTS[7]:  6969 6298
BTS[8]:  7543 7442
BTS[9]:  6841 7621
BTS[10]:  7727 6986
BTS[11]:  4562 7883
BTS[12]:  7178 5435
BTS[13]:  5919 7605
BTS[14]:  5965 4539
BTS[15]:  6706 4191
BTS[16]:  9566 9370
BTS[17]:  6323 8594
BTS[18]:  3681 3117
BTS[19]:  2799 5119
BTS[20]:  8882 5889
BTS[21]:  7250 8573
BTS[22]:  5358 3284
BTS[23]:  5816 8873
BTS[24]:  2514 3377
BTS[25]:  4629 9470
BTS[26]:  2935 7419
BTS[27]:  3578 2926
BTS[28]:  1571 5761
BTS[29]:  1144 4607
BTS[30]:  9616 6052
BTS[31]:  8135 10142
BTS[32]:  10904 6954
BTS[33]:  11180 3482
BTS[34]:  923 9036
BTS[35]:  200 1515
BTS[36]:  10648 10169
BTS[37]:  5687 10433
BTS[38]:  8423 84
BTS[39]:  73 1929


******* CSS *******

CSS[0]:  7829 8317
CSS[1]:  4208 6460
CSS[2]:  9263 4140
CSS[3]:  2587 5538


******* LE ********

LE[0]:  5228 4470
LE[1]:  7351 6094
```

```
/*****************************************************************/
/* _BTStraffic.dat - information about type and traffic of BTSs  */
/*****************************************************************/


           TYPE      TRAFFIC

BTS[0]:      0          24
BTS[1]:      0          19
BTS[2]:      2          25
BTS[3]:      2          25
BTS[4]:      2          25
BTS[5]:      0          19
BTS[6]:      0          15
BTS[7]:      2          22
BTS[8]:      2          20
BTS[9]:      0          24
BTS[10]:     2          25
BTS[11]:     2          18
BTS[12]:     0          17
BTS[13]:     0          20
BTS[14]:     0          23
BTS[15]:     2          25
BTS[16]:     2          35
BTS[17]:     2          26
BTS[18]:     0          37
BTS[19]:     2          48
BTS[20]:     2          38
BTS[21]:     2          42
BTS[22]:     2          36
BTS[23]:     2          28
BTS[24]:     0          44
BTS[25]:     2          40
BTS[26]:     1          27
BTS[27]:     0          45
BTS[28]:     2          32
BTS[29]:     1          28
BTS[30]:     2          30
BTS[31]:     1          36
BTS[32]:     1          37
BTS[33]:     1          27
BTS[34]:     1          39
BTS[35]:     2          32
BTS[36]:     2          32
BTS[37]:     2          31
BTS[38]:     1          31
BTS[39]:     1          27
```

```
/*****************************************************************/
/* _cplexpar_VNS.dat - general information on VNS Cplex parameters*/
/*****************************************************************/

      Total time limit (hours)                              :       6

      Maximum number of diversifications                    :    1000

      # Global algorithm Cplex parameters

            Tree memory limit                               :     128

            Node storage file indicator                     :       1

      # Parameters for initial general research

            Time limit (minutes)                            :      60

            Maximum number of nodes                         : 5000000

            Precedence to optimality (0) or feasibility (1) :       1

            Which number of admiss solutions before stop    :       1

      # Parameters for diversification (shaking)

            Time limit (minutes)                            :      30

            Maximum number of nodes                         : 5000000

            Precedence to optimality (0) or feasibility (1) :       1

            Which number of admiss solutions before stop    :       7

            Minimum number of changes in network status     :      50

            Maximum number of changes in network status     :     200

            Step increment                                  :      50

      # Parameters for intensification (local research)

            Time limit (minutes)                            :       3

            Maximum number of nodes                         : 5000000

            Precedence to optimality (0) or feasibility (1) :       0

            Abs tolerance on gap between best sol and LB    :  1e-009

            Rel tolerance on gap between best sol and LB    :  1e-009

            Minimum number of changes in network status     :       2

            Maximum number of changes in network status     :       6

            Step increment                                  :       2
```

```
/*****************************************************************/
/* _cplexpar_DRT.dat - general information on DRT Cplex parameters*/
/*****************************************************************/

      Total time limit (hours)                                :       6

      Maximum number of diversifications                      :    1000

      # Global algorithm Cplex parameters

            Tree memory limit                                 :     128

            Node storage file indicator                       :       1

      # Parameters for initial general research

            Time limit (minutes)                              :      60

            Maximum number of nodes                           : 5000000

            Precedence to optimality (0) or feasibility (1)   :       1

            Which number of admiss solutions before stop      :       1

      # Parameters for diversification

            Time limit (minutes)                              :      30

            Maximum number of nodes                           : 5000000

            Minimum numb of devices' changes in net status    :       1

            Step increment                                    :       2

      # Parameters for refining

            Time limit (minutes)                              :       5

            Maximum number of nodes                           : 5000000

      # Parameters for tight refining

            Time limit (minutes)                              :       5

            Maximum number of nodes                           : 5000000

            Abs tolerance on gap between best sol and LB      :  1e-009

            Rel tolerance on gap between best sol and LB      :  1e-009

            Minimum number of links' changes per device       :       1

            Maximum number of links' changes per device       :       4
```

APPENDICE B

Listato del codice sorgente

Qui di seguito sono riportate le parti di codice più importanti dei programmi C++ che implementano i metodi VNS e DRT. Si tratta di:

NOCPLEX_VNS_DRT.h: è l'header file che raccoglie la struttura dati, vale a dire la definizione delle classi, dei tipi e degli identificatori globali utilizzati poi dal programma vero e proprio; tale file è utilizzato sia da NOCPLEX_VNS.cpp (file .cpp che implementa l'algoritmo VNS) che da NOCPLEX_DRT.cpp (file .cpp che implementa l'algoritmo DRT).

main (VNS): è la funzione main del programma NOCPLEX_VNS.cpp e che, sulla base della struttura dati definita in NOCPLEX_VNS_DRT.h, realizza l'algoritmo VNS descritto a pag. 48.

main (DRT): è la funzione main del programma NOCPLEX_DRT.cpp e che, sulla base della struttura dati definita in NOCPLEX_VNS_DRT.h, realizza l'algoritmo DRT descritto a pag. 68.

```
/*************************************************************************/
/*  NOCPLEX_VNS_DRT.h - header file for VNS/DRT global variables' and Cplex parameters' */
/*                      definition                                       */
/*************************************************************************/

/*************************************************************************/
/*  DEFINITION DATA STRUCTURE                                            */
/*************************************************************************/

typedef IloArray<IloNumArray> IloNumMatrix;                 /* matrix of values      */
typedef IloArray<IloNumVarArray> IloNumVarMatrix;           /* matrix of variables   */
typedef IloArray<IloRangeArray> IloRangeMatrix;             /* matrix of constraints */

/*************************************************************************/
/*  DEFINITION MODEL PARAMETERS                                          */
/*************************************************************************/

IloEnv env;                     /* environment in which all Cplex parameters will be defined */

IloNumArray T_BTS, d_BTS, e_BTS,            /* traffic, channels and modules of BTSs   */
            f_CSS_simple, f_CSS_complex,    /* costs of CSSs                           */
            T_CSS_simple, T_CSS_complex,    /* maximum load of traffic for CSSs        */
            E_CSS_simple, E_CSS_complex,    /* maximum load of modules for CSSs        */
            N_CSS_simple, N_CSS_complex,    /* maximum load of BTSs linked for CSSs */
            f_LE, T_LE, E_LE;               /* cost and max loads for LEs              */

IloNumMatrix g_BTS_CSS, g_BTS_LE,   /* connection costs for BTSs-CSSs/LEs' links       */
             c_CSS_LE, M, F;        /* connection costs for CSSs-LEs' links and upper */
                                    /* bounds on traffic and number of modules         */

IloInt Q;                           /* maximum number of channels in 1 module          */

/*************************************************************************/
/*  DEFINITION MODEL VARIABLES                                           */
/*************************************************************************/

IloNumVarArray y_CSS_simple, y_CSS_complex, y_LE;       /* variables about CSSs/LEs' */
                                                        /* activation  */
IloNumVarMatrix x_BTS_CSS, x_BTS_LE,                /* variables about links' activation */
                x_CSS_LE, z_CSS_LE, w_CSS_LE;       /* and their traffic and modules     */

/*************************************************************************/
/*  DEFINITION MODEL EXPRESSIONS (for their meaning let's see relative file .cpp)  */
/*************************************************************************/

// General model expressions

IloExpr expr_obj, expr_LEs;

IloExprArray expr_devices_out_BTS, expr_types_CSS, expr_BTSs_in_CSS,
             expr_traffic_in_CSS, expr_channels_in_CSS, expr_modules_in_CSS,
             expr_LEs_out_CSS, expr_traffic_out_CSS, expr_modules_out_CSS,
             expr_traffic_in_LE, expr_modules_in_LE;

// Diversification and intensification model expression (for NOCPLEX_VNS.cpp)

IloExpr expr_change_net;
```

```
// Diversification, refining and tight refining model expressions (for NOCPLEX_DRT.cpp)

IloExpr expr_change_dev;

IloExprArray expr_change_BTSs_in_CSS, expr_change_BTSs_in_LE, expr_change_CSSs_in_LE;

/*****************************************************************************/
/*  DEFINITION MODEL CONSTRAINTS                                             */
/*****************************************************************************/

// General model constraints

IloRange gen_constr_16;

IloRangeArray gen_constr_0, gen_constr_1, gen_constr_2, gen_constr_3, gen_constr_4,
              gen_constr_6, gen_constr_7, gen_constr_9, gen_constr_10,
              gen_constr_11, gen_constr_15;

IloRangeMatrix gen_constr_5, gen_constr_8, gen_constr_12, gen_constr_13, gen_constr_14;

// Diversification and tabu constraints

IloRange div_neigh_constr_1, div_neigh_constr_2, tabu_constr;

// Intensification constraint (for NOCPLEX_VNS.cpp)

IloRange int_neigh_constr;

// Refining and tight refining constraints (for NOCPLEX_DRT.cpp)

IloRange ref_neigh_constr;

IloRangeArray tight_ref_neigh_constr_0, tight_ref_neigh_constr_1, tight_ref_neigh_constr_2;

/*****************************************************************************/
/*  INTERNAL CONSTANTS, VARIABLES AND CPLEX PARAMETERS                       */
/*****************************************************************************/

IloInt i, j, k,
       m, n, p,                          /* number of CSSs, BTSs and LEs of the network */
       tot_tilim,                        /* maximum time of execution (in hours)        */
       max_num_div,                      /* maximum number of diversifications          */
       k_min_div, k_max_div, k_step_div, /* parameters for diversification research     */
       k_min_int, k_max_int, k_step_int, /* parameters for VNS intensif research        */
       k_min_tight_ref, k_max_tight_ref; /* parameters for DRT tight ref research       */

const char *FILE_FINAL_NET = "_finalnet.dat";          /* output file for best net display */
ofstream Out_finalnet;

// Global algorithm Cplex parameters

IloInt tree_memory_limit,                          /* memory limit for branching tree  */
       node_stor_file_ind;                 /* when memory limit is reached from branching tree,*/
                                           /* it decides to stop execution (0) or not (1,2,3)  */
// Initial general research Cplex parameters

IloInt tilim_gen,                  /* maximum time of execution (in minutes)                */
       max_num_nodes_gen,          /* maximum number of nodes Cplex can explore before stop */
       which_MIPEmph_gen,          /* (0) precedence to optimality, (1) to feasibility      */
       which_start_sol;            /* number of admiss solutions Cplex can find before stop */
```

```
// Diversification research Cplex parameters

IloInt tilim_div,                   /* maximum time of execution (in minutes)                */
       max_num_nodes_div,           /* maximum number of nodes Cplex can explore before stop */
       which_MIPEmph_div,           /* (0) precedence to optimality, (1) to feasibility      */
       which_div_sol;               /* number of admiss solutions Cplex can find before stop */

// Intensification research Cplex parameters (for NOCPLEX_VNS.cpp)

IloInt tilim_int,                   /* maximum time of execution (in minutes)                */
       max_num_nodes_int,           /* maximum number of nodes Cplex can explore before stop */
       which_MIPEmph_int;           /* (0) precedence to optimality, (1) to feasibility      */

IloNum abs_tolerance_int, rel_tolerance_int;        /* how much best solution must   */
                                                    /* be close to the real minimum  */

// Refining research Cplex parameters (for NOCPLEX_DRT.cpp)

IloInt tilim_ref,                   /* maximum time of execution (in minutes)                */
       max_num_nodes_ref;           /* maximum number of nodes Cplex can explore before stop */

// Tight refining research Cplex parameters (for NOCPLEX_DRT.cpp)

IloInt tilim_tight_ref,             /* maximum time of execution (in minutes)                */
       max_num_nodes_tight_ref;     /* max num of nodes Cplex can explore before stop        */

IloNum abs_tolerance_tight_ref, rel_tolerance_tight_ref;    /* how much best solution    */
                                                            /* must be close to the real minimum*/

/*************************************************************************/
/*  DEFINITION TYPES, CLASSES AND OVERLOADED OPERATORS                   */
/*************************************************************************/

                                                    /* for both VNS and DRT */
enum Type_Research { general, diversif, intensif, refining, tight_refining };


class Solution_Values               /* an object of this class contains all information about  */
{                                   /* a Cplex solution (network cost, devices located, etc.) */
        public:

                // Default constructor

                Solution_Values()
                {
                        y_CSS_simple = IloNumArray(env, m);
                        y_CSS_complex = IloNumArray(env, m);
                        traffic_CSS = IloNumArray(env, m);
                        modules_in_CSS = IloNumArray(env, m);
                        modules_out_CSS = IloNumArray(env, m);
                        BTSs_in_CSS = IloNumArray(env, m);

                        y_LE = IloNumArray(env, p);
                        traffic_in_LE = IloNumArray(env, p);
                        modules_in_LE = IloNumArray(env, p);

                        x_BTS_CSS = IloNumMatrix(env, n);
                        x_BTS_LE = IloNumMatrix(env, n);
```

```
                        for (i=0; i<n; i++)
                        {
                                x_BTS_CSS[i] = IloNumArray(env, m);
                                x_BTS_LE[i] = IloNumArray(env, p);
                        }

                        x_CSS_LE = IloNumMatrix(env, m);
                        z_CSS_LE = IloNumMatrix(env, m);
                        w_CSS_LE = IloNumMatrix(env, m);

                        for (j=0; j<m; j++)
                        {
                                x_CSS_LE[j] = IloNumArray(env, p);
                                z_CSS_LE[j] = IloNumArray(env, p);
                                w_CSS_LE[j] = IloNumArray(env, p);
                        }
                }

                // Data members

                IloAlgorithm::Status status;

                IloNum gap;

                IloInt obj, lower_bound, num_nodes,
                        Num_CSS_Loc, Num_LE_Loc;

                IloNumArray y_CSS_simple, y_CSS_complex, traffic_CSS,
                                modules_in_CSS, modules_out_CSS, BTSs_in_CSS,
                                y_LE, traffic_in_LE, modules_in_LE;

                IloNumMatrix x_BTS_CSS, x_BTS_LE,
                                x_CSS_LE, z_CSS_LE, w_CSS_LE;
};


class Converted_Time                    /* an object of this class contains a time expressed    */
{                                       /* in hours, minutes, seconds                           */
        public:
                IloInt hours, minutes, seconds;
};


ostream& operator <<(ostream& os, const Converted_Time& converted_time)
{
        os << converted_time.hours << " h  "
           << converted_time.minutes << " m  "
           << converted_time.seconds << " s";

        return os;
}
```

```
/*************************************************************************/
/*  main function of VNS  method  (NOCPLEX_VNS.cpp)                      */
/*************************************************************************/

int main()
{
        // Definition main function variables

        IloInt num_div,                         /* counter of number of diversifications      */
               k1_div, k2_div,      /* min/max numb of net changes during a diversif research  */
               k_int;               /* number of network changes during an intensif research   */

        Solution_Values start_min, best_min, new_local_min, current_sol;
                                /* start min is local min from which diversification starts    */
                               /* best_min is the incumbent solution (best solution found)    */
                               /* new_local_min is local min obtained from last loc research  */
                               /* current_sol is current solution                             */

        IloModel model(env);
        IloCplex cplex(env);

        // Initialisation network features

        init_par();                                     /* read parameters from input files    */
        init_matrix_cost();                             /* build connection costs' matrices    */

        // Initialisation output files

        Out_finalnet.open(FILE_FINAL_NET);              /* output file for best net recording  */
        if (!Out_finalnet)
        {
                cout << "\nCannot open output file " << FILE_FINAL_NET << ".\n"
                     << "\nError in NOCPLEX_VNS.cpp (main). Execution terminated! \n\n";
                exit(1);
        }

        env.out().add("_out.dat");                      /* output file for intermediate solutions      */

        try
        {
                // Init model coefficients, variables, constraints, objective function and output file

                init_model_coefficients_e_variables();
                init_expressions_for_general_model();
                add_constraints_for_general_model(model);       /* model of general problem  */

                // Searching for start solution

                cplex.extract(model);
                cplex.exportModel("_model.lp");             /* output file for model recording */
                set_cplex_parameters(cplex, general);
                env.out() << "Start solution:\n\n";
                if (! cplex.solve())                            /* solve initial general model  */
                {
                        env.out() << "Failed research start solution\n\n"
                                  << "Total time elapsed = " << convert_time(env.getTime())
                                  << "\n\n\n\n";
                        throw(-1);
                }
```

```
start_min = best_min = get_values(cplex);                    /* starting solution   */
display_solution_values(start_min);

                    /* consider nearest diversification neighbourhood to start_min */
k1_div = k_min_div;  k2_div = k1_div + k_step_div;
num_div = 0;

              /* while time is not finished and num of diversifications is not max   */
while ( (env.getTime() < tot_tilim*3600) && (num_div < max_num_div) )
{
          // Diversification (searching for a solution away from start_min)

          if (start_min.obj == best_min.obj)
                    env.out() << "Diversification of best_min using neighbourhood "
                              << "with k1_div = " << k1_div << ", k2_div = "
                              << k2_div << "\n\n";
          else
                    env.out() << "Diversification of start_min using neighbourhood "
                              << "with k1_div = " << k1_div << ", k2_div = "
                              << k2_div << "\n\n";
          init_expression_for_change_network(start_min);
          add_constraint_for_div_neigh(model, k1_div, k2_div); /*diversif model*/
          set_cplex_parameters(cplex, diversif);
          if (! cplex.solve())                              /* solve diversif model */
          {
                    env.out() << "Failed diversification using neighbourhood with "
                              << "k1_div = " << k1_div << ", k2_div = " << k2_div
                              << "\n\n"
                              << "Increase neighbourhood distance\n\n"
                              << "Total time elapsed = "
                              << convert_time(env.getTime()) << "\n\n\n\n";
                    remove_constraint_for_div_neigh(model);

                    if (k1_div < k_max_div)        /* if current div neigh is not the  */
                                        /* farthest from start_min, increase distance */
                    {

                              k1_div += k_step_div;  k2_div = k1_div + k_step_div;
                    }
                    else
                    {         /* change start_min and consider nearest div neigh to it*/

                              env.out() << "Reached neighbourhood maximum "
                                        << "distance\n"
                                        << "Change start minimum generating "
                                        << "neighbourhoods to the last local minimum "
                                        << "found\n\n\n\n";
                              start_min = new_local_min;          /* start_min is now */
                                                                  /*    new_local_min */
                              init_expression_for_change_network(start_min);
                              add_constraint_for_tabu(model);   /* no more visits for */
                                                                /*start_min            */

                              k1_div = k_min_div;  k2_div = k1_div + k_step_div;
                                        /* consider nearest div neigh to new start_min  */

                              num_div++;
                    }
                    continue;
          }
```

```
        /* diversification has found a solution from which start loc research */
current_sol = get_values(cplex);
env.out() << "Solution inside neighbourhood with k1_div = " << k1_div
          << ", k2_div = " << k2_div << "\n\n";
display_solution_values(current_sol);

// Decreasing research for new local minimum

env.out() << "Decreasing research for a possible new best minimum\n\n";
k_int = k_min_int;      /* consider nearest int neigh to current solution  */
do          /* while farthest neigh is not reached and time is not finished  */
{
        init_expression_for_change_network(current_sol);
        add_constraint_for_int_neigh(model, k_int);  /*intensif model*/
        set_cplex_parameters(cplex, intensif, current_sol);
        env.out() << "Try decreasing research using neighbourhood with "
                  << "k_int = " << k_int << "\n\n";

        if ((cplex.solve()) && (round(cplex.getObjValue()) < current_sol.obj))
        {    /* intensif has found a solution better than the current one */

                current_sol = get_values(cplex);  /* this sol becomes */
                                                  /* the current one    */
                k_int = k_min_int;     /* consider nearest int neigh to */
                                                  /* current sol */
                env.out() << "New improving solution in decreasing "
                          << "research:\n\n";
                display_solution_values(current_sol);
        }
        else
        {           /*int neigh explored has not found solutions better  */
                                                  /* than current */

                env.out() << "Failed decreasing research using "
                          << "neighbourhood with k_int = " << k_int
                          << "\n\n"  << "Total time elapsed = "
                          << convert_time(env.getTime()) << "\n\n";
                k_int += k_step_int;          /* increase neighbourhood   */
                                                  /* distance  */
        }

        remove_constraint_for_int_neigh(model);

}
while ( (k_int <= k_max_int) && (env.getTime() < tot_tilim*3600) );


                                          /* a new local minimum was found  */
env.out() << "No more improvements. End of decreasing research\n\n";
remove_constraint_for_div_neigh(model);
new_local_min = current_sol;
env.out() << "New local minimum:\n\n"
          << "Objective value = " << new_local_min.obj << "\n\n"
          << "Tot CSSs located = " << new_local_min.Num_CSS_Loc
          << endl  << "Tot LEs located = "
          << new_local_min.Num_LE_Loc << "\n\n"
          << "Total time elapsed = " << convert_time(env.getTime())
          << "\n\n\n\n";
```

```
                    // Verify if new local minimum is better than the actual best one

                    if (new_local_min.obj < best_min.obj)   /*if local min is better than best*/
                    {           /* consider this local min as new best and as start for next div*/

                            env.out() << "New local minimum is the best one found so far\n\n\n\n";
                            start_min = best_min = new_local_min;       /* update best_min */
                                                                        /* and  start_min to new_local_min */
                            init_expression_for_change_network(start_min);
                            add_constraint_for_tabu(model);             /* no more visits for */
                                                                        /* this start_min       */

                            k1_div = k_min_div;  k2_div = k1_div + k_step_div;  /* nearest */
                                                /* div neighbourhood to the new best solution  */
                            num_div++;
                    }

                    else                                /* local minimum is not better than best  */

                            if (k1_div < k_max_div)    /* if furthest div neigh is not reached, */
                            {                           /* increase div neigh distance from start_min */

                                    env.out() << "New local minimum doesn't improve the "
                                              << "best one found so far\n\n\n\n";
                                    k1_div += k_step_div;  k2_div = k1_div + k_step_div;
                            }
                            else        /* local minimum doesn't improve best solution, and */
                                        /* furthest diversif neighbourhood from start_min     */
                            {           /* has been reached                                   */

                                    env.out() << "Reached neighbourhood maximum dimension\n"
                                              << "Change solution generating neighbourhoods "
                                              << "to the new local minimum found\n\n\n\n";

                                    start_min = new_local_min;          /* start_min is now */
                                                                        /* new_local_min    */
                                    init_expression_for_change_network(start_min);
                                    add_constraint_for_tabu(model);   /* no more visits for */
                                                                      /* start_min          */

                                    k1_div = k_min_div;  k2_div = k1_div + k_step_div;
                                              /* consider nearest div neigh to new start_min */
                                    num_div++;
                            }
            }

            env.out() << "Best solution found:\n\n"
                      << "Objective value = " << best_min.obj << "\n\n"
                      << "Tot CSSs located = " << best_min.Num_CSS_Loc << endl
                      << "Tot LEs located = " << best_min.Num_LE_Loc << "\n\n"
                      << "Total time elapsed = " << convert_time(env.getTime()) << "\n\n\n\n";

            display_final_net(best_min);         /* display detailed information about best net  */
    }

    catch (IloException& e)                     /* if something in "try" block has gone wrong  */
    {
            env.out() << "Concert exception caught: " << e << "\n\n\n\n";
    }
```

```
	catch(...)
	{
		env.out() << "Unknown exception caught\n\n\n\n";
	}

	env.end();				/* free environment from all Cplex parameters */

	Out_finalnet.close();

	return 0;
}
```

```
/*************************************************************************/
/*  main function of DRT method  (NOCPLEX_DRT.cpp)                       */
/*************************************************************************/

int main()
{
        // Definition main function variables

        IloInt num_div,         /* counter of number of diversifications                */
               k1_div, k2_div,  /* min/max number of devices' changes during a diversif */
               k_tight_ref;     /* dimension of neighbourhood during tight refining phase */

        IloBool new_sol;        /* it says if Cplex has found a solution (1) or not  (0) */

        Solution_Values current_sol, best_sol;      /* respectively current and best solution */

        IloModel model(env);
        IloCplex cplex(env);

        // Initialisation network features

        init_par();                                 /* read parameters from input files */
        init_matrix_cost();                         /* build connection costs' matrices */

        // Initialisation output files

        Out_finalnet.open(FILE_FINAL_NET);          /* output file for best net recording */
        if (!Out_finalnet)
        {
                cout << "\nCannot open output file " << FILE_FINAL_NET << ".\n"
                     << "\nError in NOCPLEX_DRT.cpp (main). Execution terminated! \n\n";
                exit(1);
        }

        env.out().add("_out.dat");                  /* output file for intermediate solutions */

        try
        {
                // Init model coefficients, variables, constraints, objective function and output file

                init_model_coefficients_e_variables();
                init_expressions_for_general_model();
                add_constraints_for_general_model(model);       /* initial general model */

                // Searching for start solution

                cplex.extract(model);
                cplex.exportModel("_model.lp");         /* output file for model recording */
                set_cplex_parameters(cplex, general);
                env.out() << "Start solution:\n\n";
                if (! cplex.solve())                    /* solve initial general model */
                {
                        env.out() << "Failed research start solution\n\n "
                                  << "Total time elapsed = " << convert_time(env.getTime())
                                  << "\n\n\n\n";
                        throw(-1);
                }

                current_sol = best_sol = get_values(cplex);         /* starting solution */
                display_solution_values(current_sol);
```

```
num_div = 0;

while ( (env.getTime() < tot_tilim*3600) && (num_div < max_num_div) )
{
        // Diversification (searching for a new devices' configuration)

        init_expression_for_change_devices(current_sol);
        add_constraint_for_tabu(model);    /* current devices' configuration will */
                                                        /* never be explored  */
        set_cplex_parameters(cplex, diversif);

            /* consider nearest diversification neighbourhood to current devices'*/
        k1_div = k_min_div;  k2_div = k1_div + k_step_div;    /* configuration */

        do          /* while Cplex doesn't find a solution and time is not finished,*/
        {           /* change diversification neigh (increasing its distance)        */

                env.out() << "Try diversification with k1_div = " << k1_div
                          << " and k2_div = " << k2_div << "\n\n";
                add_constraint_for_div_neigh(model, k1_div, k2_div);  /*div model*/
                if (new_sol = cplex.solve())
                {                                   /* if Cplex has found a solution */

                        current_sol = get_values(cplex);  /*save it in current_sol*/
                        env.out() << "Diversification solution:\n\n";   /* and verify */
                        update_best_solution(current_sol, best_sol); /* if it's better*/
                                                                     /* than best   */
                        display_solution_values(current_sol);
                }
                else
                {     /* if Cplex has not found a solution, increase neigh distance */

                        env.out() << "Failed diversification with k1_div = "
                                  << k1_div << " and k2_div = " << k2_div << "\n\n"
                                 << "Total time elapsed = "
                                  << convert_time(env.getTime()) << "\n\n\n\n";
                        k1_div += k_step_div + 1;  k2_div = k1_div + k_step_div;
                }

                remove_constraint_for_div_neigh(model);
                num_div++;
        }
        while ( (! new_sol) && (env.getTime() < tot_tilim*3600) );

        // Refining (searching for best sol of current fixed devices' configuration)

        if (env.getTime() > tot_tilim*3600)              /* if time is finished, esc */
                break;

        env.out() << "Try refining with MIPEmphasis = 0\n\n";
        init_expression_for_change_devices(current_sol);
        add_constraint_for_ref_neigh(model);                    /* refining model  */
        set_cplex_parameters(cplex, refining, best_sol);
        if (! (new_sol = cplex.solve()))
        {    /* if no solution was found with MIPEmph=0, try with MIPEmph=1 */

                env.out() << "Failed refining with MIPEmphasis = 0\n\n"
                        << "Total time elapsed = "
                        << convert_time(env.getTime()) << "\n\n"
                        << "Try refining with MIPEmphasis = 1\n\n";
```

```
            cplex.setParam(IloCplex::MIPEmphasis, 1);
            if (! (new_sol = cplex.solve()))
            {  /* if no solution was found yet, save only status of research */

                        env.out() << "Failed refining with MIPEmphasis = 1\n\n"
                                  << "Total time elapsed = "
                                  << convert_time(env.getTime()) << "\n\n\n\n";
                        current_sol.status = cplex.getStatus();
            }
}

if (new_sol == 1)
{                                       /* if Cplex has found a solution */

            current_sol = get_values(cplex);    /*save it in current_sol and */
            env.out() << "Refining solution:\n\n";   /* verify if it is better  */
            update_best_solution(current_sol, best_sol);       /*  than best */
            display_solution_values(current_sol);
}
else
            if (current_sol.status == IloAlgorithm::Infeasible)
            {           /* Cplex has not found a solution because best solution */
                        /* (cut upper limit) was better than lower bound           */

                        env.out() << "This refining cannot improve best "
                                  << "solution\n\n\n\n";
                        remove_constraint_for_ref_neigh(model);
                        continue;           /* skip directly to next diversification */
            }

if (current_sol.status == IloAlgorithm::Optimal)
{           /* optimal solution of current devices' configuration has already */
            /* been found, so let's directly skip to next diversification          */

            env.out() << "Refining is already optimal, so there's no need of  "
                        << "tight refining\n\n\n\n";
            remove_constraint_for_ref_neigh(model);
            continue;
}

// Tight refining (searching for best solution of current fixed
// devices' configuration using a decreasing research)

if (env.getTime() > tot_tilim*3600)           /* if time is finished, esc   */
            break;

env.out() << "Tight refining solutions:\n\n";

                        /* consider the smallest tight refining neighbourhood */
k_tight_ref = k_min_tight_ref;                          /* of current solution */

do          /* while largest neigh is not reached and time is not finished   */
{
            env.out() << "Try decreasing research with tight refining "
                        << "dimension = " << k_tight_ref << "\n\n";
            init_expressions_for_tight_ref_neigh(current_sol);

                                                /* tight refining model     */
            add_constraints_for_tight_ref_neigh(model, current_sol, k_tight_ref);
            set_cplex_parameters(cplex, tight_refining, current_sol);
```

```
                        if ((cplex.solve()) && (round(cplex.getObjValue()) < current_sol.obj))
                        {       /* tight ref has found a solution better than the current one */

                                current_sol = get_values(cplex);      /* this sol becomes  */
                                                                      /* the current one   */
                                k_tight_ref = k_min_tight_ref;   /* consider the smallest*/
                                                /* refining  neigh of current solution   */

                                env.out() << "New improving solution in decreasing "
                                          << "research:\n\n";
                                display_solution_values(current_sol);
                        }
                        else        /* tight refining neighbourhood explored has not found */
                        {           /* solutions better than the current one               */

                                env.out() << "Failed decreasing research with tight "
                                          << "refining dimension = " << k_tight_ref << "\n\n"
                                          << "Total time elapsed = "
                                          << convert_time(env.getTime()) << "\n\n";

                                k_tight_ref++;     /* increase neighbourhood dimension */
                        }

                        remove_constraints_for_tight_ref_neigh(model);

                }
                while ((k_tight_ref <= k_max_tight_ref) && (env.getTime() < tot_tilim*3600));

                env.out() << "No more improvements. End of tight refining\n\n";

                env.out() << "Best tight refining solution:\n\n";

                                /* verify if final solution of tight refining is better  */
                update_best_solution(current_sol, best_sol);         /* than the best one */

                env.out() << "Objective value = " << current_sol.obj << "\n\n"
                          << "Tot CSSs located = " << current_sol.Num_CSS_Loc << endl
                          << "Tot LEs located = " << current_sol.Num_LE_Loc << "\n\n"
                          << "Total time elapsed = " << convert_time(env.getTime())
                          << "\n\n\n\n";

                remove_constraint_for_ref_neigh(model);
        }

        env.out() << "Best solution found:\n\n"
                  << "Objective value = " << best_sol.obj << "\n\n"
                  << "Tot CSSs located = " << best_sol.Num_CSS_Loc << endl
                  << "Tot LEs located = " << best_sol.Num_LE_Loc << "\n\n"
                  << "Total time elapsed = " << convert_time(env.getTime()) << "\n\n\n\n";

        display_final_net(best_sol);          /* display detailed information about best net */
}

catch (IloException& e)                       /* if something in "try" block has gone wrong */
{
        env.out() << "Concert exception caught: " << e << "\n\n\n\n";
}
```

```
	catch(...)
	{
		env.out() << "Unknown exception caught\n\n\n\n";
	}

	env.end();				/* free environment from all Cplex parameters  */

	Out_finalnet.close();

	return 0;
}
```

BIBLIOGRAFIA

[1] Fischetti M., Romanin Jacur G., Salazar J.J.G., "Optimization of the Interconnecting network of a UMTS Radio Mobile Telephone System", *European Journal of Operational Research, 2*000-revised 2001.

[2] Laspertini A., "Sistemi di Telefonia Radiomobile di Terza Generazione: Progetto Ottimo di Rete di Interconnessione", Tesi di Laurea, Università di Padova, Italia, 1997.

[3] Voss S., Martello S., Osman I.H., Roucairol C., "Meta-Heuristics Advances and Trends in Local Search Paradigms for Optimization, cap. 30, Hansen P., Mladenovic N., "An Introduction to Variable Neighborhood Search", pp. 433-458, 1999.

[4] ACTS deliverable A016/COF/DS/P/008/al, "Principles and methodologies for set-up of the fixed network infrastructure", AC 016 STORMS, 1996.

[5] Berruto E., Colombo G., Napolitano A., "From GSM to UMTS: a continuous evolution with steps of innovation", Technical Reports, CSELT, 1995.

[6] Berruto E., Eynard C., Napolitano A., "Development trends and migration issues towards Universal Mobile Telecommunication System", Technical Reports, CSELT, 1995.

[7] Boorstyn R.R., Frank H., "Large-scale network topological optimization", IEEE Transactions on Communications, vol. 25, pp. 29-46, 1977.

[8] Buitenwerf E., Colombo G., Mitts H., Wright P., "UMTS: Fixed Network Issues and Design Options", IEEE Personal Communications, vol. 2, n. 1, pp. 30-37, 1995.

[9] Caprara A., Fischetti M., "Branch-and-Cut Algorithms: an Annotated Bibliography", in Annotated Bibliographies in Combinatorial Optimization, M. Dell'Amico, F. Maffioli and S. Martello eds., John Wiley and Sons, pp. 45-63, 1997.

[10] Chardaire P., Lutton J.L., and Sutter A., "Upper and Lower Bounds for the Two-Level Simple Plant Location Problem", Annals of Operations Research, vol. 86, pp. 117-140, 1999.

[11] CEC deliverable R2066/BT/PM2/DS/P/070/b2 part. I, "UMTS System Structure Document", RACE 2066 MONET, 1994.

[12] Dasilva J.S., Fernandes B.E., "The European Research Program for Advanced Mobile Systems", IEEE Personal Communications, vol. 2, n. 1, pp. 14-19, 1995.

[13] Klose, A., “A Lagrangean Relax-and-Cut Approach for the Two-Stage Capacitated Facility Location Problem”, European Journal of Operational Research, vol. 126, n. 2, pp. 185-198, 2000.

[14] ILOG CPLEX, “User’s Manual”, 2000.

[15] Labbè M., Louveaux F.L., “Location Problems”, in Annotated Bibliographies in Combinatorial Optimization, M. Dell’Amico, F. Maffioli and S. Martello eds., John Wiley and Sons, pp. 261-281, 1997.

[16] Lee C.Y., “An algorithm for the design of multi-type concentrator networks”, Journal of the Operational Research Society, vol. 44, pp. 471-482, 1993.

[17] Martello S., Toth P., “Knapsack Problems: Algorithms and Computer Implementations”, Wiley, Chichester, 1990.

[18] Padberg M.W., Rinaldi G., “A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems”, SIAM Review, vol. 33, pp. 60-100, 1991.

[19] Rapeli J., “UMTS: Targets, System Concept and Standardization in a Global Framework”, IEEE Personal Communications, vol. 2, n.1, pp. 20-28, 1995.

[20] Savelsbergh M.P., Nemhauser G.L., “Functinal Description of MINTO, a Mixed Integer Optimizer”, Technical Report, Georgia Institute of Technology, School of Industrial and System Engineering, Atlanta, 1995.

[21] Andreatta G., Mason F., Romanin Jacur G., “Appunti di ottimizzazione su Reti”, Edizioni Libreria Progetto, Padova, 1990.

[22] Battiti R., Protasi M., “Reactive Local Search for the Maximun Clique Problem”, Technical Report, TR-95-052, ICSI, Berkeley, CA, 1995.

[23] Battiti R., “Reactive Search Towards Self-Tuning Heuristics”, Modern Heuristic Search Methods, cap. 4, pp. 61-83, 1996.

[24] Battiti R., Tecchiolli G., “The Reactive Tabu Search”, ORSA Journal on Computing 6(2), pp. 126-140, 1994.

[25] Beccari C., Canavero F., Rossetti U., Valabrega P.,“Saper comunicare cenni di scrittura tecnico scientifica”, Torino (IT), Politecnico di Torino, 1993.

[26] Buttò M., Colombo G., Tofoni T.., Tonietti A., “Ingegneria del traffico nelle reti di telecomunicazioni”, L’Aquila (IT), Scuola Superiore G. Reiss Romoli, 1991.

[27] Glover F., Laguna M., “Modern Heuristic Techniques for Combinatorial Problems”, 1992.

[28] Hertz A., Taillard E., de Werra D., “Local search in combinatorial optimiszation”, J.K. Lenstra Editions, 1992.

[29] Kernighan B.W., Ritchie D.M., “The C programming language 2 ed.”, Englewoods Cliffs (US,NJ), Prentice-Hall, 1988.

[30] Pirkul H., “Efficient algorithms for the concentrator location problem”, Computers and Operations Research, vol. 14, pp. 197-208, 1987.

[31] Wirth N., “Algorithms+data structures=programs”, Englewoods Cliffs (US,NJ), Prentice- Hall, 1976.