UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE
E NATURALI

CORSO DI LAUREA IN MATEMATICA

TESI DI LAUREA

Metodi Esatti di Risoluzione per il Problema di Vehicle Routing con Finestre Temporali.

Relatore: Dott. Carlo Filippi

Laureanda: Marina Malatesta

A.A. 2003 – 2004

A mia madre,
la donna a cui un giorno voglio assomigliare.

Signore, quando ho fame, dammi qualcuno
che ha bisogno di cibo;
quando ho sete, mandami qualcuno che ha bisogno
di una bevanda;
quando ho freddo, mandami qualcuno da scaldare;
quando ho un dispiacere, offrimi qualcuno
da consolare;
quando la mia croce diventa pesante,
fammi condividere la croce di un altro;
quando sono povero, guidami da qualcuno nel bisogno;
quando non ho tempo, dammi qualcuno
che io possa aiutare per qualche momento;
quando sono umiliato, fà che io abbia
qualcuno da lodare;
quando sono scoraggiato, mandami qualcuno
da incoraggiare;
quando ho bisogno della comprensione degli altri,
dammi qualcuno che ha bisogno della mia;
quando ho bisogno che ci si occupi di me,
mandami qualcuno di cui occuparmi;
quando penso solo a me stesso,
attira la mia attenzione su un'altra persona.

Madre Teresa di Calcutta

Indice

Introduzione

Come definito dal *Logistics Management Council* (una associazione di professionisti nella gestione della catena di approvvigionamento), la logistica è "il processo di pianificazione, implementazione e controllo del flusso efficiente, e dello stoccaggio di beni, servizi e relative informazioni dal luogo di origine a quello di consumo allo scopo di conformarsi alla richiesta del cliente".

Con le tendenze di produzione di "lean manufacturing" e di operazioni "just-in-time" (produzione leggera che non si basa su scorte di magazzino), l'azienda di trasporti che risulti in grado di fornire un servizio efficiente e tempestivo consegue un indubbio vantaggio competitivo. Apportando migliorie agli assegnamenti delle rotte ai mezzi utilizzati (assegnamenti di routing), è possibile conseguire significativi risparmi attraverso la diminuzione dei costi di trasporto, la riduzione dei costi di stoccaggio e di inventario, l'eliminazione di penalità dovute a carichi e consegne intempestivi. Il trasporto è infatti il principale fattore competitivo nella catena di approvvigionamento, dato che esso gioca un ruolo molto importante nella logistica, i cui costi totali dipendono direttamente dalle decisioni di trasporto.

In questo contesto, il problema dell'assegnazione delle rotte e la conseguente pianificazione dei percorsi di distribuzione e consegna divengono cruciali nella gestione della flotta di veicoli di una compagnia di trasporti.

Per quanto semplice, un sistema di logistica coinvolge una complessità di passaggi che seguono in pratica il ciclo di produzione di un bene e possono essere riassunti come segue:

1. trasporto delle materie prime dai fornitori ai grossisti;
2. spedizione di queste alla fabbrica per la lavorazione;
3. movimentazione dei prodotti finiti verso i diversi depositi;
4. distribuzione dei beni ai clienti finali.

Le fasi che in maniera un po' approssimativa abbiamo sopra delineato presuppongono tutte una gestione efficiente dei trasporti.

In questo lavoro ci concentreremo sull'ultima fase, cioè su quella di distribuzione di un bene ai clienti finali, che dà luogo a problemi di routing molto ardui, la cui analisi e risoluzione rende necessario l'uso di modelli matematici, astrazioni che permettono di descrivere in termini "matematici" le caratteristiche salienti del problema che si vuole studiare e risolvere.

La costruzione di un modello per un problema reale permette la razionalizzazione sistematica delle criticità ed un più facile controllo delle grandezze coinvolte; questo paradigma, figlio invero della *technè* che ispira il nostro tempo, porta alla concreta possibilità di una diminuzione dei costi per la logistica che si traduce in un aumento, sebbene non automatico, dei margini operativi di profitto.

Possiamo dire che i problemi di routing costituiscono un sottoinsieme dei problemi di logistica con il quale hanno a che fare quasi tutte le compagnie e le organizzazioni. Essi riguardano il trasporto fisico per esempio di materiali, persone o informazioni, problemi riguardanti non solo le compagnie nel settore del trasporto ma ogni azienda che debba affrontare trasporti interni (trasporto di prodotti da un luogo all'altro dell'azienda stessa) o grandi compagnie che debbano gestire un servizio di posta interna.

L'aumentata specializzazione nella lavorazione o fabbricazione di un prodotto e la crescente globalizzazione dell'economia mondiale hanno condotto ad un aumento significativo del bisogno, e di conseguenza dell'importanza, del trasporto e ad una particolare attenzione rivolta al costo del trasporto, che rappresenta certamente una porzione significativa dei costi totali di ogni prodotto e servizio.

In un problema di routing "puro" c'è solamente una *componente geografica*, ma problemi di routing più realistici includono anche un aspetto di scheduling, come ad esempio una *componente temporale*.

I problemi di routing e di scheduling sono dunque elementi importanti nella maggior parte dei sistemi logistici e per questo motivo una larga parte della ricerca si è concentrata sullo sviluppo di metodi atti a produrre buone soluzioni a questi, che sono in realtà complessi problemi di *Ottimizzazione Combinatoria*.

La Ricerca Operativa ha come oggetto lo studio e la messa a punto di metodologie e strumenti quantitativi per la soluzione di problemi decisionali e, al momento attuale, rappresenta uno strumento indispensabile per supportare

ogni tipo di processo decisionale. Tra le classi in cui possono essere suddivisi i modelli di Ricerca Operativa spicca quella dei modelli di Programmazione Matematica.

Il campo della Ricerca Operativa ha approntato negli ultimi decenni numerosi algoritmi e metodi di ottimizzazione, tra cui metodi esatti di Programmazione Matematica, euristiche e metaeuristiche. Molti di questi approcci si sono rivelati ad un tempo efficaci ed efficienti e per questo trovano largo impiego per la risoluzione di larga parte dei problemi logistici odierni.

La Ricerca Operativa risolve i problemi di ottimizzazione con tecniche di Programmazione Lineare Intera, basate sulla rappresentazione matematica del problema mediante un modello di Programmazione Matematica in cui le variabili, che sono numeri interi e spesso solo binari, sono legate da equazioni e disequazioni lineari (che rappresentano i vincoli del problema) e l'obiettivo è quello di ottimizzare (massimizzare o minimizzare) una funzione lineare delle variabili del problema (la funzione obiettivo).

Dall'ottimizzazione dei percorsi di una flotta di veicoli, i quali debbono sottostare ad una serie di vincoli dati, prendono le mosse i cosiddetti *Vehicle Routing Problems* (VRPs). Nucleo centrale di tali problemi è la pianificazione dei percorsi (le rotte) su cui sono disposti i clienti da raggiungere e servire, in modo che risultino minimizzati i costi di routing e di assegnamento dei veicoli ai relativi percorsi.

Più precisamente, il VRP comporta la ricerca di un insieme di rotte per i veicoli di una flotta in modo che ciascun veicolo parta dal deposito e vi ritorni al termine della rotta associata e che allo stesso tempo un dato insieme di clienti sia interamente servito. Ad ogni veicolo deve essere assegnata una rotta che copra un certo numero di clienti in modo che ogni cliente sia visitato esattamente da un veicolo (e quindi esattamente una volta nell'intero processo di distribuzione). Ciascun cliente ha una domanda specifica che deve essere soddisfatta dalla consegna e nessun veicolo può servire più clienti di quanto gli permetta di fare la propria capacità. L'obiettivo può essere quello di minimizzare la distanza totale percorsa dalla flotta o il numero di veicoli usati o una combinazione di questi obiettivi.

Il VRP è il più importante tra i problemi di routing, ma non è certamente l'unico che sia in grado di rappresentare efficacemente problemi reali di trasporto.

Il più semplice problema di routing, e per questo motivo anche il più studia-

to fra tutti, è il problema del commesso viaggiatore, il cosiddetto *Travelling Salesman Problem* (TSP), nel quale un commesso (che può essere visto come un veicolo) deve visitare un certo numero di clienti, ognuno uno sola volta, partendo dal deposito e tornandovi alla fine del viaggio.

Dato un grafo (che rappresenta, ad esempio, una rete di strade) i cui nodi rappresentano i clienti da visitare ed agli archi del quale sono associati dei costi (le distanze tra i nodi congiunti dagli archi), il problema consiste nel cercare la rotta più corta (o quella più economica dal punto di vista dei costi di routing) che parta dal deposito e vi ritorni dopo aver visitato ogni nodo del grafo esattamente una volta. Il grafo contiene anche un nodo che rappresenta il deposito da cui si immagina di far partire il commesso e che si assume come nodo termine del percorso di distribuzione. Il TSP è un problema puramente geografico, nel senso che i vincoli del problema e la funzione obiettivo (che rappresenta la grandezza da ottimizzare) dipendono solamente da una componente geografica.

Nonostante la semplicità di formulazione del problema TSP, trovare il percorso migliore che minimizzi la distanza percorsa o i costi totali e che visiti tutti i nodi del grafo una sola volta, è un problema di difficile risoluzione.

Assumendo di avere a disposizione un certo numero m di commessi, che possono servire un certo insieme di clienti, si ottiene il problema m-TSP, in cui ogni cliente deve essere visitato da un solo (arbitrario) commesso, il quale deve partire dal deposito e farvi ritorno alla fine del proprio percorso. L'obiettivo è quello di minimizzare la spesa complessiva ed anche tale problema risulta puramente geografico.

Il VRP è, in realtà, un m-TSP in cui ad ogni cliente è associata una domanda e nel quale ogni veicolo ha una limitata capacità di carico; per questo motivo viene anche detto *Capacitated Vehicle Routing Problem* (CVRP). Esso non è puramente geografico dato che la domanda può essere vincolante ed è anche più difficile del TSP, tanto che generalmente è possibile trovare la soluzione ottima solo se il numero di clienti da servire è relativamente piccolo.

Il VRP deve la sua importanza anche al fatto che rappresenta il modello di base per una vasta gamma di problemi di routing, ovvero diverse estensioni del problema di base ciascuna delle quali cerca di rappresentare con la miglior approssimazione possibile un determinato problema reale di trasporto.

In questo lavoro ci concentreremo su una variante del VRP, quella che comprende le *finestre temporali*, ovvero il cosiddetto *Vehicle Routing Problem*

with Time Windows, spesso indicato semplicemente con la sua sigla, VRPTW. Tale problema estende quello di base attraverso un vincolo ulteriore: i clienti devono essere serviti all'interno di un certo intervallo cronologico rappresentato dalla finestra temporale.

Questo problema risponde all'esigenza sempre più impellente e realistica di considerare l'aspetto temporale dei problemi di routing, che si risolve nella necessità per le aziende non solo di tagliare il più possibile i costi logistici, ma anche di essere concorrenziali sulla differenziazione del servizio.

Ciò ha prodotto uno sviluppo assai rapido e straordinario della ricerca su problemi di routing e scheduling vincolati sul tempo. La dimensione temporale è stata così incorporata nei VRPs nella forma di vincoli di finestre temporali imposti sui clienti da raggiungere e servire.

Più precisamente, il VRPTW coinvolge una flotta di veicoli che, partendo da un deposito, deve raggiungere e servire un certo numero di clienti, situati in diverse posizioni geografiche, ciascuno dei quali con specifiche domande e finestre temporali all'interno delle quali deve avvenire il servizio. Come nel VRP, ogni cliente deve essere servito esattamente da un veicolo e quindi una sola volta nell'intero processo di distribuzione. Inoltre, ad ogni veicolo deve essere assegnata una rotta che contenga consegne tali da non superare la capacità del veicolo stesso, il quale al termine delle consegne deve tornare al deposito da cui è partito.

Per quanto riguarda il vincolo supplementare, cioè quello temporale, esistono due principali varianti del problema:

- la variante in cui le finestre temporali sono "*hard*", cioè da rispettare rigidamente, nel senso che un veicolo può arrivare in anticipo da un cliente ma in tal caso per poterlo servire deve aspettare l'inizio della relativa finestra temporale e comunque non può mai arrivare dal cliente dopo la fine della finestra;

- la variante in cui le finestre temporali sono "*soft*", cioè possono essere violate al costo però dell'introduzione di un contributo di penalizzazione sulla funzione obiettivo.

In generale, i ricercatori hanno considerato problemi VRPTW con finestre temporali rigide ed una flotta omogenea di veicoli. Tuttavia, con il maturare del campo di ricerca, l'aumentato livello di sofisticazione dei metodi sviluppati

ha permesso di trattare anche flotte eterogenee. Per quanto riguarda la dimensione della flotta, questa è stata fissata a priori oppure assunta libera, cioè da determinare simultaneamente con il migliore insieme di rotte per i veicoli; comunque molti metodi possono attualmente ottimizzare la dimensione della flotta durante il processo.

Sebbene l'introduzione delle finestre temporali nel problema VRP renda ovviamente più difficile la costruzione ed il mantenimento di un insieme realizzabile di rotte, tale estensione premette la specificazione di funzioni obiettivo più realistiche rispetto alla "semplice" minimizzazione della distanza totale percorsa. Infatti, oltre a quest'ultima o al numero di veicoli utilizzati, si possono minimizzare la durata totale del viaggio oppure il costo complessivo coinvolto nel routing e nello scheduling, che consiste in costi fissi di utilizzo dei veicoli ed in costi variabili, come quelli di tempo necessario per gli spostamenti, tempi d'attesa e di carico e scarico.

Nonostante la sua intrinseca complessità, il VRPTW ha ricevuto una considerevole attenzione negli ultimi anni nella comunità di Ricerca Operativa. Prima di tutto perché è ancora uno dei problemi di Ottimizzazione Combinatoria più difficili e quindi rappresenta una grande sfida per ogni ricercatore; in secondo luogo per un aspetto più pratico già introdotto, vale a dire per il fatto che la soluzione di questo problema contribuisce direttamente alla riduzione dei costi nell'importante area della logistica.

Il VRPTW è stato studiato ed affrontato usando molte diverse tecniche che includono sia metodi esatti che euristiche e metaeuristiche. Data la difficoltà di realizzazione di metodi esatti veramente efficienti (in molti casi è necessario un tempo proibitivo per trovare una soluzione ottima ad una istanza del problema), la maggior parte della ricerca si è focalizzata sullo sviluppo di diverse ed efficienti tecniche euristiche in grado di fornire in un tempo ragionevole una buona soluzione al problema (ammissibile, ma non certamente ottima). Tra gli approcci euristici e metaeuristici al VRPTW, numerosi sono quelli basati su *tabu search*, *simulated annealing* ed *algoritmi genetici*, con risultati davvero considerevoli.

Certamente meno numerosi, sono stati sviluppati anche efficienti approcci esatti al VRPTW ed è proprio di tali approcci che ci occuperemo in questo lavoro, facendone una panoramica completa.

Dopo aver presentato un modello di Programmazione Lineare Intera del VRPTW, analizzeremo infatti nel dettaglio gli approcci esatti al problema

elaborati nell'area della Ricerca Operativa a partire dalla fine degli anni '80 fino a quelli più recenti.

Nella nostra trattazione tali approcci saranno divisi in tre categorie generali e studiati in base a tale divisione. Infatti, tra i diversi approcci sviluppati per la risoluzione esatta del VRPTW, si distinguono tre filoni principali che si distinguono l'uno dall'altro per le tecniche di base utilizzate:

- *Programmazione dinamica*;
- *Rilassamento lagrangiano*;
- *Generazione di colonne.*

Mostreremo anche come, al momento, i migliori risultati nell'area della Ricerca Operativa siano dati dagli algoritmi che usano la generazione di colonne.

In tali metodi il problema viene diviso in due problemi: un *master problem*, cioè un problema principale, ed un *subproblem*, cioè un suo sottoproblema. Il master problem è il rilassamento lineare di un *set partitioning problem* e garantisce che ogni cliente sia visitato esattamente una volta; il subproblem è invece un *Shortest Path Problem* (SPP) con vincoli aggiuntivi (i vincoli di capacità e di finestre temporali). Usando il master problem vengono calcolati i costi ridotti per ogni arco e tali costi vengono poi usati nel subproblem al fine di generare delle rotte che partano dal deposito e poi vi tornino. Le migliori tra queste rotte vengono allora "rimandate" al master problem ed entrano a far parte del problema di set partitioning rilassato. Dato che il problema di set partitioning viene rilassato rimuovendo i vincoli di interezza, la soluzione ottenuta raramente è intera e per questo motivo il metodo di generazione di colonne viene inserito in una tecnica di soluzione basata sulla separazione (come un metodo branch-and-bound).

Analizzeremo anche un approccio esatto ibrido di Ricerca Operativa e Programmazione con Vincoli basato sulla generazione di colonne. Tale approccio al VRPTW, come tanti altri sviluppati per altri tipi di problemi, è il risultato di un recente tentativo di integrazione dei due campi nel tentativo di risolvere più efficaciemente difficili problemi di Ottimizzazione Combinatoria.

Gli studi relativi a tale integrazione hanno portato a risultati molto interessanti ed hanno aperto nuove ed affascinanti strade, alternative all'usuale approccio di Ricerca Operativa, sulle quali molto si è lavorato e la ricerca è tuttora molto attiva.

Recentemente derivato dal campo dell'Intelligenza Artificiale, il paradigma di Programmazione con Vincoli fornisce una buona flessibilità nel modellare un problema, che in combinazione con la completa separazione tra il modello e la ricerca, crea uno strumento efficace per la risoluzione di problemi reali. Tuttavia, il metodo di ricerca usuale usato in Programmazione con Vincoli mostra spesso tempi di esecuzione proibitivi per problemi di dimensione ragionevole.

La combinazione di metodi di ottimizzazione derivati dalla Ricerca Operativa, dall'Intelligenza Artificiale e dalla Programmazione con Vincoli sembra così essere fertile, dato che questa area di ricerca mostra vantaggi complementari che potrebbero essere combinati minimizzando gli svantaggi.

Per introdurre l'analisi degli approcci esatti al VRPTW analizzeremo prima i problemi di ottimizzazione e soprattutto quelli di Ottimizzazione Combinatoria, alla cui categoria appartiene il problema in esame. Presenteremo poi i principali metodi esatti di risoluzione in Ricerca Operativa, la teoria alla base della Programmazione con Vincoli e le possibili integrazioni tra i due campi. Concluderemo cercando di valutare gli aspetti positivi e quelli negativi dell'approccio ibrido, l'unico presente in letteratura per il problema in questione, mettendolo a confronto con gli approcci puri basati sulle tecniche esatte di Ricerca Operativa.

Capitolo 1

Problemi di Ottimizzazione Combinatoria e Ricerca Operativa

1.1 Ricerca Operativa

La Ricerca Operativa (comunemente indicata con la sigla OR, dall'inglese *Operations Research*), secondo la definizione data dall'INFORMS (Institute For Operations Research and Management Science) "ha lo scopo di fornire basi razionali al processo decisionale cercando di comprendere e strutturare situazioni complesse ed utilizzare questa comprensione per prevedere il comportamento dei sistemi e migliorare le loro prestazioni. Gran parte di questo lavoro utilizza tecniche analitiche e numeriche per sviluppare e manipolare modelli matematici ed informatici per sistemi organizzativi composti da persone, macchine e procedure ...".

Nella definizione si parla di "sistemi" e di "situazioni" complesse in modo generico. Infatti varie tipologie di sistemi e situazioni possono essere oggetto di studio in Ricerca Operativa: gestione del personale (turnazioni ospedaliere, assegnamenti degli equipaggi ai voli di una compagnia aerea), trasporti e logistica (assegnamenti di carichi e rotte ai veicoli di una compagnia di trasporti, costruzione di un orario ferroviario), produzione (definizione dei tempi di esecuzione di operazioni in un reparto industriale, gestione del magazzino), amministrazione pubblica (valutazione delle prestazioni di scuole, ospedali, pianificazione delle risorse idriche), telecomunicazioni (progetto della rete, asse-

gnamento delle frequenze ad un sistema di telefonia cellulare). Naturalmente l'elenco degli esempi potrebbe continuare a lungo data la vastità dei campi di applicazione della OR.

I problemi affrontati sono tipicamente quelli in cui bisogna prendere decisioni sull'uso di risorse disponibili in quantità limitata in modo da rispettare un insieme assegnato di restrizioni, massimizzando il "beneficio" ottenibile dall'uso delle risorse stesse o minimizzando il "costo" dell'utilizzo stesso. Questi problemi sono anche detti *problemi di ottimizzazione*.

La OR è quindi una disciplina che tratta dello sviluppo e dell'applicazione di metodi scientifici per la risoluzione di problemi di decisione che si presentano in molteplici e diversi settori della vita reale. Il suo scopo è dunque quello di fornire una base scientifica per cercare di analizzare e comprendere situazioni anche con strutture molto complesse e quindi utilizzare queste informazioni per predire il comportamento di un sistema e per migliorare le prestazioni del sistema stesso.

I sistemi studiati in Ricerca Operativa hanno in comune la possibilità di essere modellati in modo matematico a partire da una descrizione il più possibile quantitativa. Inoltre l'indagine non è limitata all'analisi descrittiva e predittiva del sistema. Vi sono infatti molti modelli matematici di fenomeni naturali che ne descrivono e predicono l'evoluzione, ma che non fanno parte dei tipici problemi di OR. In questi è presente anche la possibilità di intervenire sul comportamento del sistema con una scelta opportuna di alcuni parametri. Questa scelta è di fatto una decisione. Siccome sono normalmente disponibili varie decisioni alternative si vuole anche decidere nel modo più soddisfacente ai fini delle prestazioni del sistema.

In Ricerca Operativa si vuole anche pervenire alla soluzione, cioè alla decisione più soddisfacente, per via algoritmica, in modo da poter poi applicare lo stesso algoritmo a molti sistemi, che differiscono tra loro solo per i dati quantitativi ma non per la struttura (le diverse istanze di uno stesso problema). Essenziale è anche che l'algoritmo prescelto abbia tempi ragionevoli di calcolo. A questo fine è doveroso analizzare la complessità computazionale dell'algoritmo progettato o del problema stesso.

L'approccio di OR per risolvere un problema di decisione viene solitamente realizzato attraverso diverse fasi, le quali possono essere schematizzate nel seguente modo:

1. L'analisi della situazione reale, discriminando gli aspetti essenziali da

quelli marginali. Tale fase consiste quindi nell'analisi della struttura del problema per individuare i legami logico-funzionali e gli obiettivi.

2. La traduzione della descrizione del problema reale in un modello matematico, bilanciando l'adeguatezza alla realtà del modello con la sua maneggevolezza risolutiva. In questa fase di costruzione del modello, chiamata anche *formulazione*, si descrivono in termini matematici le caratteristiche principali del problema.

3. L'analisi del modello che prevede la deduzione per via analitica, in riferimento a determinate classi di problemi, di alcune importanti proprietà, tra le quali l'esistenza e l'unicità della soluzione ottima e le condizioni di ottimalità (cioè una caratterizzazione analitica della soluzione ottima).

4. La ricerca della soluzione numerica mediante opportuni algoritmi di calcolo.

5. La valutazione della soluzione ottenuta rispetto alle caratteristiche del problema reale. In questa fase la soluzione viene interpretata dal punto di vista applicativo in modo da evitare che abbia scarso rilievo pratico; in questo caso, le eventuali cause di inaccettabiltà devono essere inglobate nel modello stesso, determinando così un nuovo modello più completo del precedente.

L'approccio risolutivo della OR si può quindi definire un *approccio modellistico* che organizza l'analisi di un problema reale in due fasi cruciali:

1. la rappresentazione del problema attraverso un *modello matematico* che ne astragga gli aspetti essenziali e che schematizzi le interrelazioni esistenti tra i diversi aspetti del fenomeno che si sta studiando;

2. lo sviluppo di *metodi matematici efficienti* (algoritmi di soluzione) per determinare una soluzione ottima del problema o una sua buona approssimazione.

Il termine *modello* è solitamente usato per indicare una struttura appositamente costruita per mettere in evidenza le caratteristiche principali di alcuni oggetti reali. Nella OR non si tratta di modelli concreti, ma piuttosto di *modelli astratti*, cioè *modelli matematici* che usano il simbolismo dell'algebra per

descrivere in modo semplificato, ma sempre rigoroso, uno o più fenomeni del mondo reale.

Tra le classi principali di modelli di OR spicca quella dei *modelli deterministici*, i quali considerano grandezze esatte. Nel seguito verranno analizzati questi modelli che sono di fatto i più usati; in particolare si farà riferimento ai *modelli di Programmazione Matematica*, in cui tutto il sistema sotto esame è descritto per mezzo di relazioni matematiche (o logiche) tra variabili che rappresentano gli elementi del sistema ed è definito esplicitamente un obiettivo da minimizzare o massimizzare. Si noti che in questo contesto il termine "programmazione" è inteso nel senso di "pianificazione" e non di costruzione di programmi per il calcolatore.

Alcune situazioni reali particolarmente semplici sono già state studiate ed i rispettivi modelli costituiscono paradigmi la cui risoluzione è stata oggetto di analisi particolarmente approfondite. Tuttavia la maggior parte dei problemi reali non si lascia inquadrare facilmente in casistiche precostituite e presenta sempre qualche elemento di novità. In questi casi è necessario procedere amalgamando nel miglior modo possibile procedure note con procedure nuove progettate per l'occasione.

In generale, le caratteristiche comuni dei modelli costruiti ed utilizzati in Ricerca Operativa ed i passi che si devono seguire per pervenire al modello stesso sono i seguenti:

- bisogna individuare tutte le grandezze presenti nel problema e tra queste bisogna identificare quali sono valori esterni fuori dal controllo diretto del decisore (solitamente detti *dati*), e quali invece sono valori sotto il controllo diretto del decisore (*grandezze decisionali*). Questi ultimi vanno suddivisi in valori che il decisore può fissare direttamente e soggettivamente (*parametri decisionali*) e valori da determinare in base al modello (*variabili decisionali*);

- i dati, i parametri e le variabili decisionali non sono mai grandezze indipendenti, ma sono sempre legate da vincoli che dipendono dalla struttura stessa del problema (*vincoli strutturali*). Bisogna quindi identificare con precisione questi vincoli;

- decisioni alternative e compatibili con i vincoli non sono quasi mai equivalenti per un decisore. Bisogna quindi esplicitare le relazioni di preferenza

fra le decisioni e da queste bisogna individuare uno o più *obiettivi* che il decisore desidera perseguire, sotto forma di funzioni delle grandezze del problema da minimizzare oppure massimizzare (*funzioni obiettivo*);

- per raggiungere certi obiettivi è spesso utile introdurre ulteriori vincoli che non sono strutturali, non dipendono dalla natura stessa del problema ma dall'intenzione del decisore di indirizzare la decisione in una determinata direzione. Tali vincoli non devono essere necessariamente rispettati in ogni circostanza, sono perciò indicati come *vincoli flessibili*;

- una volta trovata una soluzione (cioè i valori delle variabili decisionali) è indispensabile analizzarla alla luce del problema reale e nel caso essa non sia valida occorre rivedere il modello.

Va subito detto che in questo schema vi sono ampi margini di incertezza, di approssimazione e di flessibilità. Una prima causa di approssimazione è dovuta al fatto che i dati di un problema sono noti esattamente solo in rari casi e quindi dobbiamo essere consapevoli dell'effetto di questa imprecisione sulla decisione finale. Vi sono situazioni in cui si può solo ipotizzare la distribuzione di probabilità dei dati e quindi la decisione finale sarà necessariamente affetta da incertezza.

L'approssimazione può anche essere dovuta alla costruzione stessa del modello. Ad esempio, introdurre tutti i vincoli possibili non è mai conveniente per diversi motivi: siccome vi sono categorie di modelli matematici per i quali sono noti e disponibili dei metodi risolutivi, è opportuno costruire il modello cercando di farlo rientrare in una delle categorie note. Se alcuni vincoli sono inesprimibili all'interno del particolare tipo di modello scelto e inoltre risultano all'analisi meno rilevanti di altri, è più conveniente non considerarli. Quindi è inevitabile che il modello sia approssimato rispetto alla realtà che si vuole descrivere. Inoltre la presenza di un elevato numero di vincoli e di variabili rende un modello troppo sensibile ad aspetti marginali e quindi meno robusto rispetto a variazioni dei dati.

Altri margini di incertezza sono dovuti alla presenza nei problemi reali di aspetti difficilmente formalizzabili in modo quantitativo e che vanno quindi riformulati mediante "trucchi" che non hanno un immediato riscontro con la realtà e la cui efficacia va verificata sperimentalmente.

In conclusione, nell'analizzare la realtà per mezzo di modelli non va mai dimenticato lo scarto esistente tra la realtà stessa ed il modello: la soluzione di

un problema è in realtà sempre la soluzione della rappresentazione che abbiamo costruito del problema reale. È sempre necessario prestare grande attenzione alla fondatezza del modello costruito: il modello sarà sempre una descrizione molto limitata della realtà, ma dovrà rappresentare con ragionevole accuratezza gli aspetti che interessano ai fini della soluzione del problema decisionale che si sta affrontando.

Le motivazioni che rendono molto utile la costruzione di un modello matematico, cioè un approccio modellistico come quello della OR, sono molteplici e possono essere riassunte come segue:

- *La possibilità di risolvere matematicamente il problema.*
  Grazie al modello è possibile infatti analizzare matematicamente il problema ed ottenere così una soluzione che, soprattutto in riferimento a scopi di pianificazione, permette di adottare strategie che da una semplice analisi strutturale del problema non apparirebbero evidenti o che a volte potrebbero essere perfino controintuitive.

- *Una maggiore comprensione del problema.*
  Il modello è una rappresentazione semplificata del problema e spesso la sua costruzione consente di individuare proprietà strutturali del problema che altrimenti non sarebbero affatto evidenti.

- *La deduzione analitica di importanti proprietà.*
  Nella fase di analisi del modello è possibile dedurre per via analitica alcune importanti proprietà del problema sulla base dei risultati disponibili per la classe di problemi a cui si fa riferimento.

- *La possibilità di simulazioni.*
  Con un modello è possibile effettuare esperimenti che spesso non è possibile effettuare direttamente nella realtà.

Le principali critiche all'approccio modellistico possono invece essere sintetizzate nei seguenti due punti:

- L'impossibilità di quantificare soddisfacentemente con opportuni valori numerici alcuni dati richiesti dal modello (ad esempio, nel tentativo di quantificare con un costo o con un profitto alcuni valori sociali soprattutto in relazione a scopi di pianificazione). Tale punto riguarda quindi la possibilità (non remota) di dover trattare concetti non facilmente quantificabili.

- La possibilità che la qualità delle risposte che un modello produce dipendano profondamente dall'accuratezza dei dati prodotti. Tale punto riguarda quindi la possibile mancanza di precisione di alcuni dei dati immessi nel modello.

Alla prima critica si può però obiettare che ogni approccio scientifico può difficilmente evitare la difficoltà di quantificazione di certi dati e che il modo migliore di superare tale problema consiste nell'incorporare tale quantificazione nel modello stesso. La seconda critica è meno rilevante della precedente, in quanto anche se alcuni dati introdotti sono poco accurati, è ancora possibile che la struttura del modello sia tale da garantire che la soluzione sia sufficientemente accurata.

Una volta determinato il modello corretto per un problema, la Ricerca Operativa si occupa di fornire una procedura esplicita per determinarne una soluzione. Tale procedura può essere rappresentata da metodi matematici analitici o, come più spesso accade, da metodi numerici che determinano la soluzione del problema mediante specifici algoritmi di calcolo.

In questo contesto, il merito maggiore della OR consiste nello studiare un sistema nel suo complesso, dato che la maggior parte dei problemi reali coinvolge diverse parti di un sistema mutuamente interagenti ed è quindi essenziale studiarne l'interazione reciproca. Questa è una caratteristica distintiva della OR rispetto ad altre discipline ed è quindi evidente che un aspetto caratterizzante la OR sia proprio l'interdisciplinarietà. In effetti, le tecniche di cui la OR fa uso sono numerose e provengono da diverse branche della matematica: dall'algebra lineare alla logica, dalla statistica alla teoria dei giochi, dalla teoria delle decisioni alla teoria dei sistemi. Questo ha prodotto lo sviluppo di metodologie di soluzione che rappresentano un'inusuale combinazione di tecniche e strumenti tipici di altri settori.

Nel seguito di questo lavoro, dopo aver introdotto quelli che sono i problemi tipicamente trattati dalla OR, cioè i problemi di ottimizzazione e più in particolare quelli di Ottimizzazione Combinatoria, illustreremo le principali tecniche risolutive sviluppate ed utilizzate dalla Ricerca Operativa per tali problemi.

1.2 Problemi di Ottimizzazione

Un *problema* è una domanda espressa in termini generali, la cui risposta dipende da un certo numero di parametri e variabili.

Un problema viene usualmente definito per mezzo di:

- una descrizione dei suoi parametri, in generale lasciati indeterminati;
- una descrizione delle proprietà che devono caratterizzare la risposta o soluzione desiderata.

Una *istanza* di un dato problema P è quella particolare domanda che si ottiene specificando particolari valori per tutti i parametri di P.

Molto spesso un problema viene definito fornendo l'insieme F delle possibili soluzioni. Di tale insieme, detto *insieme realizzabile*, viene in generale data la struttura con i parametri da cui essa dipende; i suoi elementi vengono detti *soluzioni realizzabili*.

Frequentemente l'insieme F viene specificato indicando un insieme "base" F' tale che $F \subseteq F'$, ed ulteriori condizioni (vincoli) che gli elementi di F devono soddisfare. In questo caso si parla spesso degli elementi di $F' \setminus F$ come di *soluzioni non realizzabili*.

Un *problema di ottimizzazione* è un problema in cui sull'insieme realizzabile F viene definita una funzione, detta *funzione obiettivo*,

$$c : F \longrightarrow \mathbb{R}$$

che fornisce il costo o il beneficio associato ad ogni soluzione; la *soluzione ottima* del problema è un elemento di F che rende minima (oppure massima) la funzione obiettivo.

Questo tipo di problema può essere matematicamente scritto come

$$(P) \qquad \min\{c(x) : x \in F\}.$$

Dato un problema P, chiamiamo $z(P)$ il valore ottimo della funzione obiettivo. Allora, una soluzione realizzabile $x^* \in F$ tale che $c(x^*) = z(P)$ è detta soluzione ottima per P; si tratta cioè di un vettore $x^* \in F$ tale che

$$c(x^*) \leq c(x) \qquad \forall\, x \in F.$$

È immediato verificare che un problema di ottimizzazione può essere indifferentemente codificato come problema di massimo o di minimo. Infatti, dato il problema

$$(P') \qquad \max\{-c(x) : x \in F\},$$

è immediato verificare che $z(P) = -z(P')$, cioè il valore che si ottiene minimizzando la funzione $c(x)$ sull'insieme F è uguale all'opposto del valore che si ottiene massimizzando la funzione $-c(x)$, sempre su F. I due problemi P e P' sono *equivalenti*, in quanto hanno le stesse soluzioni realizzabili e le stesse soluzioni ottime.

In certi casi ciò che il problema richiede è semplicemente la determinazione di una qualsiasi soluzione realizzabile, ovvero di determinare se l'insieme realizzabile F sia vuoto o meno; in questo caso si parla di *problema decisionale* oppure di *problema di esistenza*. Per tali problemi, si richiede di fornire un elemento $x \in F$, se ne esiste uno, oppure di dichiarare che F è vuoto.

Dato un problema decisionale definito su $F \subseteq F'$, ad esso è naturalmente associato il *problema di certificato*: dato $x \in F'$, verificare se $x \in F$. Il problema di certificato è un problema decisionale che richiede semplicemente una risposta "si" oppure "no".

Un problema di ottimizzazione può essere espresso come problema decisionale: basta usare come insieme in cui si cerca una soluzione realizzabile l'insieme delle sue soluzioni ottime. Analogamente, un problema decisionale può essere formulato come problema di ottimo: basta definire la funzione obbiettivo su di un opportuno insieme F' che includa F, assegnandole valore 0 per ogni elemento che appartiene a F e valore 1 per ogni elemento che non vi appartiene.

Dal punto di vista della risolubilità, ossia della possibilità di determinare una soluzione ottima di un problema di ottimizzazione (o una soluzione realizzabile di un problema decisionale), i problemi di ottimizzazione (o rispettivamente, decisionali) possono essere in prima approssimazione divisi in due classi: problemi "facili" e problemi "difficili". Queste due classi coincidono in larga parte rispettivamente con la *classe dei problemi polinomiali* ($\mathcal{P}$) e con la *classe dei problemi $\mathcal{NP}$–ardui*, che illustreremo dettagliatamente nella sezione successiva.

Dato un problema di ottimizzazione, possiamo pensare ad un problema di minimo, chiameremo *problema decisionale associato*, o sua *versione decisionale*, il problema che consiste nel verificare l'esistenza di una soluzione

realizzabile in

$$F_k = \{x \in F : c(x) \leq k\},$$

dove k è un prefissato valore. Si cerca cioè se esiste una soluzione realizzabile del problema di ottimizzazione che fornisca un valore della funzione obiettivo non superiore a k.

In un certo senso, il problema decisionale associato ad un problema di ottimizzazione ne è una versione parametrica: facendo variare il parametro k e risolvendo ogni volta un problema di esistenza, è possibile determinare il valore ottimo della funzione obiettivo, o almeno una sua approssimazione con precisione arbitraria (risolvere il problema di ottimizzazione equivale a risolverne la variante decisionale per ogni possibile valore di k). Infatti, normalmente i problemi di ottimizzazione e le loro versioni decisionali sono "ugualmente difficili".

Dato un problema di ottimizzazione, accade spesso di voler costruire una qualche "approssimazione" del problema dato, ad esempio tenendo in conto solamente alcune delle condizioni (vincoli) che le soluzioni realizzabili devono soddisfare.

In particolare, si definisce *rilassamento* di un qualsiasi problema P il problema

$$(\bar{P}) \qquad \min\{\bar{c}(x) : x \in \bar{F}\}$$

tale che $F \subseteq \bar{F}$ e $\bar{c}(x) \leq c(x) \quad \forall\, x \in F$. In altre parole, $\bar{P}$ è un rilassamento di P se ha "più soluzioni" di P e/o se la sua funzione obiettivo è una approssimazione inferiore della funzione obiettivo c di P sull'insieme F. In questo caso, è immediato verificare che il valore ottimo della funzione obiettivo di $\bar{P}$ fornisce una valutazione inferiore (un *lower bound*) del valore ottimo della funzione obiettivo di P, ossia $z(\bar{P}) \leq z(P)$.

Nel caso di problemi di massimo, la seconda condizione diventa $\bar{c}(x) \geq c(x) \quad \forall\, x \in F$, ed il rilassamento fornisce una valutazione superiore (un *upper bound*) del valore ottimo della funzione obiettivo di P, ossia $z(\bar{P}) \geq z(P)$.

Spesso accade che i rilassamenti siano più "facili" dei problemi originali, ossia che tali valutazioni (inferiori o superiori) possano essere ottenute molto più rapidamente rispetto al tempo richiesto per risolvere il problema originario; ciò può chiaramente rendere tali valutazioni molto utili.

1.3 Complessità Computazionale

Una volta che un problema P sia stato formulato, deve essere risolto; si è quindi interessati alla messa a punto di strumenti di calcolo, cioè di *algoritmi*, che data una qualsiasi istanza p del problema siano in grado di fornirne una soluzione in un tempo finito.

Un algoritmo che risolve P può essere definito come una sequenza finita di istruzioni che, applicata ad una qualsiasi istanza p di P, si arresta dopo un numero finito di passi (ovvero di computazioni elementari), fornendo una soluzione di p oppure indicando che p non ha soluzioni realizzabili.

Generalmente si è interessati a trovare l'algoritmo più "efficiente" per un dato problema. Per poter studiare gli algoritmi dal punto di vista della loro efficienza, o *complessità computazionale*, è necessario definire un modello computazionale; classici modelli computazionali sono la Macchina di Turing, la R.A.M. e la Macchina a Registri (MR).

1.3.1 Misure di Complessità

Dato un problema P, una sua istanza p, ed un algoritmo A che risolve P, indichiamo con costo (o *complessità*) di A applicato a p una misura delle risorse utilizzate dalle computazioni che A esegue su una macchina MR (presumibilmente in grado di calcolare qualsiasi funzione effettivamente computabile con procedimenti algoritmici) per determinare la soluzione di p.

Le risorse, in principio, sono di due tipi: memoria occupata e tempo di calcolo. Molto spesso la risorsa più critica è il tempo di calcolo (tempo di esecuzione dell'algoritmo) e quindi si usa soprattutto questa come misura della complessità degli algoritmi. Nell'ipotesi che tutte le operazioni elementari abbiano la stessa durata, il tempo di calcolo può essere espresso come numero di operazioni elementari effettuate dall'algoritmo.

Dato un algoritmo, è opportuno disporre di una misura di complessità che consenta una valutazione sintetica della sua "bontà" ed eventualmente un suo agevole confronto con algoritmi alternativi. Conoscere la complessità di A per ognuna delle istanze del problema P non è possibile (l'insieme delle istanze di un problema è normalmente infinito), né sarebbe di utilità pratica.

Si cerca allora di esprimere la complessità come una funzione $g(n)$ della dimensione n dell'istanza cui viene applicato l'algoritmo, definita come una misura del numero di bit necessari per rappresentare, con una codifica "ra-

gionevolmente" compatta, i dati che definiscono l'istanza, cioè una misura della lunghezza del suo input.

Dato che per ogni dimensione si hanno in generale molte istanze di quella dimensione, si sceglie $g(n)$ come il costo necessario per risolvere la più difficile tra le istanze di dimensione n; si parla allora di *complessità nel caso peggiore.*

A questo punto la funzione $g(n)$ risulta definita in modo sufficientemente rigoroso, continuando però ad essere di difficile uso come misura della complessità, dato che risulta difficile, se non praticamente impossibile, la valutazione di $g(n)$ per ogni dato valore di n. Questo problema si risolve sostituendo alla funzione $g(n)$ il suo ordine di grandezza; si parla allora di *complessità asintotica.*

Date due funzioni $f(x)$ e $g(x)$, diremo che:

1. $g(x)$ è $O(f(x))$ se esistono due costanti c_1 e c_2 per cui $g(x) \leq c_1 f(x) + c_2 \quad \forall\, x$;
2. $g(x)$ è $\Omega(f(x))$ se $f(x)$ è $O(g(x))$;
3. $g(x)$ è $\Theta(f(x))$ se $g(x)$ è allo stesso tempo $O(f(x))$ e $\Omega(f(x))$.

Sia $g(x)$ il numero di operazioni elementari che vengono effettuate dall'algoritmo A applicato alla più difficile istanza, tra tutte quelle che hanno lunghezza di input x, di un dato problema P. Diremo allora che la complessità di A è un $O(f(x))$ se $g(x)$ è un $O(f(x))$, è un $\Omega(f(x))$ se $g(x)$ è un $\Omega(g(x))$ ed un $\Theta(f(x))$ se $g(x)$ è un $\Theta(f(x))$.

Un algoritmo è detto avere una *complessità polinomiale* se il tempo di esecuzione è dell'ordine $O(x^k)$, dove k è una costante indipendente dalla lunghezza di input x. Se la funzione di complessità di tempo non può essere limitata da un polinomio, l'algoritmo è detto avere *complessià esponenziale.* Se l'espressione b^l, dove l è una costante ed b è il valore di input più grande, è parte della funzione che limita, cioè se il tempo di running è dell'ordine di $O(x^k b^l)$, allora l'algoritmo è detto avere *complessità pseudo-polinomiale.*

1.3.2 Classi P e NP

Chiameremo *problemi trattabili* quelli per cui esistono algoritmi la cui complessità sia un $O(p(x))$, con $p(x)$ un polinomio in x, e *problemi intrattabili* quelli per cui un tale algoritmo non esiste.

Per poter effettuare una più rigorosa classificazione dei diversi problemi, conviene far riferimento a problemi in forma decisionale.

Una prima importante classe di problemi è la *classe* $\mathcal{NP}$, costituita da tutti i problemi decisionali il cui problema di certificato associato può essere risolto in tempo polinomiale. In altri termini, i problemi in $\mathcal{NP}$ sono quelli per cui è possibile verificare efficientemente una risposta "si" , perché è possibile decidere in tempo polinomiale se una soluzione x è realizzabile per il problema.

Equivalentemente, si può definire $\mathcal{NP}$ come la classe di tutti i problemi decisionali risolubili in tempo polinomiale da una macchina MR non-deterministica ($\mathcal{NP}$ sta infatti per $\mathcal{P}$olinomiale nel calcolo $\mathcal{N}$on-deterministico). Una MR non-deterministica è il modello di calcolo (astratto) in cui una MR, qualora si trovi ad affrontare un'operazione di salto condizionale, può eseguire *contemporaneamente* entrambi i rami dell'operazione, e questo ricorsivamente per un qualsiasi numero di operazioni; si tratta cioè di un computer capace di eseguire un numero illimitato (ma finito) di calcoli in parallelo.

In altre parole, i problemi in $\mathcal{NP}$ sono quelli per cui esiste una computazione di lunghezza polinomiale che può portare a costruire una soluzione realizzabile, se esiste, ma questa computazione può essere "nascosta" entro un insieme esponenziale di computazioni analoghe tra le quali, in generale, non si sa come discriminare.

Un sottoinsieme della classe $\mathcal{NP}$ è la *classe* $\mathcal{P}$, costituita da tutti i problemi risolubili in tempo polinomiale, ossia contenente tutti quei problemi decisionali per i quali esistono algoritmi di complessità polinomiale che li risolvono. Per questo motivo i problemi in $\mathcal{P}$ sono anche detti *problemi polinomiali*.

Chiaramente $\mathcal{P} \subseteq \mathcal{NP}$, ma una domanda particolarmente importante è se esistano problemi in $\mathcal{NP}$ che non appartengono anche a $\mathcal{P}$, cioè se sia $\mathcal{P} \neq \mathcal{NP}$. A questa domanda non si è in grado di rispondere, anche se si ritiene fortemente probabile che la risposta sia positiva, cioè che effettivamente sia $\mathcal{P} \neq \mathcal{NP}$.

1.3.3 Problemi NP–completi e NP–ardui

Molti problemi, anche se apparentemente notevolmente diversi, possono tuttavia essere ricondotti l'uno all'altro. Dati due problemi decisionali P e Q e supponendo l'esistenza di un algoritmo A_Q che risolve Q in tempo costante (indipendente dalla lunghezza dell'input), diremo che P *si riduce in tempo polinomiale a* Q, e scriveremo $P \propto Q$, se esiste un algoritmo che risolve P in

tempo polinomiale utilizzando come sottoprogramma A_Q. In tal caso si parla di *riduzione polinomiale* di P a Q.

È facile verificare che la relazione $\propto$ ha le seguenti proprietà:

1. è riflessiva: $A \propto A$;
2. è transitiva: $A \propto B$ e $B \propto C \Rightarrow A \propto C$;
3. $A \propto B$ e $A \notin \mathcal{P} \Rightarrow B \notin \mathcal{P}$;
4. $A \propto B$ e $B \in \mathcal{P} \Rightarrow A \in \mathcal{P}$.

Possiamo ora definire la *classe dei problemi $\mathcal{NP}$–completi*: un problema A è detto $\mathcal{NP}$–completo se $A \in \mathcal{NP}$ e se per ogni $B \in \mathcal{NP}$ si ha che $B \propto A$.

La classe dei problemi $\mathcal{NP}$–completi costituisce un sottoinsieme della classe $\mathcal{NP}$ di particolare importanza. Un fondamentale teorema (dovuto a Cook, 1971) garantisce che tale classe non è vuota.

I problemi $\mathcal{NP}$–completi hanno l'importante proprietà che se esiste per uno di essi un algoritmo polinomiale, allora (per transitività) necessariamente tutti i problemi in $\mathcal{NP}$ sono risolubili in tempo polinomiale, e quindi $\mathcal{P} = \mathcal{NP}$. In un certo senso, tali problemi sono i più difficili tra i problemi in $\mathcal{NP}$. Un problema si dice *$\mathcal{NP}$–completo in senso forte* se per esso non esiste alcun algoritmo pseudo-polinomiale a meno che $\mathcal{P} = \mathcal{NP}$.

Un problema che abbia come caso particolare un problema $\mathcal{NP}$–completo si dice *problema $\mathcal{NP}$–arduo* ed ha la proprietà di essere almeno tanto difficile quanto i problemi $\mathcal{NP}$–completi (a meno di una funzione moltiplicativa polinomiale). Si noti che un problema $\mathcal{NP}$–arduo può anche non appartenere alla classe $\mathcal{NP}$. Come definito per i problemi $\mathcal{NP}$–completi, allo stesso modo un problema si dice *$\mathcal{NP}$–arduo in senso forte* se per esso non esiste alcun algoritmo pseudo-polinomiale a meno che $\mathcal{P} = \mathcal{NP}$.

Nonostante tutti gli sforzi dei ricercatori, non è stato possibile fino ad ora determinare per nessun problema $\mathcal{NP}$–arduo un algoritmo polinomiale, il che avrebbe fornito algoritmi polinomiali per tutti i problemi della classe. Questo fa oggi ritenere che non esistano algoritmi polinomiali per i problemi $\mathcal{NP}$–completi, ossia, come già detto, che sia $\mathcal{P} \neq \mathcal{NP}$.

Allo stato delle conoscenze attuali, quindi, tutti i problemi $\mathcal{NP}$–ardui sono presumibilmente intrattabili. Naturalmente, ciò non significa che non sia in molti casi possibile costruire algoritmi in grado di risolvere efficientemente

istanze di problemi $\mathcal{NP}$–ardui di dimensione significativa (quella richiesta dalle applicazioni reali).

1.4 Problemi di Ottimizzazione Combinatoria

All'interno dei problemi di ottimizzazione, in base alla struttura dell'insieme realizzabile, si possono distinguere due importanti classi di problemi. Nel caso in cui un problema di ottimizzazione P sia caratterizzato dal fatto che in ogni sua istanza la regione realizzabile F contiene un numero finito di punti (e quindi la soluzione ottima può essere trovata confrontando un numero finito di soluzioni), si parla di *problema di Ottimizzazione Combinatoria* (o *Ottimizzazione Discreta*); si parla invece di *problema di Ottimizzazione Continua* se la regione realizzabile può contenere un'infinità non numerabile di punti. Mentre nei modelli di Ottimizzazione Combinatoria le variabili sono vincolate ad essere numeri interi, nei modelli di Ottimizzazione Continua le variabili possono assumere tutti i valori reali.

Tali classi di modelli rientrano nella categoria più generale della *Programmazione Matematica*, disciplina che svolge un ruolo di fondamentale importanza all'interno della Ricerca Operativa e che ha per oggetto lo studio di problemi in cui si vuole minimizzare o massimizzare una funzione reale definita su $\mathbb{R}^n$ le cui variabili sono vincolate ad appartenere ad un insieme prefissato, vale a dire lo studio di problemi di ottimizzazione.

Solitamente l'insieme realizzabile F viene descritto da un numero finito di disuguaglianze del tipo $g(x) \geq b$, dove g è una funzione definita su $\mathbb{R}^n$ a valori reali e $b \in \mathbb{R}$. Formalmente, date m funzioni $g_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m$, si esprime F nella forma

$$F = \{x \in \mathbb{R}^n : g_i \geq b_i, i = 1, \ldots, m\}.$$

Un problema di ottimizzazione può quindi essere riscritto nella forma:

$$\min\{c(x) : g_i \geq b_i, i = 1, \ldots, m\},$$

e viene chiamato *problema di Programmazione Matematica.*

I problemi di Programmazione Matematica si possono classificare in base alla struttura delle funzioni che li definiscono.

Più precisamente, si hanno problemi di:

- *Programmazione Lineare* se la funzione $c(x)$ e tutte le g_i sono lineari;
- *Programmazione Lineare Intera* se oltre alla condizione di linearità per la funzione obiettivo ed i vincoli, il vettore x delle variabili deve essere intero;
- *Programmazione Lineare Mista* se non tutto l'insieme delle variabili ma solo un suo sottoinsieme deve assumere valori interi.

Mentre i problemi di Programmazione Lineare appartengono alla classe $\mathcal{P}$, la maggior parte dei problemi di Programmazione Lineare Intera, che sono problemi di Ottimizzazione Combinatoria, sono problemi $\mathcal{NP}$–ardui. Quindi, in generale, mentre i primi sono "facili" da risolvere, per i problemi di Programmazione Lineare Intera è necessario un algoritmo con complessità esponenziale.

1.4.1 Programmazione Lineare

La *Programmazione Lineare* (brevemente, PL) è una disciplina matematica relativamente giovane, che si fa comunemente risalire al 1947, quando G.B. Dantzig propose un algoritmo, l'*algoritmo del simplesso*, come metodo efficiente per risolvere problemi coinvolgenti grandezze lineari. Al tempo Dantzig lavorava nello SCOOP (*Scientific Computation of Optimum Programs*), un gruppo di ricerca americano risultante da una intensa attività scientifica durante la seconda guerra mondiale, che mirava a razionalizzare la logistica legata alle operazioni di guerra.

Nel 1939 il matematico sovietico Kantorovitch aveva già proposto un simile metodo per l'analisi di piani economici, ma il suo contributo è stato a lungo ignorato dalla comunità scientifica occidentale. Sembra che persino Fourier (nel 1827) avesse già pensato a metodi di questo genere per trovare soluzioni realizzabili di un sistema di disuguaglianze lineari.

Viene quindi spontaneo chiedersi per quale motivo tale metodo è riuscito a prendere piede solo con l'intervento di Dantzig. Bisogna innanzitutto dire che la caratteristica di tutti i lavori antecedenti quelli di Dantzig era uno scarso interesse verso l'applicabilità pratica, dovuta principalmente all'impossibilità di effettuare i calcoli necessari.

Di certo quello che rende il contributo di G.B. Dantzig così importante rispetto ai precedenti, è la concomitanza con altri due fenomeni:

- il considerevole sviluppo del computer che ha permesso l'implementazione di algoritmi per la risoluzione di problemi reali di dimensioni considerevoli;

- il parallelo sviluppo di una struttura matriciale (*Interindustry Input-Output Model*) per lo studio dell'economia proposto da W.A. Leontieff, che con il suo lavoro ha mostrato come l'intera economia potesse essere rappresentata in una sorta di struttura di programmazione lineare.

Al contrario dei metodi proposti precedentemente, il metodo del simplesso si rivelò efficiente nella pratica e questo, unitamente al simultaneo avvento dei calcolatori elettronici, decretò il successo della PL e, con esso, l'inizio dello sviluppo rigoglioso della Ricerca Operativa.

Per questi motivi il metodo di Programmazione Lineare ha cominciato ad essere considerato sia uno strumento efficiente per il calcolo delle soluzioni di una larga scala di problemi di ottimizzazione, sia un paradigma generale di equilibrio economico tra diversi settori di scambio di risorse e servizi, attirando così l'interesse di larga parte della ricerca scientifica.

Storicamente, lo sviluppo il PL è guidato dalle sue applicazioni nell'economia e nella gestione. Inizialmente Dantzig ha sviluppato il metodo del simplesso per risolvere i problemi di pianificazione dei voli della Air Force americana, e problemi di pianificazione e di scheduling ancora dominano le applicazioni di PL.

Il suo grande successo è invece dovuto al fatto che essa si presta a modellare con facilità una larga varietà di attività economiche, come ad esempio la produzione da risorse scarse (allocazione ottima di risorse), la logistica (molti aspetti del trasporto e dispiegamento di risorse), lo scheduling (assegnamento di personale e macchine ai lavori), l'equilibrio economico (equilibrio domanda/offerta in un mercato competitivo), la copertura dai rischi (analisi di portfolio), etc.

Il problema generale della PL è detto *programma lineare* e consiste in un problema di ottimizzazione caratterizzato dalle seguenti proprietà:

1. un numero finito n di variabili x_i , che possono assumere valori reali non negativi;

2. una funzione obiettivo lineare, cioè del tipo:

$$f(x) = c^T x = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n = \sum_{i=1}^{n} c_i x_i,$$

dove $c \in \mathbb{R}^n$ è il vettore dei costi (fissato) ed $x \in \mathbb{R}^n$ è il vettore delle variabili;

3. l'insieme realizzabile è definito da un insieme finito di equazioni e/o disequazioni, cioè vincoli lineari del tipo $ax = b$, $ax \leq b$ e $ax \geq b$, dove $a \in \mathbb{R}^n$ e $b \in \mathbb{R}$.

Un problema di PL può sempre essere espresso per mezzo della formulazione seguente:

$$\max\{c^T x : Ax \leq b, x \geq 0\},$$

dove A è una matrice reale $m \times n$ e $b \in \mathbb{R}^m$ ed m è il numero di vincoli. In questo caso si dice che il programma lineare è in *forma canonica*. Notiamo che un problema in cui si voglia minimizzare anziché massimizzare la funzione obiettivo, si può riscrivere in *forma standard* (cioè con vincoli di uguaglianza) cambiando il segno al vettore c dei coefficienti di costo. In questo modo un qualsiasi problema di PL come essere scritto come problema di minimizzazione.

Un vettore x che soddisfi i vincoli del programma lineare, rappresentati dall'espressione $Ax \leq b$, $x \geq 0$, è detto *soluzione realizzabile* del programma, e tale vettore è detto *soluzione ottima* se oltre ad essere una soluzione realizzabile massimizza (o minimizza, a seconda del problema) la funzione obiettivo.

Ogni istanza di un programma lineare rientra in una fra tre categorie distinte; infatti, un programma lineare si dice:

1. *irrealizzabile*, se non esiste alcuna soluzione realizzabile al problema, non ci sono cioè vettori x per i quali tutti i vincoli del problema siano soddisfatti;

2. *illimitato*, se i vincoli non contengono sufficientemente la funzione di costo in modo che per ogni soluzione realizzabile data, possa essere trovata un'altra soluzione realizzabile che produca un ulteriore miglioramento della funzione di costo;

3. *ottimo*, se non è irrealizzabile o illimitato, cioè se la funzione di costo ha un unico valore minimo (o massimo); si noti però che questo non significa che i valori delle variabili che producono la soluzione ottima siano unici.

I problemi di PL formano una delle più importanti classi di modelli di ottimizzazione, per i seguenti motivi:

- molti problemi di ottimizzazione nella pratica possono essere modellati come problemi di PL;
- la PL appartiene alla classe dei problemi polinomiali, cioè "facili"; in effetti, per tali problemi sono disponibili algoritmi estremamente efficienti in grado di risolvere anche istanze di dimensione molto grande;
- la PL fornisce gli strumenti fondamentali per l'analisi e la costruzione di algoritmi efficienti per molte altre classi di problemi.

Proprio per il fatto che molti problemi reali possono essere espressi nell'ambito della PL, molta attenzione è stata rivolta allo studio di algoritmi per la risoluzione di tale classe di problemi.

Nel caso della PL si dice che un algoritmo *risolve* un problema se è capace di determinare correttamente se il problema dato ha un insieme realizzabile vuoto oppure è limitato oppure, se nessuno di questi due casi risulta verificato, sia capace di individuare una soluzione ottima.

Esistono diversi algoritmi per la risoluzione di problemi di PL, ma quello più spesso usato è proprio l'algoritmo del simplesso introdotto da Dantzig. Esso consiste di due passi base, spesso detti "fasi": la prima fase consiste nel cercare una soluzione realizzabile al problema, mentre la seconda consiste nel tentativo di migliorare iterativamente il valore della funzione obiettivo, cercando una variabile del problema che possa essere aumentata a spese della diminuzione di un'altra variabile, in modo da produrre un miglioramento globale della funzione obiettivo. L'algoritmo del simplesso risolve programmi lineari molto rapidamente, almeno nella pratica; tuttavia, su particolari input, esso può richiedere un tempo di esecuzione esponenziale.

L'algoritmo del simplesso è un algoritmo "geometrico", nel senso che si basa sull'interpretazione geometrica di un programma lineare.

Ogni vincolo di un programma lineare corrisponde geometricamente ad un semispazio in $\mathbb{R}^n$. L'insieme di vincoli di un problema corrisponde ad un sistema di disequazioni in $\mathbb{R}^n$, la cui soluzione è l'intersezione di un numero finito di semispazi, cioè un *poliedro* in $\mathbb{R}^n$.

Dunque la regione realizzabile è rappresentata da un poliedro, che è un insieme convesso in cui per ogni coppia di punti si ha che appartiene al poliedro

anche il segmento che li congiunge. Sapere se un programma lineare ammette almeno una soluzione è equivalente a sapere se il poliedro corrispondente è vuoto oppure no. Il numero di vettori x da esaminare in tale insieme per determinarne uno ottimo è un numero finito, rappresentato dai vertici del poliedro (che sono in numero finito). Una soluzione che massimizzi la funzione obiettivo non può che trovarsi in qualche "punto estremo" o "vertice" della regione realizzabile. Trovare il valore ottimo per una funzione lineare sotto determinati vincoli equivale a trovare il vertice del poliedro più lontano dall'origine secondo una direzione dello spazio (quella del vettore di costo c). Nel caso in cui il poliedro sia limitato (e così il valore massimo del programma lineare è sempre finito), si parla di *politopo*.

L'algoritmo del simplesso è un algoritmo geometrico che partendo da un qualunque vertice del politopo, procede da uno spigolo all'altro fino a che la funzione obiettivo non può più essere migliorata ed in tale caso il valore migliore trovato rappresenta la soluzione ottima del problema.

Altri algoritmi geometrici più complessi sono stati proposti (come il metodo dell'elissoide), caratterizzati da complessità teoriche polinomiali; ma l'algoritmo del simplesso, sebbene nel caso peggiore possa essere esponenziale, è in pratica più rapido della maggior parte di questi metodi.

Mettiamo infine in evidenza le caratteristiche che un problema reale deve possedere per poter essere formulato come modello di PL.

Le ipotesi che vengono assunte nel formulare un problema come modello di PL sono le seguenti:

- *proporzionalità*: il contributo di una variabile di decisione alla funzione obiettivo e ai vincoli è proporzionale secondo una costante moltiplicativa alla quantità rappresentata dalla variabile stessa;

- *additività*: il contributo delle variabili di decisione alla funzione obiettivo e ai vincoli è dato dalla somma dei contributi di ogni singola variabile;

- *continuità* (o *divisibilità*): ogni variabile di decisione può assumere tutti i valori reali nell'intervallo di realizzabilità, e quindi le variabili possono assumere valori frazionari.

In relazione ad applicazioni reali, queste ipotesi non rappresentano una grossa restrizione nel senso che, come già detto, sono molti gli ambiti ed i problemi

che sono ben rappresentati da un modello di PL. Bisogna dire però che l'ultima è forse l'ipotesi più spesso non verificata nelle applicazioni reali, dato che frequentemente alcune incognite possono assumere solo valori interi, a volte soltanto binari.

La particolare attenzione dedicata ai modelli di PL deriva, comunque, dai numerosi vantaggi che essa presenta e che possono essere così sintetizzati:

1. *Generalità e flessibilità.*
   I modelli di PL possono descrivere moltissime situazioni reali anche assai diverse tra loro e quindi hanno un carattere di universalità e di adattabilità alle diverse realtà applicative e anche quando l'ipotesi di linearità non è accettabile, il modello lineare costituisce una buona base di partenza per successive generalizzazioni.

2. *Semplicità.*
   I modelli di PL sono espressi attraverso il linguaggio dell'algebra lineare e quindi sono facilmente comprensibili anche in assenza di conoscenze matematiche più elevate.

3. *Efficienza degli algoritmi risolutivi.*
   Come accennato in precedenza, i modelli reali hanno dimensioni molto elevate ed è quindi indispensabile l'uso di un calcolatore che con opportuni programmi di calcolo possa rapidamente fornire una soluzione numerica. Relativamente ai modelli di PL esistono programmi molto efficienti e largamente diffusi che sono in grado di risolvere rapidamente problemi di dimensione molto considerevole.

4. *Possibilità di analisi qualitative.*
   I modelli di PL permettono di ottenere, oltre alla soluzione numerica del problema, anche ulteriori informazioni relative alla dipendenza della soluzione da eventuali parametri presenti, che possono avere significative interpretazioni economiche.

1.4.2 Programmazione Lineare Intera

In molti problemi pratici, attività e risorse sono "entità" indivisibili e sono presenti regole che definiscono un numero finito di scelte permesse. Conseguentemente tali problemi possono essere formulati in modo appropriato

usando procedure che trasformano le descrizioni logiche delle alternative in vincoli lineari su un qualche insieme di variabili che devono assumere certi valori discreti, cioè possono essere modellati come problemi di PL con l'aggiunta di un vincolo di interezza per le variabili usate. Tali problemi sono detti problemi di *Programmazione Lineare Intera* (brevemente, PLI).

I problemi di PLI costituiscono un'estensione importantissima del campo di applicazione dei modelli deterministici di Programmazione Matematica. Attraverso la modellazione di variabili a valori discreti è possibile infatti includere in un modello scelte tra diverse alternative logiche che permettono di formulare modelli di notevole interesse applicativo.

In analogia con il formalismo introdotto per la PL, un problema generale di PLI è detto *programma lineare intero* e consiste in un problema di ottimizzazione caratterizzato dalle seguenti proprietà:

1. un numero finito n di variabili x_i, che possono assumere solamente valori interi non negativi;

2. una funzione obiettivo lineare, cioè del tipo $f(x) = c^T x$, dove $c \in \mathbb{R}^n$ è il vettore dei costi (fissato) ed $x \in \mathbb{Z}$ è il vettore delle variabili;

3. l'insieme realizzabile è definito da un insieme finito di vincoli lineari del tipo $ax = b$, $ax \leq b$ e $ax \geq b$, dove $a \in \mathbb{R}^n$ e $b \in \mathbb{R}$.

Un programma lineare intero si può sempre scrivere in forma canonica nel modo seguente:

$$\max\{c^T x : Ax \leq b, x \geq 0, x \in \mathbb{Z}\},$$

dove A è una matrice reale $m \times n$ e $b \in \mathbb{R}^m$ ed m è il numero di vincoli. Come detto per i programmi lineari, anche in questo caso è comunque sempre possibile riscrivere un programma lineare intero di massimizzazione come problema di minimizzazione.

I problemi di PLI presentano molte più difficoltà rispetto a quelli di PL, tanto che è praticamente insperabile risolvere in tempo ragionevole problemi di taglia molto grande. Mentre per i problemi di PL esiste un algoritmo generale, l'algoritmo del simplesso, che può essere applicato ad un qualsiasi problema, per la PLI non esiste un tale algoritmo ma esistono piuttosto algoritmi particolari per ogni tipologia di problemi che sfruttano la particolare struttura del problema in esame.

In particolare, negli ultimi anni si è assistito allo sviluppo di metodologie specializzate in grado di determinare soluzioni ottime per problemi di notevole dimensione. Tuttavia, per la maggior parte dei modelli di PLI non esistono algoritmi in grado di risolvere qualunque istanza con tempi di calcolo che crescano a velocità minore dell'esponenziale col crescere della dimensione.

Appare chiaro che nel caso di modelli di ottimizzazione intera, la fase della modellistica e quella algoritmica siano fortemente legate. A differenza da quel che accade per i problemi di PL, nel caso dell'Ottimizzazione Discreta una formulazione "matematicamente corretta" di un problema non è in genere sufficiente per poter risolvere il problema stesso. Spesso occorre un lavoro molto accurato di formulazione del problema prima di poter iniziare la sua risoluzione. Solo recentemente alcuni risolutori sono stati equipaggiati di fasi di "pre-processing" orientate al miglioramento automatico della formulazione.

Inoltre, dato che ottenere una soluzione ottima di un problema di PLI di dimensioni considerevoli (come nel caso della maggior parte dei problemi reali di un certo interesse) in un tempo di calcolo ragionevole può giustamente dipendere dal modo in cui il problema è stato formulato, molta recente ricerca è stata diretta alla riformulazione di problemi di Ottimizzazione Combinatoria.

Recentemente, numerosi problemi "difficili" sono stati risolti riformulandoli o come *problemi di set covering* o come *problemi di set partitioning*, che hanno un numero enorme di variabili. Per questo motivo, anche piccole istanze di un problema risultano difficilmente risolvibili e quindi vengono utilizzate tecniche in grado di affrontare efficacemente tale tipologia di problemi, note con il nome di *generazione di colonne* o di *decomposizione di Dantzig-Wolfe* o ancora come *metodi di banch-and-price*.

Come già detto, il caso della PLI, dove si cercano soluzioni a coordinate intere è molto più complesso del caso della PL. In effetti, dal punto di vista geometrico, il dominio di ottimizzazione non è più un corpo convesso continuo, ma un insieme discreto di punti. Nella PL, a causa della convessità del programma lineare, si può sfruttare il fatto che un ottimo locale è anche un ottimo globale e quindi per trovare la soluzione ottima è sufficiente percorrere l'insieme dei vertici del politopo (dato che essi corrispondono a delle soluzioni). Nella PLI tale approccio non è più valido, dato che i vertici del politopo sono in generale a coordinate frazionarie (tranne quando la matrice A è totalmente unimodulare, nel qual caso i vertici sono a coordinate intere).

Quando si discute la nozione di "buona formulazione", normalmente si pen-

sa alla creazione di un problema più facile da risolvere che approssimi bene il valore della funzione obiettivo del problema originale. Visto che la convessità della regione realizzabile è distrutta in un programma lineare intero proprio dalla restrizione di interezza per i valori delle variabili, l'approssimazione generalmente più usata rimuove tale restrizione. Una tale approssimazione è nota come *rilassamento lineare* o *rilassamento continuo* del problema originale.

Quindi, dato il programma lineare intero

$$\max\{c^T x : Ax \leq b, x \geq 0, x \in \mathbb{Z}\}$$

si può risolvere lo stesso problema in cui però si elimina la condizione di interezza, si può cioè risolvere il problema

$$\max\{c^T x : Ax \leq b, x \geq 0\}.$$

Il rilassamento lineare ha chiaramente un insieme di soluzioni realizzabili più grande rispetto a quello del problema originario, contenendo soluzioni intere ma anche frazionarie.

Se il sistema $Ax \leq b$ di vincoli nella rappresentazione geometrica del problema individua il poliedro P', si ha che tutti i punti all'interno di P' sono soluzioni, intere o frazionarie, realizzabili del rilassamento mentre le soluzioni del programma lineare intero sono solo i punti di coordinate intere. Se la soluzione del problema rilassato è intera, allora essa rappresenta anche la soluzione ottima del problema originario; se invece la soluzione del problema rilassato è frazionaria, questa rappresenta una approssimazione per eccesso della soluzione ottima del problema originario ed infine se il rilassamento è irrealizzabile lo è anche il problema originario.

Nel caso in cui la soluzione del rilassamento lineare sia frazionaria, visto che tale soluzione rappresenta una approssimazione di quella ottima, potrebbe sembrare una buona strategia quella di arrotondare la soluzione trovata. Tuttavia, un utilizzo non ragionato dello strumento dell'arrotondamento non può portare a risultati apprezzabili. Oltretutto, anche l'arrotondamento è un'operazione computazionale molto onerosa: in un problema con n variabili, se, data una soluzione di un rilassamento, si desiderasse controllare la realizzabilità di un arrotondamento, occorrerebbe effettuare due tentativi per la prima variabile (arrotondamento per eccesso e per difetto), due per la seconda, e così per tutte le n variabili, giungendo in questo modo a dover valutare 2^n soluzioni possibili, senza alcuna garanzia di poterne trovare una realizzabile.

Non esiste un unico rilassamento lineare di un problema di PLI, infatti possono essere trovati infiniti sistemi di disequazioni diversi dal sistema $Ax \leq b$ che costituiscono un poliedro diverso da P' ma che al suo interno mantiene gli stessi punti interi (soluzioni del programma lineare intero) interni a P'.

Esiste, in particolare, un rilassamento lineare del programma lineare intero che prende il nome di *formulazione ideale* ed è una particolare formulazione che permette di risolvere il programma lineare intero direttamente risolvendo il corrispondente rilassamento lineare (in quanto le soluzioni di questo sono certamente intere). Possiamo dire che la formulazione di un programma lineare intero è tanto migliore quanto più si avvicina alla formulazione ideale del problema stesso, vale a dire tanto più il poliedro che la rappresenta si avvicina al poliedro ideale, cioè il poliedro relativo alla formulazione ideale.

Comunque, rimuovere semplicemente le restrizioni di interezza può alterare in modo così significativo la struttura del problema che la soluzione del problema rilassato può essere lontana da quella intera del problema originario. Si potrebbe quindi considerare di aggiungere restrizioni aggiuntive, vincoli detti *piani di taglio*, al problema rilassato in modo che, almeno nelle vicinanze della soluzione ottima, il politopo del problema rilassato approssimi da vicino il poliedro descritto dall'inviluppo convesso di tutti i punti realizzabili per il programma lineare intero originale. Quando si considera di aggiungere tali vincoli al rilassamento di un problema di Ottimizzazione Combinatoria iterativamente all'interno di un algoritmo globale, si ottiene un *algoritmo di cutting plane*.

Un altro metodo algebrico che consente di trovare vincoli più forti ad un programma lineare intero in modo da avere formulazioni molto vicine a quella ideale è il *metodo di branch-and-bound*. Si tratta dell'approccio enumerativo più comunemente usato. Il "ramificare" (*branching*) si riferisce alla parte di enumerazione della tecnica risolutiva, mentre il "limitare" (*bounding*) si riferisce al sondare le soluzioni possibili per confronto con un limite superiore o inferiore sul valore di una soluzione nota.

Bisogna anche dire che la restrizione di integralità non è il solo approccio possibile per rilassare un problema. Infatti, un approccio alternativo per la risoluzione di problemi di PLI consiste nel rimuovere un insieme di vincoli, solitamente i più complicati, dall'insieme di tutti i vincoli del problema e nell'inserirli nella funzione obiettivo in "modo lagrangiano", cioè con fissati moltiplicatori (che vengono cambiati iterativamente). Questo approccio è conosciuto come *rilassamento lagrangiano*. Rimuovendo i vincoli più complicati si ottiene

un problema molto più semplice, che viene risolto ripetutamente fino a trovare valori ottimi per i moltiplicatori.

Al rilassamento lagrangiano è collegato un altro approccio, noto come *metodo di decomposizione lagrangiana*, che tenta di rinforzare i limiti trovati dal rilassamento. Questo approccio consiste nell'isolare insiemi di vincoli in modo da ottenere problemi facili e separati su ognuno dei sottoinsiemi.

Nel capitolo successivo descriveremo nel dettaglio tali tecniche risolutive per problemi di Ottimizzazione Combinatoria, sulle quali si basano i diversi approcci esatti al VRPTW che presenteremo nel seguito di questo lavoro.

Capitolo 2

Tecniche Risolutive di Ricerca Operativa

2.1 Introduzione

Nonostante per la maggior parte dei problemi di Ottimizzazione Combinatoria il processo di risoluzione sia molto difficile, sono stati sviluppati numerosi metodi efficaci in grado di risolvere (anche all'ottimalità) in un tempo ragionevole larga parte di tali problemi. Esiste un certo numero di approcci assai diversi per risolvere tali problemi di ottimizzazione, ed attualmente essi sono frequentemente combinati in soluzioni "ibride" che cercano di sfruttare i benefici di ognuno minimizzando i difetti.

Le tecniche di risoluzione disponibili possono essere classificate in due principali classi:

- la classe degli *algoritmi* (o *metodi*) *esatti*, i quali garantiscono di trovare una soluzione ottima di un problema e di provare la sua ottimalità o provare che non esistono soluzioni realizzabili;

- la classe degli *algoritmi approssimati*, i quali sacrificano la garanzia di trovare soluzioni ottime nell'interesse di ottenere soluzioni davvero buone in un tempo ragionevolmente breve.

I metodi citati nel capitolo precedente relativamente alla risoluzione di problemi di PLI, appartengono alla prima classe e sono alla base dei moderni algoritmi per la risoluzione di problemi di Ottimizzazione Combinatoria.

Nel seguito analizzeremo nel dettaglio i principali metodi esatti di PLI, vale a dire:

- il metodo di branch-and-bound;
- gli algoritmi di cutting plane;
- i metodi basati sul rilassamento lagrangiano;
- la generazione di colonne e la decomposizione di Dantzig-Wolfe;
- i metodi di programmazione dinamica.

Per quanto riguarda la seconda classe di metodi risolutivi, accenneremo alle principali tecniche di ricerca locale, euristiche e metaeuristiche, in grado di fornire buone soluzioni a problemi per i quali sia difficile elaborare metodi esatti, senza però dare garanzie sull'ottimalità delle soluzioni trovate.

I vantaggi dei metodi esatti rispetto quelli approssimati sono notevoli e possono essere riassunti come segue:

1. Se un algoritmo esatto ha successo si ottengono soluzioni certamente ottime.
2. Si ottengono informazioni preziose su limiti inferiori e superiori sulla soluzione ottima anche nel caso in cui l'algoritmo si arresti prima del completamento della procedura di ricerca (interruzione dovuta alla definizione di un criterio di arresto per l'algoritmo, anche prima del ritrovamento di una soluzione ottima).
3. I metodi esatti permettono di potare le parti dello spazio di ricerca in cui non possono esservi soluzioni ottime.

Comunque, a dispetto delle caratteristiche vantaggiose, i metodi esatti presentano anche un certo numero di svantaggi:

1. Per molti problemi la dimensione delle istanze che sono risolvibili nella pratica è piuttosto limitata ed il tempo necessario per la computazione aumenta fortemente con la dimensione dell'istanza.
2. La memoria necessaria per algoritmi esatti può essere anche molto grande e portare all'aborto precoce di un programma.

3. Per molti problemi di Ottimizzazione Combinatoria gli algoritmi che si comportano meglio sono specifici per un problema e richiedono molto tempo di sviluppo da parte di esperti in PLI.

4. Algoritmi esatti molto efficaci per un dato problema sono spesso difficilmente estensibili nel caso in cui vari qualche dettaglio nella formulazione del problema stesso.

Lo stato attuale dei metodi esatti è che qualche istanza anche di dimensione notevole di certi tipi di problemi può essere risolta molto velocemente, mentre per altri tipi di problemi non è possibile risolvere nemmeno istanze molto piccole.

Per quanto riguarda la seconda classe di metodi risolutivi, quella della ricerca locale è la classe di algoritmi approssimati di maggior successo. Quando applicata a problemi di Ottimizzazione Combinatoria difficili, la ricerca locale produce soluzioni di alta qualità applicando iterativamente piccole modifiche (movimenti locali) ad una soluzione, nella speranza di trovarne una migliore.

Inserito in meccanismi di livello più alto come le metaeuristiche, questo approccio ha mostrato di riuscire ad ottenere soluzioni quasi ottime (e qualche volta ottime) ad un certo numero di problemi difficili.

I metodi di ricerca locale hanno principalmente i seguenti vantaggi:

1. Nella pratica rappresentano i migliori algoritmi usati per la risoluzione di un grande numero di problemi.

2. Possono esaminare un'enorme quantità di possibili soluzioni in un tempo computazionale ragionevolmente breve.

3. Sono spesso più facili da adattare a lievi varianti di un problema, cioè sono spesso più flessibili dei metodi esatti.

4. Sono tipicamente più facili da capire ed implementare rispetto ai metodi esatti.

Tuttavia, i metodi basati sulla ricerca locale presentano diversi svantaggi:

1. Non possono provare l'ottimalità.

2. Tipicamente non possono ridurre in maniera dimostrabile lo spazio di ricerca.

3. Non hanno criteri ben definiti di termine (questo è particolarmente vero per le metaeuristiche).

4. Incontrano spesso delle difficoltà con problemi altamente vincolati in cui le aree realizzabili dello spazio delle soluzioni sono separate.

5. Nella pratica non sono disponibili risolutori di ricerca locale con finalità generali e di conseguenza la maggior parte delle applicazioni richiede sforzi considerevoli di programmazione (sebbene spesso minori di quelli necessari per i metodi esatti con finalità particolari, costruiti cioè per un problema in particolare).

Quindi, approcci esatti ed approssimati presentano particolari vantaggi e svantaggi e possono essere visti come complementari. Appare quindi ovvia l'idea di tentare di combinare queste due diverse tecniche risolutive in algoritmi ancora più potenti; comunque, gran parte della letteratura presenta combinazioni piuttosto semplici di ricerca locale e tecniche di PLI. Queste combinazioni consistono in un semplice approccio in due fasi: una prima fase in cui viene usato un metodo di una classe ed una seconda fase in cui viene usato un metodo dell'altra classe.

2.2 Metodo di Branch-and-Bound

Il *metodo di branch-and-bound* è il metodo più comunemente implementato nel software commerciale per la PLI e si basa sull'idea di un'enumerazione implicita delle soluzioni realizzabili di un problema.

Tale metodo consiste in un algoritmo che, partendo dal rilassamento lineare di un programma lineare intero, arriva a trovare una soluzione ottima intera del problema originario suddividendolo in vari sottoproblemi distinti, cioè suddividendo in vari sottoinsiemi l'insieme di tutte le soluzioni realizzabili per il problema, per mezzo della scelta dei valori di una delle variabili decisionali.

Dati un problema di PLI, che supponiamo di minimizzazione ed in forma standard,

$$\min\{c^T x : Ax = b, x \geq 0, x \in \mathbb{Z}\}$$

ed il poliedro associato al suo rilassamento lineare

$$P = \{x : Ax = b, x \geq 0\},$$

lo schema su cui si basano i metodi di tipo branch-and-bound è il seguente:

1. Viene risolto il rilassamento lineare del problema originale determinando $x^* \in \arg\min\{c^T x : x \in P\}$.

2. Se x^* è intera, allora essa è anche soluzione ottima del problema.

3. Altrimenti, sia x_j^* una componente di x^* non intera. Il problema originale è equivalente al problema seguente:

$$\min\{c^T x : x \in P \cap \mathbb{Z} \cap (\{x : x_j \leq \lfloor x_j^* \rfloor\} \cup \{x : x_j \geq \lceil x_j^* \rceil\})\}.$$

4. Dal problema originale vengono originati due sottoproblemi:

$$\min\{c^T x : x \in P, x \leq \lfloor x_j^* \rfloor, x \in \mathbb{Z}\}$$

   e

$$\min\{c^T x : x \in P, x \geq \lceil x_j^* \rceil, x \in \mathbb{Z}\},$$

   ai quali viene applicato, in modo ricorsivo, il medesimo algoritmo; delle due soluzioni ottime eventualmente trovate, quella di valore inferiore è la soluzione ottima del problema.

Questa tecnica è conosciuta anche come *procedura per separazione e valutazione*. La separazione, vale a dire l'operazione di suddivisione di un problema in due sottoproblemi, prende il nome di *branching* ("ramificare"), per il fatto che la situazione viene usualmente rappresentata mediante un albero di ricerca. La valutazione della funzione obiettivo, che prende il nome di *bounding* ("limitare"), permette di non considerare ("potare", usando la terminologia riferita ad una struttura arborescente) le parti dell'albero che conducono ad una soluzione non ottima.

L'esecuzione del metodo segue così il percorso di un albero in cui i nodi rappresentano le variabili ed i rami corrispondono agli assegnamenti di ciascuno dei valori possibili alle variabili. Una volta trovata nell'esplorazione di una zona dell'albero di ricerca una soluzione realizzabile per il problema, per verificare se si tratta di una soluzione ottima occorre visitare la restante parte dello spazio di ricerca e verificare che non esistano soluzioni migliori, cioè assegnamenti di valori alle variabili che migliorino il valore della funzione obiettivo rispetto a quello della soluzione trovata.

Un sottoproblema si dice "attivo" per indicare che può essere risolto iterando lo stesso procedimento di ramificazione. La ramificazione di un sottoproblema può terminare nelle foglie in due modi distinti: con un successo, nel caso in cui essa conduca ad una soluzione realizzabile oppure con un insuccesso, nel caso in cui essa conduca ad una soluzione non realizzabile o ad una soluzione realizzabile ma peggiore di quella corrente. Ogni volta che si presenta una soluzione realizzabile migliore di quella trovata fino a quel momento, viene aggiornato il limite superiore per il valore ottimo.

In generale, l'algoritmo di branch-and-bound in un certo istante ha una lista di problemi attivi, rappresentati da tutte le foglie del corrente albero, tra cui scegliere un problema da risolvere. La scelta delle decisioni di ramificazione prioritarie è cruciale. L'ordine di percorrenza dei rami dell'albero di ricerca è importante nel caso di problemi di realizzabilità, in cui si vogliano cioè trovare solo le soluzioni realizzabili. In tal caso, si cerca di visitare per primi i rami dell'albero che sembrano condurre a buone soluzioni, secondo un principio noto come *principio best-first*. Nel caso di problemi di ottimalità, invece, non risulta molto rilevante l'ordine di esplorazione dei rami dell'albero, dato che per trovare la soluzione ottima occorre comunque valutare tutte le soluzioni realizzabili e quindi esplorare l'intero albero di ricerca.

Per evitare l'esplorazione di tutto lo spazio di ricerca o calcoli ridondanti, e quindi minimizzare il tempo di esecuzione del procedimento risolutivo, è necessario avere a disposizione buoni criteri di esplorazione, cioè dei criteri che permettano di eliminare prima possibile dall'analisi le parti dell'albero che conducono certamente a soluzioni non accettabili.

Tale potatura dell'albero di ricerca può essere eseguita secondo il criterio noto come *pruning by infeasibility*, il quale produce una potatura derivante da un ragionamento di realizzabilità che consiste nell'eliminazione delle ramificazioni che rappresentano assegnamenti di valori alle variabili che conducono a soluzioni irrealizzabili.

Un altro importante criterio di potatura dell'albero è quello noto con il nome di *pruning by bound*, il quale produce una potatura derivante da un ragionamento di ottimalità che consiste nell'eliminazione delle ramificazioni che conducono a soluzioni realizzabili peggiori della migliore soluzione trovata fino a quel momento. Più precisamente, tale potatura consiste nell'eliminare quei rami (che rappresentano dei rilassamenti lineari) che conducono ad un *lower bound* (una qualsiasi soluzione del rilassamento, che è una approssimazione

per difetto) sul valore ottimo intero che risulti maggiore o uguale al *upper bound* corrente (una qualsiasi soluzione intera del problema originario, che rappresenta una approssimazione per eccesso del valore ottimo).

Il motivo per il quale l'operazione di bounding è valida è semplicemente legato al fatto che nel passaggio dal problema associato ad un nodo ai due problemi generati dal nodo stesso, l'insieme realizzabile del rilassamento lineare si riduce. Poiché il valore dell'ottimo di un rilassamento costituisce, nei problemi di minimo, un limite inferiore al valore dell'ottimo e nel passaggio da un problema a due sottoproblemi tale valore non può diminuire, ne segue che se ad una data iterazione il valore del rilassamento è peggiore del valore di una soluzione realizzabile (che costituisce un limite superiore all'ottimo), nessuna eventuale soluzione intera di valore strettamente minore può essere trovata nel sottoalbero che ha come radice il nodo corrente.

L'idea di base del metodo è quindi la seguente: per un dato problema di minimizzazione di una funzione obiettivo z soggetta ad un insieme di vincoli ed alla restrizione che la soluzione sia intera, si calcolano iterativamente upper bound e lower bound di z in modo che, passo dopo passo, la differenza tra i due limiti diminuisca fino a diventare nulla; a questo punto i due limiti coincidono e rappresentano proprio il valore ottimo intero di z, cioè la soluzione ottima del programma lineare intero originario. Se si vuole massimizzare anziché minimizzare, il procedimento da seguire è lo stesso a patto del rovesciamento delle regole sui limiti inferiore e superiore.

L'utilizzo sistematico dei limiti nella procedura descritta sottolinea l'interesse dei metodi di rilassamento di problemi, metodi che producono tali limiti e possono essere agevolmente combinati con altri, come il metodo di branch-and-bound.

2.3 Algoritmi di Cutting Plane

Dato un problema di PLI, un metodo per ottenere un suo rilassamento lineare più vicino possibile alla formulazione ideale del problema consiste nell'aggiunta di disequazioni, dette *piani di taglio* (o più semplicemente, *tagli*), al sistema di vincoli del rilassamento.

In base a tale metodo, all'insieme di vincoli di un rilassamento lineare che ha una soluzione ottima frazionaria si aggiungono delle disequazioni che "tengono da una parte" le soluzioni intere del problema e contemporaneamente

non sono soddisfatte dalla soluzione ottima frazionaria del rilassamento corrente. Si tratta quindi di aggiungere al sistema di vincoli preesistente degli ulteriori vincoli che "tagliano fuori" (da qui il nome dato a questo tipo di vincoli) la soluzione frazionaria (eliminando così la parte della regione realizzabile contenente tale soluzione) che in quanto tale non può certamente essere una soluzione ottima del problema originario. Questo procedimento, cioè l'aggiunta di piani di taglio al rilassamento del problema originario, si itera fino al raggiungimento di una soluzione intera, che sarà la soluzione ottima del problema.

Un algoritmo basato sul metodo dei tagli prende il nome di *algoritmo di cutting plane*. Metodi di questo tipo sono stati i primi utilizzati per la risoluzione dei problemi di PLI, e sono tuttora i più importanti insieme a quelli di tipo enumerativo, come il metodo di branch-and-bound.

Quindi, la generazione di piani di taglio determina l'inserimento nel sistema di vincoli del problema rilassato di alcune disequazioni lineari che rimuovono via via le soluzioni frazionarie, irrealizzabili per il problema di PLI originario, senza eliminare alcuna soluzione intera.

L'aggiunta di tali disequazioni rafforza il rilassamento lineare, nel senso che ha l'effetto di avvicinare sempre più il poliedro relativo al rilassamento al poliedro ideale del problema. Il metodo dei piani di taglio si basa quindi sulla soluzione iterativa di una sequenza di programmi lineari (a partire dal rilassamento iniziale dei vincoli di interezza del problema), ognuno dei quali approssima il problema originario ed è seguito da una approssimazione ancora migliore.

Il problema della ricerca del taglio da aggiungere ad un certo punto della procedura in cui si trovi una soluzione frazionaria, cioè della disequazione soddisfatta da tutte le soluzioni realizzabili del problema originario (soluzioni intere del programma lineare considerato nel punto della procedura) ma non dalla soluzione frazionaria trovata, è detto *problema di separazione*.

I piani di taglio per un dato problema possono essere molti, visto che ne esiste uno per ogni variabile frazionaria di una soluzione ottima di un suo rilassamento. Aggiungendo tali tagli al problema, uno alla volta, si può quindi aumentare di molto la taglia del problema; per questo motivo è conveniente eliminare qualche taglio nel momento in cui esso risulti ridondante.

La teoria relativa al metodo basato sui piani di taglio è dovuta a Ralph Gomory [50], il quale introdusse nel 1958 uno dei primi e tra i più importanti

metodi di taglio, il *metodo dei tagli di Gomory.*

Illustriamo ora la procedura teorica per ottenere, mediante il metodo algebrico per generare tagli introdotto da Gomory, la formulazione ideale per un generico problema di PLI nella forma

$$\max\{c^T x : Ax \leq b, x \geq 0, x \in \mathbb{Z}\}.$$

Indichiamo con

$$\mathcal{P}_I = conv\{x : Ax \leq b, x \geq 0, x \in \mathbb{Z}\}$$

l'inviluppo convesso relativo a tale problema, cioè l'insieme di tutti i vettori che possono essere espressi come combinazione convessa di vettori interi che soddisfano l'insieme di vincoli $Ax \leq b$. Si tratta naturalmente del poliedro ideale per il problema in studio.

Sia, invece,

$$P_0 = \{x : Ax \leq b, x \geq 0\}$$

l'insieme di tutti i vettori, non necessariamente interi, che soddisfano l'insieme di vincoli $Ax \leq b$. In generale, P_0 contiene $\mathcal{P}_I$, contenendo non solo soluzioni intere ma anche soluzioni frazionarie del problema.

L'obiettivo è quello di ottenere una descrizione dell'insieme $\mathcal{P}_I$ partendo da P_0 e cercando delle disuguaglianze (vincoli) da aggiungere al problema soddisfatte da tutti gli elementi di $\mathcal{P}_I$ ma non da tutti quelli di P_0.

Per trovare tali disuguaglianze, cioè i tagli, occorre ricordare che dato un qualsiasi poliedro P, un semispazio rappresentato dalla disuguaglianza $gx \leq h$ è detto "supportare" P se geometricamente "tocca" P, cioè se esiste un vettore $x \in P$ tale che $gx = h$ e se P è tutto contenuto nel semispazio in questione.

Per quanto riguarda il caso in esame, supponiamo che le a_i siano le colonne della matrice A e b_i gli elementi del vettore b, con $i = 1, \ldots, m$, e rappresentiamo l'insieme di vincoli del problema originale con le disuguaglianze $a_i^T x \leq b_i$. Siano $y_1, \ldots, y_m$ dei numeri reali non negativi, $g = \sum_{i=1}^m y_i a_i$ e $h = \sum_{i=1}^m y_i b_i$. Allora la disuguaglianza $gx \leq \lfloor h \rfloor$, con g intero, è verificata da tutti i vettori di $\mathcal{P}_I$, essendo soddisfatta da tutti i vettori interi che soddisfano $Ax \leq b$ e quindi anche da tutte le combinazioni convesse di tali punti. Essendo il valore $\lfloor h \rfloor$ un intero, la disuguaglianza $gx \leq \lfloor h \rfloor$ non è soddisfatta da tutti gli elementi di P_0, non essendo soddisfatta dai suoi elementi frazionari.

Si può allora definire l'insieme

$$P_1 = \{P_0 \cap (gx \leq \lfloor h \rfloor)\}$$

per tutti i semispazi $gx \leq \lfloor h \rfloor$ che supportano P_0 con g intero. P_1 è ottenuto da P_0 aggiungendo dei vincoli che non sono soddisfatti da tutti i vettori di P_0, ma che sicuramente sono soddisfatti da tutti i vettori di $\mathcal{P}_I$. Allora P_1 è più fine di P_0, nel senso che P_1 si avvicina più a $\mathcal{P}_I$ rispetto a P_0; inoltre P_1 è certamente un poliedro visto che le disuguaglianze aggiunte a P_0 sono in numero finito. Tali disuguaglianze sono dei tagli.

Iterando tale procedimento si restringe via via il poliedro delle soluzioni, fino ad arrivare a $\mathcal{P}_I$, infatti esiste sicuramente un valore k finito tale che $P_k = \mathcal{P}_I$.

La teoria svolta vale solo se P_0 è un poliedro razionale, cioè se A e b hanno componenti tutte razionali. Comunque, non è sempre conveniente usare tale metodo. Ne esiste infatti uno più efficace, il quale nasce dall'osservazione che se con l'algoritmo del Simplesso si risolve il rilassamento lineare su P_0 e la soluzione trovata non è intera, allora dal dizionario ottimo relativo è immediato ricavare una disuguaglianza che risulta valida per gli elementi di $\mathcal{P}_I$ e violata dall'ottimo frazionario del rilassamento e che aggiunta alla formulazione di questo porta ad un valore ottimo più vicino a quello intero.

Supponendo di aver ottimizzato il problema su P_0 e di trovarsi nel caso descritto in cui la soluzione è frazionaria, indichiamo il dizionario ottimo nel modo seguente:

$$z = c_0 + \sum_{j \in \mathcal{N}} c_j x_j$$

$$x_i = b_i - \sum_{j \in \mathcal{N}} a_{ij} x_j$$

dove i coefficienti a_{ij} sono frazionari ed $\mathcal{N}$ è l'insieme delle variabili non in base.

In questo caso, la disuguaglianza

$$x_i + \sum_{j \in \mathcal{N}} \lfloor a_{ij} \rfloor x_j \leq \lfloor b_i \rfloor$$

è certamente soddisfatta da tutti i vettori interi che siano soluzione del problema originale.

Si può anche procedere in un altro modo. Allo scopo definiamo la parte frazionaria di a_{ij} come $f_{ij} = a_{ij} - \lfloor a_{ij} \rfloor$ e poniamo $b_i - \lfloor b_i \rfloor = f_i$. Allora, sottraendo alla disuguaglianza trovata il vincolo del dizionario, si ottiene

$$\sum_{j \in \mathcal{N}} \lfloor a_{ij} \rfloor x_j - \sum_{j \in \mathcal{N}} a_{ij} x_j \leq \lfloor b_i \rfloor - b_i$$

da cui

$$\sum_{j\in\mathcal{N}} \Big(\lfloor a_{ij}\rfloor - a_{ij}\Big)x_j \leq \lfloor b_i\rfloor - b_i$$

e quindi

$$-\sum_{j\in\mathcal{N}} f_{ij}x_j \leq -f_i$$

o equivalentemente

$$\sum_{j\in\mathcal{N}} f_{ij}x_j \geq f_i.$$

Se la soluzione trovata non è intera, allora qualche b_i è frazionario e quindi i corrispondenti valori di f_i sono positivi.

Nella soluzione di base si ha $x_j = 0$ (essendo x_j la variabile fuori base) e quindi nella soluzione di base corrente si ha sempre

$$\sum_{j\in\mathcal{N}} f_{ij}x_j = 0.$$

Allora, la disuguaglianza

$$\sum_{j\in\mathcal{N}} f_{ij}x_j \geq f_i$$

rappresenta un iperpiano soddisfatto da $\mathcal{P}_I$ ma violato dalla soluzione corrente del rilassamento.

Da tale disuguaglianza si trovano i tagli per il problema. In effetti, dalle relazioni $f_{ij} \geq 0$ e $x_j \geq 0$ si ottiene la relazione $f_{ij}x_j \geq 0$ e di conseguenza si ha :

$$-\sum_{j\in\mathcal{N}} f_{ij}x_j \leq 0.$$

Infine, sommando tale relazione e la seguente

$$x_i + \sum_{j\in\mathcal{N}} a_{ij}x_j = b_i$$

ottenuta dal dizionario ottimo del rilassamento, si ottiene la disuguaglianza

$$x_i + \sum_{j\in\mathcal{N}} \lfloor a_{ij}\rfloor x_j \leq b_i.$$

A questo punto per ottenere un taglio, il taglio di Gomory, basta considerare la disuguaglianza

$$x_i + \sum_{j\in\mathcal{N}} \lfloor a_{ij}\rfloor x_j \leq \lfloor b_i\rfloor.$$

Dunque, il rilassamento del problema intero che abbia una soluzione ottima frazionaria offre immediatamente un taglio di Gomory per il problema originale.

2.4 Metodi basati sul Rilassamento Lagrangiano

L'applicazione della tecnica lagrangiana ai problemi di Ottimizzazione Combinatoria ha assunto importanza a partire dagli anni '70 e tutt'oggi è impiegata con successo nella messa a punto di algoritmi di soluzione per uno svariato numero di problemi.

L'intuizione alla base dell'approccio lagrangiano consiste nel vedere molti dei problemi di Ottimizzazione Combinatoria "difficili" come problemi "facili", complicati da un insieme, in genere di dimensione ridotta, di vincoli. Tale approccio consiste, infatti, nell'eliminazione dei vincoli "indesiderati" e nell'inserimento di questi nella funzione obiettivo in modo tale che, nella soluzione del problema rilassato, se ne tenga conto in qualche misura.

Si consideri un generico problema di ottimizzazione vincolata:

$$\min\{f(x) : g_i(x) \geq 0, i = 1, \ldots, m, x \in F \subseteq \mathbb{R}^n\},$$

dove $x \in \mathbb{R}^n$ è il vettore delle variabili del problema, F è la regione realizzabile, $f : F \longrightarrow \mathbb{R}$ è la funzione obiettivo e le funzioni $g_i : \mathbb{R}^n \longrightarrow \mathbb{R}$ rappresentano i vincoli del sistema che si vogliono eliminare.

Si dice allora *rilassamento lagrangiano* rispetto ai vincoli $g_i(x) \geq 0$ il seguente problema, detto anche *problema lagrangiano*:

$$L(\lambda) = \min \left\{ f(x) - \sum_{i=1}^{m} \lambda_i g_i(x) : x \in F \right\}$$

con il vettore λ, costituito dai coefficienti λ_i, non negativo, cioè $\lambda \geq 0$. La funzione $L(\lambda)$ è detta *funzione lagrangiana*.

Si può facilmente verificare che il rilassamento lagrangiano è effettivamente un rilassamento, infatti il suo insieme realizzabile contiene quello originale, essendo ottenuto da questo tramite cancellazione dei vincoli $g_i(x) \geq 0$.

Inoltre per la funzione obiettivo, se x è soluzione realizzabile del problema, si avrà $g_i(x) \geq 0$ e, pertanto,

$$f(x) - \sum_{i=1}^{m} \lambda_i g_i(x) \leq f(x)$$

essendo i coefficienti λ_i non negativi. I coefficienti λ si dicono *moltiplicatori di Lagrange*.

È anche possibile definire un rilassamento lagrangiano rispetto a vincoli di uguaglianza: se il problema originale ha la forma

$$\min\{f(x) : h_i(x) = 0,\ i = 1, \ldots, l,\ x \in F\},$$

un rilassamento lagrangiano rispetto ai vincoli $h_i(x)$ è dato dal seguente problema:

$$\min\left\{ f(x) - \sum_{i=1}^{l} \lambda_i h_i(x) : x \in F \right\}$$

dove i moltiplicatori di Lagrange λ_i non sono vincolati in segno. Si può facilmente verificare che anche questo è effettivamente un rilassamento.

A volte, come per tutti i rilassamenti, è possibile certificare l'ottimalità di una soluzione realizzabile attraverso il rilassamento lagrangiano. Infatti, vale il seguente

Teorema 1. *Se, per un dato vettore di moltiplicatori $\lambda^* \geq 0$ la soluzione del rilassamento lagrangiano x^* risulta realizzabile per il problema originale e se vale la relazione di complementarietà*

$$(\lambda^*)^T g(x^*) = 0,$$

allora x^* è soluzione ottima per il problema originale e λ^* risolve il problema $D = \max L(\lambda)$ (detto duale lagrangiano*).*

Dimostrazione. Come per il rilassamento lineare, anche nel caso lagrangiano vale la proprietà di limitazione inferiore: se x è realizzabile e $\lambda \geq 0$, allora

$$L(\lambda) = \min_{x \in F} \left\{ f(x) - \sum_{i=1}^{m} \lambda_i g_i(x) \right\} \leq f(x^*).$$

Ma se nell'ottimo del rilassamento la soluzione è realizzabile e vale la proprietà di complementarietà, allora si ha

$$L(\lambda^*) = f(x^*)$$

e pertanto x^* è soluzione ottima del problema dato. ■

Il rilassamento lagrangiano deve essere naturalmente più semplice da risolvere rispetto al problema originale, ed ha l'importante proprietà che per

λ fissato la soluzione ottima costituisce un limite sull'ottimo del problema originario: rispettivamente un limite inferiore se il problema considerato è di minimo, un limite superiore se il problema è di massimo. Inoltre il miglior limite inferiore è rappresentato dalla soluzione del duale lagrangiano.

Mostriamo ora un importante legame esistente tra i valori ottimi del duale lagrangiano e del rilassamento lineare di un programma lineare intero. Si immagini che l'insieme realizzabile di un problema di PLI sia rappresentato come

$$\min\{c^T x : x \in P \cap \mathbb{Z}^n\},$$

dove

$$P = \{x \in \mathbb{R}^n : Ax \geq b, x \in \mathcal{Q}\}$$

con $\mathcal{Q}$ poliedro descritto da altre equazioni e /o disequazioni lineari.
Consideriamo poi i valori ottimi

$$z^*_{PL} = \min\{c^T x : x \in P\}$$

$$z^*_{DL} = \max_{\lambda \geq 0} \left\{ \min_{x \in \mathcal{Q} \cap \mathbb{Z}^n} \left\{ c^T x + \lambda^T (b - Ax) \right\} \right\}.$$

rispettivamente del rilassamento lineare e del duale lagrangiano.
Vale allora il seguente

Teorema 2. *In generale si ha*

$$z^*_{PL} \leq z^*_{DL},$$

cioè il duale lagrangiano fornisce limitazioni inferiori non peggiori del rilassamento lineare, ottenuto eliminando i vincoli di interezza; tuttavia se

$$conv(\mathcal{Q} \cap \mathbb{Z}^n) = \mathcal{Q}$$

allora

$$z^*_{PL} = z^*_{DL}.$$

Questa proprietà afferma che se il problema "intero", di minimo, necessario per risolvere il duale lagrangiano, è inerentemente facile, corrisponde cioè al caso in cui i vertici di $\mathcal{Q}$ sono tutti a coordinate intere, allora la limitazione ottenuta dal duale lagrangiano è la stessa che si otterrebbe risolvendo il rilassamento lineare.

Va ricordato tuttavia che al di là di quanto sembra da questa proprietà, in molti problemi può essere utile anche un rilassamento lagrangiano per il quale $z^*_{PL} = z^*_{DL}$; spesso infatti la struttura del problema lagrangiano permette la risoluzione mediante algoritmi più efficienti di quanto sarebbe il metodo del simplesso applicato al rilassamento lineare del problema.

Inoltre nel corso delle iterazioni il rilassamento lagrangiano può portare a soluzioni realizzabili che costituiscono approssimazioni all'ottimo. Tali approssimazioni possono essere utili sia nel caso di terminazione prematura dell'algoritmo, sia all'interno di una procedura di tipo branch-and-bound.

In effetti, uno degli impieghi principali della tecnica lagrangiana di rilassamento nella risoluzione di problemi di Ottimizzazione Combinatoria, è legata proprio al calcolo di limiti da utilizzare in schemi di enumerazione implicita, quali ad esempio algoritmi di branch-and-bound.

La bontà di una tecnica di rilassamento dipende principalmente da due fattori:

- la qualità del limite prodotto;
- la velocità di calcolo del limite.

La scelta di un rilassamento, in generale, non è immediata: può essere preferibile, ad esempio, scegliere limiti meno "forti", ma computazionalmente meno onerosi, piuttosto che limiti migliori a fronte di tempi di calcolo elevati.

Come già detto, per una data formulazione di PLI, il duale lagrangiano è maggiore od uguale al valore del rilassamento lineare, mentre i limiti lagrangiano e lineare sono uguali se la formulazione gode della proprietà di interezza. Tuttavia, anche in quest'ultimo caso, se il problema lagrangiano è risolvibile con poco sforzo computazionale, il calcolo del duale lagrangiano può essere più rapido e, quindi, può preferirsi al calcolo del rilassamento lineare.

D'altra parte, la tecnica lagrangiana è una metodologia che trae vantaggio dalla specificità del problema considerato (quali, ad esempio, proprietà particolari che rendono il problema rilassato semplice da risolvere in modo combinatorio). L'applicazione dell'approccio lagrangiano presuppone, quindi, non solo un'analisi di differenti formulazioni dello stesso problema, ma anche, per una stessa formulazione, un'analisi di diversi rilassamenti ottenibili effettuando il rilassamento lagrangiano di diversi insiemi di vincoli.

La scelta di una formulazione e di un rilassamento può effettuarsi sia con metodi analitici, sia in modo empirico. In generale, il rilassamento lagrangiano

di un minor numero di vincoli è preferibile, sia perché il problema lagrangiano, essendo di dimensioni ridotte, risulta essere più semplice da risolvere, sia perché meno sono i vincoli rimossi maggiore è la "vicinanza" del problema rilassato al problema originale.

Come semplice applicazione, consideriamo il problema di PL seguente:

$$(P) \qquad \min\{c^T x : Ax = b, Dx \leq d, x \geq 0\}$$

e supponiamo che il sistema di vincoli $Ax = b$ complichi il problema a tal punto da poter ottenere un problema facile da risolvere semplicemente non considerando tale insieme.

Allora, si può considerare il rilassamento lagrangiano di P rispetto al sistema di vincoli "indesiderati":

$$(P_\lambda) \qquad \min\{c^T x - \lambda^T (Ax - b) : Dx \leq d, x \geq 0\}.$$

Si può allora risolvere facilmente il problema P_λ per differenti vettori λ di parametri positivi e trarne informazioni su P.

Indichiamo con ξ una soluzione ottima di P e con $F = c^T \xi$ il valore ottimo di P, con ξ_λ una soluzione ottima di P_λ e con $F_\lambda = c^T \xi_\lambda - \lambda^T (A\xi_\lambda - b)$ il valore ottimo di P_λ.

Allora, essendo ξ una soluzione qualunque per P_λ, necessariamente si ha:

$$F_\lambda = c^T \xi_\lambda - \lambda^T (A\xi_\lambda - b) \leq c^T \xi - \lambda^T (A\xi - b) = c^T \xi = F.$$

Si può quindi limitare F grazie agli F_λ per ottenere un limite inferiore di F. Il vettore λ è reale e trovare il massimo esatto di F_λ è un problema di Ottimizzazione Continua. Sfortunatamente, la funzione F_λ non è differenziabile, e non si può dunque trovare il suo massimo con un calcolo di gradiente. Il metodo che si applica consiste nel calcolare un sub-gradiente, cioè una direzione (nello spazio dei moltiplicatori) nella quale il valore obiettivo di F_λ aumenta. Si osserva che il vettore $(A\xi_\lambda - b)$ è sempre un sub-gradiente dei moltiplicatori. Facendo evolvere i moltiplicatori in questo senso, si converge verso il massimo di F_λ. Secondo i casi, si cercherà il valore ottimo di tale massimo, o ci si accontenterà di una approssimazione (che resta comunque un limite inferiore di F).

Nel corso della risoluzione dei problemi P_λ, può succedere che la soluzione ξ_λ trovata sia una soluzione realizzabile del problema P, cioè che valga $A\xi_\lambda = b$.

In questo caso si ha:

$$F = c^T\xi \leq c^T\xi_\lambda = F_\lambda + \lambda^T(A\xi_\lambda - b) = F_\lambda$$

e quindi la soluzione ξ_λ è una soluzione ottima di P, dato che F_λ è limite inferiore e superiore di F.

2.5 Generazione di Colonne e Decomposizione di Dantzig-Wolfe

Sono passati quasi 50 anni da quando Ford e Fulkerson [45] proposero di trattare solo implicitamente le variabili di un *multicommodity flow problem*. Dantzig e Wolfe [23] pochi anni dopo aprirono la strada a questa idea fondamentale, sviluppando una strategia di estensione di un programma lineare in termini di colonne come aiuto nel processo di risoluzione.

Tale tecnica è nota con il nome di *generazione di colonne* ed è stata per la prima volta utilizzata da Gilmore e Gomory [49] come parte di un efficiente algoritmo euristico per la risoluzione del *cutting stock problem*.

Tale problema consiste nel determinare un modo di tagliare fogli standard di un certo materiale (per esempio, carta o legno) in varie forme minimizzando lo scarto. Supponiamo, ad esempio, di dover ricavare da assi di legno di lunghezza standard pari a L, determinate quantità di tavolette di m lunghezze diverse. Sia d_i il numero di tavolette di lunghezza l_i che devono essere prodotte, con $i = 1, \ldots, m$. Il problema consiste nel determinare il numero minimo di assi da tagliare per soddisfare la domanda di tavolette. Ciascun asse può essere tagliato in diversi modi, detti *schemi di taglio*. Sia n il numero complessivo dei possibili schemi di taglio. Se x_j è pari al numero di assi che dovranno essere tagliate secondo lo schema di taglio j-esimo, il problema può essere formulato come segue:

$$\begin{aligned}
\min \sum_{j=1}^{n} x_j \qquad & s.a & & \\
\sum_{j=1}^{n} a_{ij}x_j \qquad & \geq & d_i \qquad & i = 1, \ldots, m \\
x_j \qquad & \geq & 0 \qquad & j = 1, \ldots, n
\end{aligned}$$

dove a_{ij} sta ad indicare il numero di tavolette di lunghezza l_i prodotte tagliando un asse secondo lo schema j-esimo.

L'ostacolo che si incontra nell'affrontare tale problema sta nel fatto che il numero di schemi di taglio può essere estremamente elevato, tanto che può richiedere troppo tempo generare la matrice dei coefficienti del problema ed ancor più tempo risolverlo. Tuttavia, si sa che il numero di schemi di taglio effettivamente usati in una soluzione di base, e dunque anche in quella ottima, non supera m, valore che può essere anche molto piccolo. Il metodo di generazione di colonne permette di risolvere il problema senza generare *esplicitamente* l'intera matrice $A = (a_{ij})$, ma solamente quelle colonne che più delle altre risultano "importanti" per la minimizzazione del numero di assi.

La generazione di colonne è oggigiorno un metodo rilevante per la risoluzione di problemi in cui occorre far fronte ad un numero enorme di variabili.

L'inserimento di tale tecnica all'interno di una struttura di PL basata sul metodo di branch-and-bound, introdotto negli anni '80 da Desrochers, Desrosiers e Soumis [27] per la risoluzione del problema di vehicle routing con finestre temporali (VRPTW), è stato il passo decisivo nella progettazione di algoritmi esatti per una larga classe di problemi di PLI.

Descriviamo ora il metodo di generazione di colonne considerando il seguente programma lineare

$$\begin{aligned} z^* = \min & \left\{ \sum_{j \in J} c_j s_j \right\} \\ \text{s.a:} \quad & \sum_{j \in J} a_j s_j \geq b \\ & s_j \geq 0, \quad j \in J. \end{aligned} \tag{2.1}$$

Nel seguito indicheremo tale programma con il nome di *Master Problem* (MP).

Ad ogni iterazione del metodo del simplesso si cerca una variabile fuori base da far entrare in base, secondo una stima del "prezzo" di tale operazione.

In questa fase, nota con il termine di *pricing step*, dato un vettore u non negativo si vuol trovare

$$\arg\min\{\bar{c}_j = c_j - u^T a_j : j \in J\}. \tag{2.2}$$

La complessità richiesta da tale fase è chiaramente $O(|J|)$ ed un'esplicita ricerca può essere computazionalmente impossibile quando $|J|$ è enorme.

In pratica, si lavora con un sottoinsieme ragionevolmente piccolo $J' \subseteq J$ di colonne, che dà origine alla nozione di *Restricted Master Problem* (RMP).

Assumendo per il momento la realizzabilità del corrente RMP, siano $\bar{s}$ e $\bar{u}$ rispettivamente la soluzione primale e quella duale del RMP. Quando le colonne a_j, con $j \in J$, sono implicitamente date come elementi di un insieme $\mathcal{A} \neq \emptyset$, ed i rispettivi coefficienti di costo c_j possono essere calcolati da a_j per mezzo di una funzione $c : \mathcal{A} \longrightarrow \mathbb{Q}$, allora il sottoproblema (*subproblem*)

$$\bar{c}^* = \min\{c(a) - \bar{u}^T a : a \in \mathcal{A}\} \tag{2.3}$$

restituisce una risposta al problema di pricing, rappresenta cioè un "oracolo" per tale problema.

Supponiamo che il sottoproblema sia realizzabile, altrimenti il MP sarebbe vuoto. Se la soluzione del sottoproblema è non negativa, cioè $\bar{c}^* \geq 0$, allora nessun coefficiente di costo ridotto $\bar{c}_j$ ha valore negativo e la soluzione $\bar{s}$ (inserita in $\mathbb{R}^{|J|}$) risolve all'ottimo anche il MP. Altrimenti, cioè nel caso in cui $\bar{c}^* < 0$, si allarga il RMP aggiungendo una colonna derivata dalla risposta dell'oracolo, cioè dalla soluzione ottima del sottoproblema, e si ripete con la riottimizzazione del RMP. Per il suo ruolo nell'algoritmo, il sottoproblema (2.3) è anche detto *problema di generazione* o *generatore di colonne*.

Naturalmente, un ruolo fondamentale in questo algoritmo è giocato dal problema di pricing, il quale eredita la difficoltà del ricercare virtualmente tutte le colonne non in base.

Si potrebbe avere l'impressione di aver semplicemente rinviato la difficoltà, ma il vantaggio di risolvere il problema di ottimizzazione in (2.3) piuttosto di un'enumerazione in (2.2) diventa sempre più evidente quando si ricorda che i vettori $a \in \mathcal{A}$ rappresentano spesso oggetti combinatori come percorsi, insiemi o permutazioni. Allora, l'insieme $\mathcal{A}$ e l'interpretazione di costo sono naturalmente definiti in queste strutture ed abbiamo a disposizione un'informazione preziosa su come "appaiano" le possibili colonne.

Per quanto riguarda la convergenza della generazione di colonne, notiamo che ogni $a \in \mathcal{A}$ è generato al massimo una volta dato che nessuna variabile in un RMP ottimo ha un costo ridotto negativo. Quando ci si occupa di un qualche insieme $\mathcal{A}$ finito (come è usualmente il caso nell'Ottimizzazione Combinatoria), l'algoritmo di generazione di colonne è esatto. In caso contrario, possono essere usate delle tecniche particolari per garantire una soluzione all'interno di una qualche tolleranza specificata.

Notiamo che, se $\bar{z}$ rappresenta il valore ottimo della funzione obiettivo del RMP, dalla teoria della dualità si ha $\bar{z} = \bar{u}^T b$. È molto interessante anche notare che quando un limite superiore $\kappa \geq \sum_{j \in J} s_j$ vale per una soluzione ottima del problema principale, si stabilisce non solo un limite superiore su z^* in ogni iterazione, ma anche un limite inferiore:

$$\bar{z} + \bar{c}^* \kappa \leq z^* \leq \bar{z}. \tag{2.4}$$

Così, si può verificare la qualità della soluzione in ogni istante durante l'algoritmo di generazione di colonne. Nell'ottimo di (2.1), vale $\bar{c}^* = 0$ per le variabili in base ed i limiti sono stretti. Il limite inferiore (2.4) è computazionalmente economico e prontamente disponibile quando il problema (2.3) è risolto all'ottimo. Quando $c \equiv 1$, κ può essere rimpiazzato da z^* e si ottiene un miglior limite inferiore, cioè $\bar{z}/(1 - \bar{c}^*) \leq z^*$.

Per concludere, descriviamo brevemente il classico principio di decomposizione in PL, dovuto a Dantzig e Wolfe [23], noto con il nome di *principio di decomposizione di Dantzig-Wolfe*, che oggi è parte del repertorio standard della Programmazione Matematica.

Consideriamo un programma lineare

$$\begin{aligned} z^* = \min & \, c^T x \\ \text{s. a:} \quad & Ax \geq b \\ & Dx \geq d \\ & x \geq 0 \end{aligned} \tag{2.5}$$

detto *formulazione compatta.*

Sia

$$P = \{x \in \mathbb{R}^n : Dx \geq d, x \geq 0\} \neq \emptyset.$$

I ben noti teoremi di Minkowski e Weyl (vedere [90]) ci permettono di rappresentare ogni $x \in P$ come combinazione convessa di punti estremi $\{p_q\}_{q \in \mathcal{Q}}$ sommata ad una combinazione non negativa di raggi estremi $\{p_r\}_{r \in \mathcal{R}}$ di P, cioè,

$$x = \sum_{q \in \mathcal{Q}} p_q s_q + \sum_{r \in \mathcal{R}} p_r s_r, \quad \text{con} \sum_{q \in \mathcal{Q}} s_q = 1, \; s \in \mathbb{R}_+^{|\mathcal{Q}|+|\mathcal{R}|} \tag{2.6}$$

dove gli insiemi di indici $\mathcal{Q}$ e $\mathcal{R}$ sono finiti.

Sostituendo tale espressione ad x in (2.5) ed applicando le trasformazioni lineari $c_j = c^T p_j$ e $a_j = A p_j$ con $j \in \mathcal{Q} \cup \mathcal{R}$, si ottiene una equivalente

formulazione estesa:

$$\begin{aligned} z^* = \min & \left\{ \sum_{q\in\mathcal{Q}} c_q s_q + \sum_{r\in\mathcal{R}} a_r s_r \right\} \\ \text{s. a:} \quad & \sum_{q\in\mathcal{Q}} a_q s_q + \sum_{r\in\mathcal{R}} a_r s_r \geq b \\ & \sum_{q\in\mathcal{Q}} s_q = 1 \\ & s \geq 0. \end{aligned} \tag{2.7}$$

Tale formulazione ha un numero $|\mathcal{Q}| + |\mathcal{R}|$ molto grande di variabili, ma possibilmente meno righe di (2.5).

L'equazione $\sum_{q\in\mathcal{Q}} s_q = 1$ è detta *vincolo di convessità*. Se $x \equiv 0$ è realizzabile per P in (2.5) a costo zero, il vincolo di convessità viene sostituito da $\sum_{q\in\mathcal{Q}} s_q \leq 1$.

Sebbene la formulazione compatta e quella estesa siano equivalenti visto che danno lo stesso valore ottimo z^* della funzione obiettivo, i rispettivi poliedri non risultano equivalenti dal punto di vista combinatorio (vedere [2] e [80]).

Ora, (2.7) è un MP speciale, in cui la funzione obiettivo è lineare e l'insieme delle colonne è implicitamente definito dai punti e dai raggi estremi di un poliedro convesso P. Possiamo risolverlo con la generazione di colonne. Il corrispondente RMP è interpretato come (2.7), con sottoinsiemi correnti $\mathcal{Q}' \subseteq \mathcal{Q}$, $\mathcal{R}' \subseteq \mathcal{R}$.

Data una soluzione duale ottima $\bar{u}$, $\bar{v}$ sul RMP, dove la variabile v corrisponde al vincolo di convessità, il sottoproblema (2.3) nella classica decomposizione di Dantzig-Wolfe consiste nel determinare

$$\min_{j\in\mathcal{Q}\cup\mathcal{R}} \{c_j - \bar{u}^T a_j - \bar{v}\}.$$

Dalla nostra precedente trasformazione lineare si ottiene:

$$\bar{c}^* = \min\{(c^T - \bar{u}^T A)x - \bar{v} : Dx \geq d, x \geq 0\}. \tag{2.8}$$

Questo è ancora un programma lineare. Abbiamo assunto $P \neq \emptyset$.

Quando $\bar{c}^* \geq 0$, non esiste alcuna colonna di costo ridotto negativo e l'algoritmo termina. Quando $\bar{c}^* < 0$ e finito, la soluzione ottima a (2.8) è un punto estremo p_q di P e si aggiunge la colonna $[c^T p_q, (Ap_q)^T, 1]^T$ al RMP. Quando $\bar{c}^* = -\infty$, si identifica un raggio estremo p_r di P con una soluzione omogenea del (2.8) e si aggiunge la colonna $[c^T p_r, (Ap_r)^T, 0]^T$ al RMP.

Da (2.4) e dal vincolo di convessità, ad ogni iterazione si ottiene

$$\bar{z} + \bar{c}^* \leq z^* \leq \bar{z},$$

dove $\bar{z} = \bar{u}^T b + \bar{v}$ è ancora il valore ottimo della funzione obiettivo del RMP. Notiamo che il limite inferiore è valido anche nel caso in cui il sottoproblema generi un raggio estremo, cioè quando $\bar{c}^* = -\infty$.

2.6 Metodi di Programmazione Dinamica

La *Programmazione Dinamica* (PD) è una tecnica matematica utile spesso per prendere decisioni correlate fra loro e che consente di affrontare problemi apparentemente intrattabili, cioè dotati di un'apparente complessità esponenziale. Fu originariamente introdotta da Bellman nel 1957 per risolvere alcuni problemi decisionali a più stadi e problemi di controllo ottimo.

Originariamente la PD nasce per problemi dove si deve prendere una decisione per ogni istante di tempo, fornendo una procedura sistematica per determinare la combinazione di decisioni che rende massima l'efficienza totale.

La PD risolve problemi computazionali mettendo insieme le soluzioni di un certo numero di sottoproblemi, basandosi sul *principio di ottimalità*, secondo il quale in alcuni casi un problema opportunamente decomposto può essere risolto all'ottimo attraverso la soluzione ottima di ogni sottoproblema.

In questo contesto con il termine di "programmazione" non si intende un programma scritto in un qualche linguaggio di programmazione, ma un metodo di soluzione tabulare.

In generale, la PD viene adottata per risolvere problemi di ottimizzazione ma, al contrario della PL, non esiste una formulazione matematica standard "del" problema di PD; essa è piuttosto una tecnica generale per la risoluzione di taluni problemi, e le particolari equazioni usate devono essere sviluppate ed adattate in ogni singola situazione.

Gli algoritmi basati sulla PD suddividono il problema da risolvere in un certo numero di sottoproblemi, anche non necessariamente indipendenti, e risolvono ricorsivamente tali sottoproblemi. Nel caso di sottoproblemi non indipendenti, la risoluzione richiede di risolvere sottoproblemi comuni. In tal caso la PD risolve un problema una sola volta e ne memorizza la soluzione in una tabella per utilizzarla ogni volta che si ripresenti lo stesso problema. Alla

fine, gli algoritmi basati sulla PD fondono assieme le soluzioni ottenute per i sottoproblemi per determinare la soluzione del problema originario.

Lo sviluppo di un algoritmo di PD può essere diviso nelle seguenti quattro fasi:

1. Caratterizzazione della struttura di una soluzione ottima.
2. Definizione ricorsiva del valore di una soluzione ottima.
3. Calcolo del valore di una soluzione ottima con una *strategia bottom-up* (vale a dire, dal sottoproblema più "piccolo" a quello più "grande").
4. Costruzione di una soluzione ottima a partire dalle informazioni calcolate.

Le prime tre fasi definiscono la base di una soluzione di PD di un certo problema; nel caso in cui si debba determinare solamente il valore di una soluzione ottima, l'ultima fase non viene presa in considerazione. Tuttavia, quando è richiesto di costruire una soluzione ottima (cioè di determinare un assegnamento alle variabili del problema in modo che la funzione di costo assuma il valore ottimo trovato con le prime tre fasi), allora durante i calcoli della terza fase vengono memorizzate delle informazioni ausiliarie che facilitano l'esecuzione della quarta.

L'aspetto più complesso nella costruzione di un algoritmo di PD è l'analisi della struttura interna del problema, e la sua riduzione a sottoproblemi. Questa fase è molto delicata, perché una frantumazione eccessiva può condurre ad un numero di sottoproblemi troppo grande, ma d'altra parte senza un numero sufficiente di sottosoluzioni non sarà possibile ricostruire quella desiderata.

Una volta fatta questa analisi, viene costruita una tabella che contiene in ogni suo elemento una sottosoluzione (o i dati sufficienti a ricostruirla) ed una volta che la tabella è riempita, è facile desumere la soluzione del problema originale.

Le principali caratteristiche di un tipico problema risolvibile con la PD sono le seguenti:

1. Il problema si può dividere in stadi e, per ognuno di essi, è richiesta una politica decisionale.
2. Ogni stadio ha un numero di stati (cioè le varie possibili condizioni in cui il sistema può trovarsi) associati, numero che può essere finito o infinito.

3. L'effetto della politica di decisione ad ogni stadio è di trasformare lo stato attuale in uno stato associato allo stadio successivo.

4. Allo stadio attuale, la politica adottata per gli stadi precedenti non influenza quella dei successivi.

5. Il procedimento risolutivo comincia col trovare la politica ottima per ogni stato dello stadio precedente.

6. È disponibile una funzione ricorsiva che identifichi la politica decisionale ottima per ogni stato con n stadi rimanenti, dato il piano ottimo per ogni stato con $(n-1)$ stadi rimanenti.

7. Usando questa funzione ricorsiva, la procedura risolutiva si muove all'indietro stadio per stadio, trovando ogni volta la politica ottima per ogni stato di questo stadio, finché trova la soluzione ottima quando parte dallo stadio iniziale.

La PD si applica anche ai problemi di Ottimizzazione Combinatoria che hanno le seguenti caratteristiche:

1. Il problema si può scomporre in stadi. Ad ogni stadio è associata una decisione.

2. In ogni stadio k il problema può trovarsi in un numero finito di stati $(s_1^k, \ldots, s_{qk}^k)$.

3. È definita una funzione di costo $f_k(s_i^k)$ che dato lo stadio k e lo stato s_i^k fornisce il valore ottimo del sottoproblema definito dalle prime k decisioni, con il vincolo aggiuntivo che le decisioni prese portino il problema nello stato s_i^k.

4. È data una politica ottima (ricorsione) che dati gli stati dello stadio $k-1$ e le relative funzioni di costo, permette di determinare la decisione ottima dello stadio k-esimo necessaria per raggiungere un dato stato s_i^k.

Si noti che la ricorsione per determinare la decisione ottima x_k dello stadio k, per giungere in uno stato s_i^k, dipende direttamente solo dagli stati e costi dello stadio $k-1$ e non dalla sequenza di decisioni $x_1, \ldots, x_{k-1}$.

In pratica si valutano implicitamente tutti i cammini tra i vari stadi, sino a giungere a tutti i possibili stati dell'ultimo stadio. La soluzione ottima del

problema (se in forma di minimo) è quella che corrisponde allo stato s_i^n con minor costo.

La coppia (f^k, s_i^k) "riassume" la sequenza di decisioni necessarie per giungere in s_i^k e memorizza il valore della soluzione ottima. In realtà la DP non enumera tutte le sequenze di decisioni possibili, perché *collassa* (cioè, riassume) alcune sequenze in uno stesso stato.

2.7 Metodi Approssimati

Nell'affrontare un problema di ottimizzazione possono essere utilizzati vari approcci, a seconda sia della difficoltà specifica e delle dimensioni del problema in esame, sia degli obiettivi reali che si vogliono ottenere.

Una strada diversa dall'utilizzo di metodi esatti, che può diventare l'unica percorribile se le dimensioni e/o la complessità del problema sono elevate, è quella che consiste nel cercare soluzioni di tipo euristico, ottenute cioè applicando un algoritmo concepito in modo da produrre soluzioni che si sperano buone, ma senza garanzie a priori sulla vicinanza all'ottimo.

Un *algoritmo euristico*, o semplicemente un' *euristica*, deve essere in grado di produrre una soluzione in tempo relativamente breve.
Mentre chiaramente è possibile progettare euristiche specifiche per qualunque problema di Ottimizzazione Combinatoria, negli ultimi anni hanno acquisito importanza sempre maggiore alcuni approcci euristici di tipo generale, detti *metaeuristiche*. La struttura e l'idea di fondo di ciascuna metaeuristica sono sostanzialmente fissate, ma la realizzazione delle varie componenti dell'algoritmo dipende dai singoli problemi.

Tutti i diversi approcci metaeuristici possono vedersi in realtà in modo omogeneo, come generalizzazioni di un unico approccio fondamentale, che è quello della *ricerca locale*. La ricerca locale si basa su quello che è, per certi versi, l'approccio più semplice ed istintivo: andare per tentativi. Va comunque detto che a fronte di questa relativa semplicità, l'applicazione di una qualunque metaeuristica ad un problema di Ottimizzazione Combinatoria richiede comunque una messa a punto accurata e talvolta molto laboriosa.

Consideriamo un problema di minimizzazione ed una sua soluzione realizzabile x, con associato il valore della funzione obiettivo $f(x)$. La ricerca locale consiste nel definire un *intorno* di x (nella terminologia della ricerca locale, detto anche *vicinato*), e nell'esplorarlo in cerca di soluzioni migliori, se ve ne

sono. Se, in questo vicinato di x, si scopre una soluzione y per cui $f(y) < f(x)$, allora ci si sposta da x a y e si riparte con l'esplorazione del vicinato. Se invece nel vicinato di x non si scopre nessuna soluzione migliore, allora vuol dire che x è un *minimo locale* .

Nella ricerca locale classica, arrivati ad un minimo locale , l'algoritmo si ferma e restituisce questo minimo come soluzione al problema. Ovviamente non si ha alcuna garanzia in generale che tale valore costituisca una soluzione ottima del problema; anzi, tipicamente esso può essere anche molto distante dall'ottimo globale. Le metaeuristiche, in effetti, sono state concepite proprio per cercare di ovviare al problema di rimanere "intrappolati" in un minimo locale.

Un grosso pregio della ricerca locale sta nella sua semplicità concettuale e realizzativa, pur consentendo tale metodo di ottenere risultati molto interessanti.

Negli ultimi venti anni la ricerca si è indirizzata verso approcci euristici che generalizzano la ricerca locale cercando di ovviare ai suoi principali difetti e proprio da tale ricerca sono nate le tecniche metaeuristiche, oggi molto apprezzate e largamente utilizzate.

Spesso queste metaeuristiche traggono ispirazione (se non legittimazione) da alcune analogie con la natura fisica. Ad esempio, nella tecnica metaeuristica nota con il nome di *Simulated Annealing* (SA) si paragona il processo di risoluzione di un problema di Ottimizzazione Combinatoria, in cui si passa iterativamente da una soluzione realizzabile ad un'altra via via migliorando la funzione obiettivo, al processo con cui un solido, raffreddandosi, si porta in configurazioni con via via minor contenuto di energia.

Ricercare il minimo globale di una funzione di costo con molti gradi di libertà è un problema molto complesso se questa ammette un grande numero di minimi locali. Uno dei principali obiettivi dell'ottimizzazione è proprio quello di evitare di rimanere intrappolati in un minimo locale. Questo è uno dei limiti più grandi delle tecniche di ricerca locale.

I metodi basati sul SA applicano un meccanismo probabilistico che consente alla procedura di ricerca di fuggire da questi minimi locali. L'idea è quella di accettare, in certi casi, oltre alle transizioni che corrispondono a miglioramenti nella funzione obiettivo, anche quelle transizioni che portano a peggioramenti nel valore di questa funzione di valutazione. La probabilità di accettare tali deterioramenti varia nel corso del processo di ricerca, e discende lentamente verso

zero. Verso la fine della ricerca, quando vengono accettati solo miglioramenti, questo metodo diventa una semplice ricerca locale. Tuttavia, la possibilità di transitare in punti dello spazio di ricerca che deteriorano la soluzione ottima corrente, consente di abbandonare i minimi locali ed esplorare meglio l'insieme delle soluzioni realizzabili.

Altra analogia con la natura fisica è palese negli *Algoritmi Genetici* (AG), approcci metaeuristici in cui un insieme di soluzioni realizzabili è visto come un insieme di individui di una popolazione che, accoppiandosi e combinandosi tra loro, danno vita a nuove soluzioni che, se generate in base a criteri di miglioramento della specie, possono risultare migliori di quelle da cui sono state generate.

Gli AG fanno parte della categoria dei cosiddetti *weak methods*, costituita da quei metodi di risoluzione di problemi che si basano su poche assunzioni (o conoscenze) relative alle particolari strutture e caratteristiche dei problemi stessi, motivo per il quale questi metodi sono applicabili ad una vasta classe di problemi. Gli AG rientrano pienamente in questa categoria, dal momento che essi sono in grado di compiere una efficiente ricerca anche quando tutta la conoscenza a priori è limitata alla sola procedura di valutazione che misura la qualità di ogni punto dello spazio di ricerca (misura data, ad esempio, dal valore della funzione obiettivo).

Questa caratteristica conferisce a tali algoritmi una grande robustezza, ovvero una grande versatilità che li rende applicabili a diversi problemi, al contrario dei metodi convenzionali che, in genere, non trovano altra applicazione che quella relativa al problema per cui sono stati ideati.

Un'altra tecnica metaeuristica largamente usata nella risoluzione di problemi di Ottimizzazione Combinatoria è la *Tabu Search* (TS). Nella ricerca locale classica, ogni qual volta si esplora il vicinato (o intorno) di una soluzione, l'unica informazione relativa alla storia dell'algoritmo fino a quel momento è la migliore soluzione corrente ed il corrispondente valore della funzione obbiettivo. L'idea della TS è invece quella di mantenere una memoria di alcune informazioni sulle ultime soluzioni visitate, orientando la ricerca in modo tale da permettere di uscire da eventuali minimi locali.

L'idea di base della TS è dunque quella di utilizzare le informazioni sulle ultime mosse effettuate, memorizzate in una coda detta *tabu list*, per proibire quelle mosse per un certo tempo al fine di prevenire ricadute in punti già visitati, evitando così il ciclaggio.

La TS è oggi da molti considerata la metaeuristica che consente di avere il miglior rapporto tra la qualità delle soluzioni e lo sforzo computazionale (includendo in questo anche la complessità computazionale).

La maggior parte degli analisti di OR hanno abitualmente considerato di usare le tecniche euristiche per ottenere buone soluzioni per problemi considerati troppo complessi per essere in grado di ottenere soluzioni ottime.

Tuttavia, numerosi codici commerciali, il cui scopo è o di provare l'ottimalità o di terminare non appena una soluzione si dimostra essere all'interno di una specificata tolleranza di ottimalità, applicano abitualmente algoritmi euristici durante tutta la procedura in modo da trovare prima possibile dei buoni limiti nell'algoritmo.

Così, le euristiche servono a due scopi davvero importanti: fornire buone soluzioni a problemi per i quali algoritmi esatti sono incapaci di provare l'ottimalità in un tempo ragionevole ed "aiutare" un algoritmo esatto nella ricerca di una soluzione.

I metodi esatti, come quello di branch-and-bound o la PD, e le tecniche di ricerca locale sono viste tradizionalmente come due approcci generali ma distinti per la risoluzione efficiente di problemi di Ottimizzazione Combinatoria, ognuno con i propri vantaggi e svantaggi.

È solo piuttosto recentemente che sono stati proposti veri algoritmi ibridi, che prendono idee da ambedue i campi cercando di combinarli al meglio per superare i rispettivi difetti. Nel caso di un problema di minimizzazione, per esempio, il più semplice approccio ibrido consiste nell'uso di un algoritmo di ricerca locale per calcolare limiti superiori che permettano una potatura immediata di soluzioni non ottime nella procedura guidata da un algoritmo esatto.

Capitolo 3

Programmazione con Vincoli

3.1 Introduzione

Come detto nei precedenti capitoli, larga parte dei problemi normalmente studiati nell'area della Ricerca Operativa (OR), frequentemente affrontati anche con tecniche di Intelligenza Artificiale (brevemente AI, dall'inglese *Artificial Intelligence*), come problemi di scheduling o di allocazione di risorse, sono problemi combinatori molto difficili, spesso $\mathcal{NP}$–ardui. Nonostante le difficoltà intrinseche, sembra che per la maggior parte di essi esistano algoritmi in grado di risolverli con una complessità ragionevole nel caso peggiore.

L'approccio di OR per la risoluzione di difficili problemi di Ottimizzazione Combinatoria è stato il solo usato per molto tempo e si basa su una rappresentazione matematica del problema, tipicamente modellato come programma lineare intero in cui le variabili sono legate da disuguaglianze lineari (vincoli lineari).

Un approccio tradizionale per risolvere tali problemi consiste nel progettare un algoritmo specifico che utilizzi tecniche enumerative o tecniche di ricerca locale. Questo approccio può condurre ad un'efficienza estrema, ma solitamente necessita di molto lavoro da parte del ricercatore.

Un altro possibile approccio consiste nel considerare il problema in una struttura generale, come quella della PLI, e nell'usare un risolutore dedicato per risolvere il modello lineare che rappresenta il problema. Tale approccio consente di ottenere risultati molto soddisfacenti, ma presenta anche qualche svantaggio. Solitamente, infatti, la riscrittura in termini di programma lineare intero tende ad allargare la dimensione del problema, a distorcere in qualche

misura il problema originale o ad ignorarne qualche proprietà interessante.

Idealmente, quindi, piacerebbe avere a disposizione una struttura alternativa per dichiarare una larga classe di problemi combinatori, soprattutto quei problemi per i quali non esistono ancora algoritmi efficienti, con un linguaggio abbastanza generale ed in grado di produrre l'utilizzo di regole specifiche su un problema, al fine di accrescere l'efficienza del processo risolutivo.

Negli ultimi anni, per modellare e risolvere problemi difficili di Ottimizzazione Combinatoria, come problemi di pianificazione, di scheduling e di routing, è stato utilizzato con successo un nuovo paradigma di programmazione noto come *Programmazione con Vincoli* (brevemente CP, dall'inglese *Constraint Programming*).

In effetti, quello della CP è un approccio alternativo a quello tipico della OR in cui il processo di programmazione è limitato ad una generazione di richieste (vincoli) ed alla generazione di una soluzione a queste richieste per mezzo di metodi generali o metodi di dominio specifico. Si tratta di una emergente tecnologia software per la descrizione e la risoluzione di grandi problemi, in particolare combinatori, specialmente nelle aree della pianificazione e dello scheduling.

Il quadro della CP permette, come quello della PLI largamente studiato nel capitolo precedente, di modellare problemi di realizzabilità (in cui si vuole trovare una soluzione realizzabile qualsiasi) e problemi di ottimizzazione (in cui si vuole trovare la soluzione migliore, quella che massimizza o minimizza un certo criterio) con un insieme di variabili da assegnare rispettando formule che vincolano i valori di tali variabili. La differenza con la PLI viene essenzialmente dal linguaggio offerto per l'espressione dei vincoli. Tuttavia, bisogna dire che nonostante possa essere definita una funzione obiettivo sulle variabili del problema, i risolutori puri di CP non sono in grado di trattare efficientemente tale funzione, dato che in generale essi eseguono potenti riduzioni di domini basate su ragionamenti di realizzabilità, ma non su ragionamenti di ottimalità.

I metodi generali di CP usualmente riguardano tecniche di riduzione dello spazio di ricerca (*tecniche di propagazione di vincoli*) e specifici metodi di ricerca. In contrasto, i metodi di dominio specifico si presentano spesso nella forma di algoritmi con finalità speciali o pacchetti specializzati, comunemente detti *risolutori di vincoli* (*constraint solvers*).

I problemi che sembrano poter essere risolti in un modo naturale per mezzo della CP sono usualmente quelli per i quali mancano efficienti algoritmi (per

esempio, problemi computazionalmente intrattabili) o per i quali la formalizzazione in termini di leggi conduce ad un più flessibile stile di programmazione in cui le dipendenze tra la variabili in questione possono essere espresse in una forma più generale.

Quando si risolvono tali problemi, bisogna procedere con diverse iterazioni e modellarli per mezzo di vincoli spesso può essere vantaggioso. Infatti, l'appropriata modifica del programma può essere spesso ottenuta modificando qualche vincolo o alterando una appropiata componente fissata del programma (per esempio, quella che definisce il metodo di ricerca adottato).

Dal punto di vista della modellazione (*modelling*) di un problema combinatorio, l'espressione sotto forma di vincoli è generalmente più leggibile (grazie all'utilizzo di oggetti e di campi) e più concisa (grazie all'utilizzo di vincoli simbolici che permettono di esprimere relazioni complesse tra variabili) in CP rispetto a quanto avvenga in PL. I vincoli possono essere di natura simbolica o di natura numerica, operanti su variabili reali o intere.

Un aspetto aggiuntivo introdotto dalla CP è che il fatto di modellare un problema per mezzo di vincoli conduce ad una rappresentazione del problema stesso per mezzo di relazioni. In un certo numero di circostanze, tale rappresentazione permette di usare lo stesso programma per diversi fini e questo può essere chiaramente molto utile. Nei linguaggi di programmazione convenzionali, invece, le relazioni devono essere convertite in funzioni e quindi questa possibilità viene persa.

Un potente strumento di modellazione nella CP è rappresentato dai *vincoli globali*, vincoli simbolici che rappresentano astrazioni che permettono una dichiarazione del problema come una *composizione* di sottoproblemi.

Le caratteristiche della CP possono allora essere riassunte come segue:

- Il processo di programmazione consiste di due fasi: la generazione di una rappresentazione del problema per mezzo di vincoli e la generazione di una sua soluzione. Nella pratica, ambedue le fasi possono essere composte da diversi passi più piccoli e possono anche essere intervallate.

- La rappresentazione di un problema per mezzo di vincoli è molto flessibile dato che i vincoli possono essere aggiunti, rimossi o modificati.

- L'uso di relazioni per rappresentare vincoli offusca la differenza tra input e output. Questo rende possibile l'uso dello stesso programma per un certo numero di fini diversi.

La CP è quindi lo studio di sistemi computazionali basati su vincoli. L'idea di base della CP è quella di risolvere i problemi, noti con il nome di *problemi di soddisfazione di vincoli* (CSPs, da *Constraint Satisfaction Problems*), dichiarando vincoli (richieste) sul problema e poi cercando soluzioni che soddisfino tutti i vincoli.

Uno dei punti di forza del paradigma della CP è rappresentato dalla separazione netta tra:

- la descrizione del problema in termini di variabili e di vincoli da soddisfare;
- un processo di deduzione logica che agisce sui vincoli;
- gli algoritmi per risolvere il problema (processi di istanziazione di variabili).

Le ragioni per scegliere di rappresentare e risolvere un problema come un CSP, cioè con strumenti di CP piuttosto che con gli usuali strumenti di OR, sono essenzialmente la seguenti:

- La rappresentazione è spesso più vicina al problema originario: le variabili del CSP corrispondono direttamente alle entità del problema ed i vincoli possono essere espressi senza essere tradotti in disequazioni lineari. Questo produce una formulazione più semplice, la soluzione è più facile da capire e la scelta di buone euristiche che guidino la strategia di risoluzione risulta più semplice.
- Sebbene gli algoritmi per la risoluzione di un CSP siano davvero semplici, essi possono a volte trovare soluzioni più velocemente dei metodi di PLI.

L'origine e la base del campo della CP possono essere rintracciate nella ricerca di AI che negli anni '60 e '70 si è focalizzata sulla rappresentazione e sulla manipolazione di vincoli nei sistemi computazionali, ovvero su reti di vincoli e su problemi di soddisfazione vincolare.

Solo nell'ultima decade, comunque, è apparso chiaro come queste idee fornissero le basi per un potente approccio di programmazione, modellazione e risoluzione di problemi, e che i diversi sforzi per sfruttare tali idee potessero essere unificati sotto una comune struttura concettuale e pratica.

Il successo della CP è probabilmente dovuto alla sua introduzione nel campo della Programmazione Logica [74]. Alla fine degli anni '80, infatti, Jaffar e

Lassez [64] definiscono il paradigma della *Programmazione Logica con Vincoli* (CLP, dall'inglese *Constraint Logic Programming*) come uno schema generale che può essere specializzato su diversi domini. La CLP è un potente paradigma di programmazione che combina i vantaggi della Programmazione Logica e l'efficienza di risoluzione della Programmazione con Vincoli.

Più recentemente, sono stati sviluppati linguaggi basati sui vincoli indipendentemente dalla Programmazione Logica (come *ILOG Solver*, basato su C++), ma mantenendo uno stile di programmazione dichiarativo.

Tra i linguaggi di CP, quello su domini finiti, il CP(FD), è uno strumento efficace per modellare e risolvere problemi combinatori discreti. Esso modella un problema come un insieme di variabili che prendono i loro valori su un dominio finito di interi e che sono collegate da un insieme di vincoli. Nel seguito presenteremo la teoria risolutiva relativa a tale linguaggio riferendoci ad esso semplicemente con la sigla CP.

Negli ultimi anni, la CP ha attirato molta attenzione da parte degli esperti di molte aree della ricerca scientifica, visto il suo potenziale nella risoluzione di difficili problemi reali. L'attrattiva esercitata dalla CP non è basata solo su un forte fondamento teorico, ma anche su un assai diffuso interesse commerciale, in particolare nelle aree di modellazione di problemi eterogenei di ottimizzazione e soddisfazione.

Non sorprende quindi che la CP sia stata recentemente indicata dal ACM (*Association for Computing Machinery*) come una delle direzioni strategiche per la ricerca nel vasto campo denominato *Computer Science* (CS). Comunque, allo stesso tempo, essa risulta ancora una delle tecnologie meno conosciute e comprese.

Negli ultimi anni sia i ricercatori dell'area della OR che quelli di CP hanno cominciato a valutare la possibilità e l'utilità di una integrazione tra le tecniche dei due campi, per cercare di risolvere in modo più efficiente diverse classi di problemi di Ottimizzazione Combinatoria. Tale tentativo di integrazione occupa attualmente un gran numero di ricercatori di OR e di AI, ha già prodotto importanti e promettenti risultati e fa sperare in ulteriori passi in avanti.

Nel seguito del capitolo, dopo aver illustrato le basi teoriche della CP, analizzeremo le più importanti ed efficienti tecniche di integrazione con la OR presentate in letteratura.

3.2 Soddisfazione di Vincoli

Uno degli aspetti fondamentali della CP è la *soddisfazione di vincoli*, una teoria sviluppata nella comunità di AI che si occupa di problemi definiti su domini finiti. I problemi di soddisfazione di vincoli sono stati a lungo studiati in quanto rappresentativi della maggior parte delle applicazioni industriali attuali e giocano tuttora un ruolo centrale nel campo della AI.

Intuitivamente, un problema di soddisfazione di vincoli consiste in un insieme finito di relazioni su qualche dominio. Nella pratica si usa una sequenza finita di variabili a cui sono associati i relativi domini, cioè gli insiemi di valori possibili. Un vincolo su una sequenza di variabili è allora un sottoinsieme del prodotto cartesiano dei domini delle stesse variabili.

Per una definizione formale, un problema di soddisfazione di vincoli, cioè un *Constraint Satisfaction Problem* (CSP), è definito come:

- un insieme di variabili $X = \{x_1, \ldots, x_n\}$;
- un insieme di domini $\mathcal{D} = \{D_1, \ldots, D_n\}$, cioè per ogni variabile x_i, un insieme finito D_i di possibili valori;
- un insieme di vincoli $\mathcal{C}$ che restringono i valori che le variabili possono simultaneamente assumere.

Si noti che non è necessario che i valori siano un insieme consecutivo di interi (sebbene spesso lo siano) e nemmeno che siano valori numerici.

Una *soluzione ad un CSP* è un assegnamento di valori alle variabili del problema, un valore ad ogni variabile dal suo dominio, in modo tale che tutti i vincoli siano soddisfatti contemporaneamente. Se un CSP ha una soluzione, allora viene detto *consistente*, altrimenti *inconsistente*.

Nella risoluzione di un CSP gli scopi possibili sono tre, si può essere infatti interessati a trovare:

- solo una soluzione, con nessuna preferenza (problema di consistenza o di soddisfazione);
- tutte le soluzioni;
- un ottimo, o almeno una buona soluzione, data qualche funzione obiettivo definita in termini di alcune o di tutte le variabili del problema (problema di ottimizzazione).

Le soluzioni di un CSP possono essere trovate ricercando sistematicamente attraverso i possibili assegnamenti di valori alle variabili. I metodi di ricerca si dividono in due ampie classi: i metodi che attraversano lo spazio delle soluzioni parziali (o assegnamenti parziali di valori) e quelli che esplorano stocasticamente lo spazio degli assegnamenti completi di valori alle variabili.

In generale, ciascuno dei tre problemi sopra è computazionalmente intrattabile, cioè $\mathcal{NP}$–arduo. Intuitivamente, questo significa che nel caso peggiore può essere necessario considerare tutte le possibili istanziazioni di variabili prima di poter trovare una soluzione (o la soluzione migliore). Comunque, esistono alcune classi di problemi trattabili per i quali esistono algoritmi di risoluzione efficienti anche nel caso peggiore. Inoltre, nella pratica molte tecniche esibiscono una buona prestazione nel caso medio anche per le classi di problemi intrattabili.

Come già detto, la CP si occupa della risoluzione di problemi di soddisfazione di vincoli e quindi il problema reale da affrontare deve essere espresso come un CSP. Di conseguenza tale problema può essere formulato usando:

- un certo numero (finito) di variabili che variano su specifici domini ed un certo numero (finito) di vincoli su tali variabili;
- un qualche linguaggio in cui esprimere i vincoli.

Questa parte del processo di risoluzione del problema prende il nome di *modelling* ed è estremamente importante.

Dato che la maggior parte dei problemi reali possono essere modellati usando un insieme finito di variabili con dominio finito per ogni variabile, la soddisfazione di vincoli può essere largamente usata per problemi combinatori. Una volta completata la fase di modelling del problema, si passa alla risoluzione per mezzo di metodi generali, metodi di dominio specifico oppure una combinazione di entrambi. Da questo punto di vista la CP comprende diverse aree come l'Algebra Lineare, l'Ottimizzazione Globale, la Programmazione Lineare e quella Intera, etc.

Una importante parte della CP tratta lo studio di tecniche e metodi generali, che chiaramente possono essere specializzati a casi specifici. I metodi generali sono solitamente utili nei casi in cui non si conoscano buoni metodi specializzati per la risoluzione di un dato CSP.

Per qualche caso specifico sono stati spesso sviluppati metodi di risoluzione di vincoli con finalità speciali e sono ora disponibili nella forma di pacchetti o

librerie (come nel caso di sistemi di equazioni lineari per i quali sono disponibili algoritmi di algebra lineare).

Il fatto che per qualche specifico caso esistano metodi efficienti per la risoluzione di CSP e per altri casi non si conoscano metodi di questo tipo, richiede sistemi per la CP in cui siano disponibili sia metodi generali che metodi di dominio specifico. I primi sono usualmente presenti nella forma di costrutti specifici che supportano la ricerca, mentre i secondi sono spesso presenti nella forma di vari built-ins.

3.2.1 Ricerca Sistematica

Per risolvere un CSP è necessario cercare diversi assegnamenti di valori delle variabili che soddisfino i vincoli. Questo processo di assegnamento di valori alle variabili del problema è specificato dall'algoritmo di ricerca.

Dal punto di vista teorico, risolvere un CSP è banale usando l'esplorazione sistematica dello spazio delle soluzioni; non lo è però dal punto di vista pratico dove l'efficienza ha un ruolo fondamentale.

Anche se i metodi di ricerca sistematica presentati di seguito (senza miglioramenti aggiuntivi) sembrano davvero semplici e non efficienti, essi sono importanti perché costituiscono il fondamento di più avanzati ed efficienti algoritmi.

L'algoritmo di base della soddisfazione di vincoli, il più semplice, è detto *Generate-and-Test* (GT) e consiste nel generare (attraverso un *generatore*) tutti gli assegnamenti di valori delle variabili (detti *labelling*) e nel testare (attraverso un *tester*) se ogni assegnamento soddisfa i vincoli o meno. L'idea di base del GT è semplice: prima viene generato in modo casuale un labelling completo di variabili e, poi, se questo labelling soddisfa tutti i vincoli, la soluzione è trovata, altrimenti, viene generato un altro labelling e si ripete il ragionamento.

L'algoritmo GT è un generico algoritmo debole usato nel caso in cui ogni altro algoritmo abbia fallito. La sua efficienza è scadente a causa della mancanza di informazione del generatore, il quale genera il labelling completo senza alcuna informazione proveniente dal tester, e della conseguente scoperta ritardata delle inconsistenze.

Di conseguenza, ci sono due modi di migliorare l'efficienza del GT:

- Si fa in modo di avere un generatore di valutazioni "intelligente", nel senso di "informato", cioè che sia in grado di generare la valutazione completa in modo tale che il conflitto trovato dalla fase di test sia minimizzato.

- Si fonde il generatore con il tester, in modo che la validità del vincolo sia testata appena le rispettive variabili sono istanziate. Questo metodo è usato dall'approccio di *backtracking*.

Un altro metodo di ricerca sistematica per la risoluzione di un CSP è appunto il *BackTracking* (BT). L'idea è quella di istanziare valori delle variabili fino a trovare assegnamenti di valori che soddisfino tutti i vincoli. In pratica, il BT estende verso una soluzione completa una soluzione parziale che specifica valori consistenti per alcune delle variabili del problema, in modo incrementale, scegliendo ripetutamente un valore per un'altra variabile consistente con i valori nella corrente soluzione parziale.

Questo algoritmo attraversa lo spazio delle soluzioni parziali in modo depth-first e ad ogni passo estende una soluzione parziale (cioè, una istanziazione di variabile su un sottoinsieme di variabili che soddisfa tutti i relativi vincoli) assegnando un valore ad un'altra variabile. Quando si incontra una variabile tale che nessuno dei suoi valori è consistente con la corrente soluzione parziale, prende posto il BT e l'algoritmo considera nuovamente uno dei precedenti assegnamenti.

Più precisamente, all'inizio una delle variabili viene istanziata ad un valore specifico e poi, ad ogni passo, se ci sono $k-1$ variabili precedentemente istanziate, allora la successiva k-esima variabile è assegnata ad un valore che è consistente con le $k-1$ variabili precedentemente istanziate. Se non è possibile trovare un valore consistente per la k-esima variabile, allora l'algoritmo torna indietro (da cui il nome dell'algoritmo stesso) al punto più vicino in cui sia possibile generare una nuova ramificazione.

Come detto, si può vedere il BT come una fusione delle fasi di generazione e di test dell'algoritmo GT. Il labelling delle variabili è sequenziale ed appena tutte le variabili pertinenti ad un vincolo sono state instanziate, la validità del vincolo viene verificata. Se una soluzione parziale viola qualcuno dei vincoli, viene eseguito il BT sulla variabile più recentemente istanziata che ha ancora alternative possibili. Chiaramente, ogni volta che una istanziazione parziale viola un vincolo, il BT è in grado di eliminare un sottospazio dal prodotto cartesiano di tutti i domini delle variabili del problema.

Di conseguenza, il BT è assolutamente migliore dell'algoritmo GT, anche se la sua complessità temporale è ancora esponenziale per la maggior parte dei problemi non banali. Nei casi "fortunati" in cui l'algoritmo è in grado di assegnare con successo un valore ad ogni variabile senza la necessità di passi

di BT, la complessità diventa lineare nella dimensione del problema (spesso identificata con il numero delle sue variabili).

Il BT presenta altri svantaggi, i principali dei quali sono i seguenti:

- il thrashing, cioè ripetuti fallimenti dovuti alla stessa ragione;
- il lavoro ridondante, cioè valori contrastanti di variabili che non sono ricordati e che causano la ripetizione di un lavoro già eseguito;
- l'individuazione ritardata di un conflitto, cioè il conflitto non individuato prima che accada realmente.

Diversi miglioramenti del BT si sono concentrati su una o su ambedue le fasi dell'algoritmo: avanzare ad una nuova variabile e tornare indietro ad assegnamenti precedenti. Nel primo caso si parla di *schemi di look-ahead*, mentre nel secondo si parla di *schemi di look-back*.

I metodi per risolvere lo svantaggio del thrashing nel BT sono schemi di look-back, come il *backjumping*, il *backmarking* ed il *backchecking* che saranno presentati nel seguito. Maggiore attenzione si presta, invece, all'individuazione delle inconsistenze di una soluzione parziale con l'utilizzo di tecniche, note con il nome di *tecniche di consistenza*.

3.2.2 Tecniche di Consistenza

Un altro approccio per la risoluzione di un CSP è basato sulla rimozione di valori inconsistenti dai domini delle variabili fino ad ottenere una soluzione. I metodi di questo tipo sono detti *tecniche di consistenza*; si tratta di tecniche deterministiche, in opposizione alla ricerca non-deterministica.

Esistono diverse tecniche di consistenza (vedere [72] e [75]), ma per la maggior parte si tratta di tecniche incomplete. Quindi, tali tecniche sono raramente usate da sole per la risoluzione completa di un CSP.

I nomi delle tecniche di consistenza di base sono derivati dalle nozioni relative ai grafi. Infatti, il CSP è usualmente rappresentato come un *grafo di vincoli* (una rete) in cui i nodi corrispondono a variabili e gli archi sono etichettati da vincoli (come descritto in [79]).

Tale rappresentazione richiede che il CSP sia in una forma particolare, cioè che si tratti di un *CSP binario*, contenente quindi solo vincoli unari e binari (rispettivamente, contenenti solo una variabile e due variabili). Tuttavia,

questa restrizione non rappresenta un problema, dato che ogni arbitrario CSP può essere trasformato in un equivalente CSP binario (come mostrato in [5]) e che comunque notevoli sforzi sono stati fatti nella ricerca per estendere ad un CSP generale (in cui i vincoli non sono necessariamente binari) i risultati ottenuti per un CSP binario (vedere [8],[9],[10] e [11]).

Ogni CSP binario può quindi essere rappresentato da un grafo in cui i nodi sono le variabili e gli archi sono vincoli binari. Ogni nodo ha un dominio associato che rappresenta il dominio della variabile relativa.

La più semplice tecnica di consistenza è nota come *consistenza su nodi* (NC, dall'inglese *Node Consistency*). Tale tecnica tratta con i domini delle variabili rappresentati dai nodi del grafo rimuovendo da essi quei valori che sono inconsistenti con vincoli unari sulla rispettiva variabile. Si dice allora che un nodo del grafo di vincoli è consistente se e solo se il dominio della variabile rappresentata dal nodo non ha alcun valore che non soddisfi vincoli unari contenenti quella variabile. Un grafo è consistente su nodi se e solo se ogni nodo del grafo è consistente su nodi.

Un'altra tecnica di consistenza, la più usata, è la *consistenza su archi* (AC, dall'inglese *Arc Consistency*). Tale tecnica rimuove dai domini delle variabili quei valori che sono inconsistenti con vincoli binari. Allora, in un grafo di vincoli, si dice che un arco (x_i, x_j) è consistente se per ogni valore v_i del dominio di x_i esiste un corrispondente valore v_j nel dominio della variabile x_j per i quali il vincolo rappresentato dall'arco (x_i, x_j) è soddisfatto per la coppia di valori (v_i, v_j). Chiaramente, la tecnica AC è direzionale, cioè il fatto che l'arco (x_i, x_j) sia consistente non implica che lo sia anche l'arco (x_j, x_i). Un grafo è consistente su archi se e solo se ogni arco del grafo è consistente su archi.

La consistenza su archi assicura quindi che qualsiasi valore nel dominio di una variabile abbia un corrispondente "compagno" nel dominio di ogni altra variabile selezionata. Questo significa che ogni soluzione di un sottoproblema ad una variabile è estensibile in modo consistente ad un'altra variabile. La complessità temporale dell'algoritmo per la verifica della consistenza sugli archi è quadratica nella dimensione del problema.

Ancor più valori inconsistenti possono essere rimossi dalle tecniche di *consistenza su cammini* (PC, dall'inglese *Path Consistency*). Per ogni coppia di valori di due variabili x_i e x_j che soddisfino il relativo vincolo binario, la PC richiede che per ogni variabile lungo qualche cammino tra x_i e x_j esista un valore tale che tutti i vincoli binari nel cammino siano soddisfatti.

Questo algoritmo per la verifica della consistenza sui cammini assicura quindi che qualsiasi soluzione di un sottoproblema a due variabili sia estensibile ad una terza variabile e, come previsto, esso è più potente della consistenza su archi, scoprendo e rimuovendo molte più inconsistenze. Questo richiede però più tempo, infatti la complessità temporale in questo caso è cubica nella dimensione del problema.

Tutte le tecniche di consistenza menzionate sino ad ora sono coperte dalle nozioni generali di *K-consistenza* e di *K-consistenza forte*. Un grafo di vincoli si dice K-consistente se per ogni sistema di valori per $K-1$ variabili che soddisfino tutti i vincoli tra queste variabili, esiste un valore per un'arbitraria K-esima variabile tale che tutti i vincoli tra tutte le K variabili siano soddisfatti. Un grafo di vincoli è fortemente K-consistente se è J-consistente per tutti i $J \leq K$.

Chiaramente, la NC è equivalente alla 1-consistenza forte, la AC alla 2-consistenza forte e la PC alla 3-consistenza forte.

Esistono algoritmi per produrre un grafo di vincoli fortemente K-consistente per $K > 2$, ma nella pratica sono raramente usati per questioni di complessità. Sebbene questi algoritmi rimuovano più valori inconsistenti rispetto a qualsiasi algoritmo di consistenza su archi, essi in genere non eliminano la necessità di una ricerca.

Chiaramente, se un grafo di vincoli contenete N nodi è fortemente N-consistente, allora una soluzione al CSP può essere trovata senza alcuna ricerca. Ma la complessità nel caso peggiore dell'algoritmo per ottenere la N-consistenza in un grafo di vincoli di N nodi è esponenziale. Sfortunatamente, se un grafo è (fortemente) K-consistente per $K < N$, allora, in genere, la ricerca di BT non può essere evitata, cioè possono esistere ancora valori inconsistenti.

3.2.3 Propagazione di Vincoli

Anche se un programma di CP può utilizzare linguaggi (algebre) differenti per l'espressione dei vincoli, queste espressioni utilizzano tutte un meccanismo comune, vale a dire la *propagazione di vincoli*.

In generale, un programma di CP può essere risolto con una enumerazione, ma tale approccio non è ovviamente pratico dato che lo spazio di ricerca cresce esponenzialmente con la dimensione dei problemi. Così, questi programmi

beneficiano notevolmente della riduzione dello spazio di ricerca eseguito dalla propagazione di vincoli.

Gli algoritmi di propagazione di vincoli trasformano un dato problema in uno equivalente che è più esplicito, deducendo nuovi vincoli che vengono aggiunti al problema. In questo modo, essi possono produrre, espresse esplicitamente, alcune inconsistenze che erano implicitamente contenute nella specificazione del problema. Questi algoritmi sono molto interessanti dato che la loro complessità temporale nel caso peggiore è polinomiale nella dimensione del problema, e sono spesso davvero efficaci nello scoprire molte inconsistenze locali.

Da un punto di vista logico, si considera generalmente che un programma di CP sia composto di uno stato corrente e di operatori di trasformazione su tale stato. Lo stato viene rappresentato per mezzo di un insieme di formule logiche (definizione di oggetti, di domini, di relazioni) e le trasformazioni sono dei meccanismi di inferenza, capaci di dedurre nuovi fatti logici che non erano esplicitamente presenti nello stato, capaci anche di semplificare l'espressione dello stato corrente.

Da un punto di vista più operazionale, si può considerare che uno stato è rappresentato da un insieme di variabili alle quali sono associati domini di valori possibili. Tutti i vincoli elementari possono allora essere visti come operatori di trasformazione di stati, capaci di togliere dai domini correnti certi valori non ammissibili.

Il comportamento di ogni vincolo è definito come la soppressione di valori dai domini sotto certe condizioni. Questo permette la combinazione agevole di vincoli differenti, dato che il comportamento di ciascuno può essere descritto indipendentemente dagli altri.

Gli algoritmi di consistenza sono stati spesso usati anche come tecniche base di propagazione in risolutori di CP. Questi algoritmi non risolvono completamente un CSP, ma rimuovono inconsistenze locali che non possono apparire in alcuna soluzione consistente globale (ad esempio, un CSP può essere consistente su archi ma globalmente irrealizzabile).

Quindi, sia la ricerca sistematica che alcune tecniche di consistenza possono essere usate da sole per risolvere completamente il CSP, ma questo è fatto raramente. Una combinazione di ambedue gli approcci è un modo molto più comune di risolvere un CSP.

Si ottengono in questo modo meccanismi di riduzione di domini che pos-

sono essere più o meno efficaci. Citiamo, ad esempio, la *propagazione di valori* (la strategia forward checking che presenteremo nel seguito) ed il *filtraggio di domini* (di cui certe versioni sono dette tecniche di look-ahead) che toglie dai domini i valori incompatibili prima che alcun valore sia conosciuto definitivamente. Questo permette ad un vincolo di eliminare dal dominio di una delle sue variabili tutti i valori che sarebbero incompatibili con i domini di valori delle altre variabili del vincolo.

La propagazione di un vincolo su domini finiti corrisponde così ad una analisi di casi, più o meno elaborata, che permette di eliminare in anticipo certi valori dai domini delle variabili. Questi meccanismi di propagazione permettono di ripercuotere sui domini l'informazione contenuta nei vincoli. Ridurre i domini permette di istanziare variabili (quando il dominio è ridotto ad un singolo valore) o di individuare inconsistenze (quando un dominio diventa vuoto).

La fase completa di propagazione può esaminare i vincoli in modo isolato, gli uni dopo gli altri (si tratta della semplice consistenza sugli archi) o considerare più vincoli alla volta per ragionare sull'unione di vincoli (si può così considerare la consistenza su cammini).

Lo schema di *look-back* usa verifiche di consistenza tra variabili già istanziate e quindi risolve le inconsistenze quando accadono, senza poterle però prevedere.

L'algoritmo BT, precedentemente illustrato, è un semplice esempio di questo schema. Per evitare alcuni problemi, già menzionati, legati al BT vengono proposti altri tre schemi di look-back: il *BackJumping* (BJ), il *BackChecking* (BC) ed il *BackMarking* (BM).

Il BJ [46] è un metodo per evitare il thrashing, cioè un ripetuto fallimento per la stessa ragione, nel BT. Il controllo di BJ è esattamente lo stesso del BT, eccetto quando si rende necessario un indietreggiamento. Ambedue gli algoritmi scelgono una variabile alla volta e cercano un valore per questa variabile che assicuri che il nuovo assegnamento sia compatibile con i valori coinvolti fino a quel momento. Il BJ trova una inconsistenza e analizza la situazione al fine di identificare l'origine dell'inconsistenza. Esso usa i vincoli violati come guida per scoprire la variabile contrastante. Se vengono esplorati tutti i valori nel dominio, allora l'algoritmo BJ torna indietro alla più recente variabile contrastante. Questa è una delle differenze principali dall'algoritmo BT che invece torna indietro all' ultima variabile, quella immediatamente passata.

Gli schemi BC e BM [34] evitano invece il lavoro ridondante del BT. Il se-

condo discende dal primo ed entrambi rappresentano utili algoritmi per ridurre il numero di controlli di compatibilità, tenendo memoria delle coppie incompatibili di label ed evitando di considerare ulteriormente queste coppie nelle verifiche di consistenza. Il BM è un miglioramento del BC che riduce il numero di verifiche di compatibilità ricordando per ogni label quelli incompatibili più recenti. Inoltre, esso evita di ripetere verifiche di compatibilità che sono già state eseguite e che hanno avuto successo.

Tutti gli schemi look-back condividono lo svantaggio della scoperta ritardata del conflitto. Infatti, essi risolvono l'inconsistenza quando questa accade ma non sono in grado di prevederla. Per evitare tale svantaggio sono stati proposti gli schemi *look-ahead* che prevengono i conflitti futuri.

Quando si avanza per estendere una soluzione parziale, qualche calcolo (per esempio, la consistenza su archi) può portare a dover decidere quale variabile e quale dei valori della variabile scegliere come successivi al fine di diminuire la probabilità di incontrare situazioni in cui nessuno dei valori di una variabile risulti consistente con la soluzione parziale corrente.

Per decidere sulla variabile successiva, vengono usualmente preferite le variabili che vincolano maggiormente il resto dello spazio di ricerca, e quindi, viene scelta la variabile maggiormente vincolata. Per la selezione del valore, invece, è preferito il valore meno vincolante, al fine di massimizzare le future scelte per le istanziazioni.

Il più semplice esempio della strategia look-ahead è il *Forward Checking* (FC). Si tratta di una versione migliorata del semplice BT in cui inizialmente una variabile viene istanziata ad un valore dal suo dominio e poi, ripetutamente ad ogni passo, la variabile successiva viene istanziata ad un valore che sia consistente con gli assegnamenti precedenti.

In pratica l'algoritmo FC esegue una consistenza su archi tra coppie costituite da una variabile già istanziata ed una non ancora istanziata, cioè quando un valore viene assegnato alla corrente variabile, ogni valore nel dominio di una "futura" variabile che contrasta con questo assegnamento viene (temporaneamente) rimosso dal dominio. Quindi, FC mantiene l'invarianza che per ogni variabile senza label esiste almeno un valore nel suo dominio che è compatibile con i valori delle variabili istanziate e con label.

Diversamente dal BT, assegnando un valore alla variabile corrente, le consistenze sugli archi tra la variabile corrente e le variabili non istanziate sono mantenute. In questo modo, la variabile corrente non può assumere un valore

che causi un dominio vuoto per una delle variabili non istanziate. Se un tale valore non esiste, allora l'algoritmo torna indietro al punto in cui si può iniziare una nuova ramificazione.

L'algoritmo FC lavora molto più del BT quando ogni assegnamento viene aggiunto alla soluzione parziale corrente, tuttavia è quasi sempre una scelta migliore rispetto al BT. Inoltre, pur non essendo in grado di prevedere le inconsistenze, l'algoritmo FC riesce a scoprirle molto prima del BT.

Un altro metodo di look-ahead che rimuove ancor più inconsistenze del precedente è il *Partial Look Ahead* (PLA). Mentre l'algoritmo FC esegue solo le verifiche di consistenza tra la variabile corrente e le variabili future, il metodo PLA estende questa verifica di consistenza anche a variabili che non hanno una connessione diretta con le variabili con label.

Esiste anche un altro approccio, detto *Maintaining Arc Consistency* (MAC), che usa la consistenza su archi dopo ogni passo di labelling. Mentre nel BT la consistenza sugli archi è mantenuta solo tra la variabile istanziata ad ogni passo e le variabili precedentemente istanziate e nel FC tra le variabili istanziate e quelle non istanziate, nel MAC essa è mantenuta anche tra le variabili non istanziate. Questo permette al MAC di scoprire le inconsistenze tra le variabili future senza assegnare alcun valore. Di conseguenza, tale approccio produce un albero di ricerca più piccolo rispetto agli altri algoritmi di ricerca, presentando tuttavia lo svantaggio che potare il relativo spazio di ricerca richiede una computazione maggiore rispetto agli altri algoritmi e per questo esso può risultare non efficiente.

3.2.4 Ordinamento di Variabili e di Valori

Alla fine di un processo di propagazione di vincoli, ci si può ritrovare in tre situazioni diverse:

1. un dominio è diventato vuoto;
2. viene trovata una soluzione, cioè un assegnamento di valori per tutte le variabili;
3. qualche dominio contiene più di un valore.

Nell'ultimo caso, dato che la propagazione di vincoli non è completa, si rende necessaria una strategia di ricerca al fine di esplorare il rimanente albero di ricerca.

La strategia più comunemente usata ed anche la più semplice consiste nella selezione di variabili: viene considerata una variabile non istanziata e ad essa viene assegnato un valore tra quelli che appartengono al suo dominio. Fondamentalmente questo passo è piuttosto simile alle strategie di ramificazione usate negli algoritmi branch-and-bound, con la differenza che in questo caso non si può contare su un limite, cioè sul calcolo della soluzione ottima di un problema rilassato.

L'*ordinamento di variabili* è una strategia che specifica come la variabile successiva viene selezionata per un assegnamento di valori ad ogni passo di un algoritmo di ricerca. Esistono due tipi di ordinamento di variabili: l'ordinamento statico e quello dinamico.

Nell'ordinamento *statico* di variabili, l'ordinamento delle variabili per un assegnamento di valori è noto prima di iniziare la ricerca e non viene cambiato durante la ricerca stessa. Nell'ordinamento *dinamico* di variabili, la variabile candidata per l'assegnamento di valori è determinata durante il processo di ricerca in funzione dell'informazione parziale ad ogni passo, cioè la scelta della variabile successiva da considerare ad ogni punto del processo dipende dallo stato corrente della ricerca.

Ad esempio, una strategia di ordinamento dinamico di variabili può essere quella di selezionare ad ogni passo la variabile successiva non istanziata che ha il minor numero di valori nel proprio dominio.

Dato che l'ordinamento dinamico di variabili richiede ad ogni passo modifiche di domini di variabili, esso può essere applicato in algoritmi tipo FC e MAC. Tuttavia, l'ordinamento dinamico non è realizzabile per tutti gli algoritmi di ricerca. Per esempio, con il semplice BT nessuna informazione extra disponibile durante la ricerca potrebbe essere usata per produrre una scelta di ordinamento diverso da quello iniziale.

Esistono diverse strategie per ordinare le variabili, ma la più usata è certamente la *strategia first-fail*. Tale strategia richiede di selezionare la variabile successiva che è più probabile che fallisca, in altre parole viene selezionata la variabile con il minor numero di valori nel proprio dominio. In questo modo, i rami possono terminare a profondità ridotte e la dimensione dell'albero di ricerca diventa piccola. Allora un assegnamento di valori realizzabile per tutte le variabili può essere ottenuto più rapidamente.

Il ragionamento che sta alla base di tale principio è che se la corrente soluzione parziale non condurrà ad una soluzione completa, cioè se il cor-

rispondente ramo dell'albero di ricerca porterà ad un fallimento, allora è meglio scoprirlo prima possibile.

Questa euristica dovrebbe ridurre la profondità media dei rami nell'albero di ricerca facendo in modo che si trovino prima possibile i fallimenti. Dunque, anche se non ci sono soluzioni, nel qual caso è necessaria una ricerca completa per accertarsene, oppure se sono richieste tutte le soluzioni, la dimensione dell'albero di ricerca da esplorare è minore di quella che si ha utilizzando un ordinamento statico.

Quando tutte le variabili hanno lo stesso numero di valori, che è il caso di qualche problema nella fase iniziale, il principio first-fail indica che si dovrebbe ancora cercare di scegliere la variabile più difficile da istanziare ed una buona scelta è quella della variabile che partecipa al maggior numero di vincoli.

In effetti, se una variabile è coinvolta in più vincoli, allora diventa più difficile trovare un valore per tale variabile. Con questa strategia, i casi difficili sono trattati ancor prima durante il processo di ricerca; come conseguenza, le inconsistenze vengono scoperte prima e l'albero di ricerca risulta di dimensione più piccola.

Nel processo di ricerca, oltre la selezione di una variabile per l'assegnamento di valori, un'altra questione cruciale riguarda il valore da assegnare alla variabile corrente. La risposta a tale questione dipende dalla strategia di *ordinamento di valori* adottata dall'algoritmo di ricerca.

Infatti, a meno che i valori non vengano assegnati alle variabili semplicemente nell'ordine in cui appaiono nel dominio di ciascuna variabile, occorre avere un criterio per scegliere l'ordine in cui i valori devono essere assegnati alle variabili. Un diverso ordine di valori riordinerà i rami emananti da ogni nodo dell'albero di ricerca. Questo rappresenta un vantaggio se assicura che un ramo che conduce ad una buona soluzione è oggetto di ricerca prima dei rami che conducono a fallimenti, a condizione naturalmente che sia richiesta solo una soluzione qualsiasi.

Per esempio, può esserci una strategia che inizializza la variabile al valore nel suo dominio che con maggiore probabilità può condurre ad una soluzione e quindi con minore possibilità conduce ad un fallimento. In questo modo, la probabilità di raggiungere una soluzione alla fine del ramo corrente aumenta. Comunque, se sono richieste tutte le soluzioni oppure non ci sono soluzioni realizzabili, la strategia di ordinamento di valori non ha effetto sul tempo di soluzione, dato che vengono tentati tutti gli assegnamenti di valori.

3.3 Ottimizzazione di Vincoli

In molte applicazioni reali, non si vuole cercare una qualsiasi soluzione ma una soluzione ottima. La qualità della soluzione è usualmente misurata da una funzione dipendente dall'applicazione, cioè da una *funzione obiettivo*, definita sulle variabili del problema. Lo scopo è quello di trovare una soluzione che soddisfi tutti i vincoli e minimizzi o massimizzi la funzione obiettivo.

Tali problemi sono detti *Constraint Satisfaction Optimization Problems* (CSOP). Un CSOP consiste in uno standard CSP ed in una funzione di ottimizzazione che mappa ogni soluzione (labelling completo di variabili) in un valore numerico (come definito in [92]).

Come visto nel capitolo precedente, l'algoritmo più usato nell'area della OR per la ricerca di soluzioni ottime di un problema è quello di branch-and-bound [73], il quale può essere applicato anche ad un CSOP. In effetti, i sistemi di CP che devono affrontare un problema di ottimizzazione, solitamente implementano una sorta di algoritmo branch-and-bound.

Senza perdere di generalità, possiamo considerare un problema di minimizzazione. L'idea è quella di risolvere un insieme di problemi di realizzabilità (cioè, viene trovata una soluzione realizzabile se ne esiste almeno una) che conducono consecutivamente a soluzioni migliori.

In questa applicazione, il branch-and-bound necessita di una funzione euristica che mappi il labelling parziale ad un valore numerico. Essa rappresenta una sottostima (nel caso in esame di minimizzazione) della funzione obiettivo per il miglior labelling completo ottenuto dal labelling parziale. L'algoritmo ricerca soluzioni in modo depth-first e si comporta come il BT, eccetto per il fatto che appena un valore viene assegnato alla variabile, viene calcolato il valore della funzione euristica per il labelling. Se il valore calcolato supera un limite dato, allora il sottoalbero sotto il corrente labelling parziale viene immediatamente potato. Inizialmente, il limite viene fissato al valore $+\infty$ e durante i calcoli esso registra il valore della migliore soluzione trovata sino a quel momento.

In particolare, ogni volta che viene trovata una soluzione z^* realizzabile con costo associato $f(z^*)$, ad ogni sottoproblema nel rimanente albero di ricerca viene aggiunto il vincolo $f(x) < f(z^*)$, dove x è una qualsiasi soluzione realizzabile. Lo scopo del vincolo aggiuntivo, detto *upper bounding constraint*, è quello di rimuovere porzioni dello spazio di ricerca che non possono condurre

a soluzioni migliori della migliore trovata fino a quel momento.

L'efficienza dell'algoritmo branch-and-bound è determinato da due fattori: la qualità della funzione euristica e dal fatto che una buona soluzione sia trovata più o meno velocemente. Osservazioni su problemi reali mostrano che l'ultimo passo del processo di ottimizzazione, cioè il miglioramento ulteriore di una buona soluzione, è solitamente la parte più costosa dal punto di vista computazionale dell'intero processo di risoluzione.

Fortunatamente, in molte applicazioni, l'utente si accontenta di una soluzione vicina all'ottimo a patto che essa sia trovata velocemente. L'algoritmo di branch-and-bound può essere utilizzato anche per trovare soluzioni sub-ottimali con l'utilizzo di un secondo limite di "accettabilità". Se l'algoritmo trova una soluzione che è migliore del limite di accettabilità, allora tale soluzione può essere restituita all'utente anche se non si ha la prova che sia ottima.

I problemi legati all'approccio branch-and-bound della CP per la risoluzione di problemi di ottimizzazione sono essenzialmente due:

1. La CP non può contare su sofisticati algoritmi per il calcolo di limiti superiori ed inferiori sulla funzione obiettivo e deriva tali valori partendo dai domini delle variabili.

2. In generale, il legame tra la funzione obiettivo e le variabili decisionali del problema è piuttosto povero e non produce un efficace filtraggio di domini.

3.4 Programmazione Logica con Vincoli

Il concetto di *Programmazione Logica* (brevemente, LP, dall'inglese *Logic Programming*) è emerso nel corso degli anni '80. In LP si può dichiarare un problema come un programma logico, usando la logica come un linguaggio di programmazione. Un tale programma logico può essere passato al risolutore, il quale usa l'enumerazione e la deduzione logica per cercare una soluzione al problema. Uno dei più popolari linguaggi di LP è il PROLOG.

La LP presenta diversi vantaggi nella dichiarazione di problemi combinatori discreti:

1. La sua forma relazionale è appropriata per dichiarare i vincoli.

2. Le semantiche dichiarative facilitano la modifica e l'estensione dei programmi.

3. Il tempo di sviluppo dei programmi è considerevolmente corto.

Comunque, la LP presenta anche qualche svantaggio:

1. Il linguaggio di LP non offre una vasta gamma di costrutti sintattici. L'utente deve quindi definire da solo ogni regola e questo produce un programma logico molto complesso.

2. Il risolutore di LP usa molto spesso uno schema di enumerazione per generare soluzioni e testa le soluzioni enumerate conformi al programma logico. Questo approccio fa diminuire la performance del risolutore per problemi grandi. Anche nel caso in cui si utilizzi una procedura standard di bracktracking, la performance peggiora drasticamente con l'aumentare della dimensione del problema.

Ambedue questi svantaggi possono essere cancellati introducendo speciali costrutti sintattici nel linguaggio di LP, che hanno valore sia per l'espressività del linguaggio che per l'uso all'interno del risolutore. Questi costrutti possono essere visti come speciali vincoli che offrono all'utente la desiderata espressività ed anche al risolutore la possibilità di utilizzare questa esplicita informazione globale all'interno del suo schema risolutivo. Da tale estensione nasce la *Programmazione Logica con Vincoli* (CLP, dall'inglese *Constraint Logic Programming*), che spesso viene identificata con la CP.

Il primo passo consiste nell'estendere il linguaggio di una piattaforma di LP con vincoli globali speciali. Naturalmente questi vincoli devono avere valore per l'utente, ma dovrebbero anche fornire al risolutore qualche utile informazione.

Una volta permesso l'uso di vincoli globali in linguaggi di LP, occorre adattare il risolutore. Prima di tutto, i comuni risolutori di LP non sono in grado di trattare efficientemente questi nuovi vincoli introdotti nel linguaggio. In aggiunta, una delle ragioni per introdurre i vincoli globali è stata quella di aver la possibilità di usarli durante il processo di risoluzione.

Mentre lo schema di risoluzione della LP inizia enumerando le possibili soluzioni e verificando la realizzabilità di una soluzione generata (schema GT), lo schema della CLP cerca di enumerare e testare in un modo più "intelligente". Lo scopo è quello di ridurre il tempo di risoluzione riducendo il numero di

nodi da enumerare e potando lo spazio di ricerca prima possibile. In CLP vengono utilizzate le tecniche di consistenza per verificare se i vincoli sono ancora soddisfatti durante il processo di risoluzione.

Dunque, la CLP può essere vista come l'estensione della LP con costrutti speciali e tecniche risolutive, vale a dire come l'unione di due paradigmi dichiarativi: la risoluzione di vincoli e la LP. I vantaggi rispetto alla LP sono rappresentati da una estesa espressività e dal tempo di risoluzione ridotto.

Più precisamente, lo schema di CLP definisce una classe di linguaggi basati sul paradigma della CP [64]. Questi linguaggi sono supportati da una semplice struttura di semantiche concepita per incapsulare sia il paradigma della risoluzione di vincoli che quello di LP.

L'idea generale che sta dietro a tale schema è l'implementazione di diversi strumenti matematici per soddisfare vincoli numerici e l'uso di verifiche di consistenza e di tecniche di propagazione di vincoli per risolvere vincoli simbolici.

La CLP, mantenendo i vantaggi della LP, ottiene notevoli guadagni in efficienza, dato che ogni volta che il dominio di una variabile viene cambiato, tutti i vincoli esistenti che si riferiscono a quella variabile vengono "risvegliati", processo che può provocare una nuova propagazione di vincoli dato che più domini sono cambiati [93]. Questo può essere davvero efficiente (dato che può in molti casi eliminare completamente il backtracking) e rappresenta la base dei miglioramenti di performance ottenuti attraverso l'uso della CLP.

La CLP è stata la base per molti sistemi commerciali di successo che modellano e risolvono problemi reali usando la tecnologia basata sui vincoli. In particolare, CHIP è stato il primo linguaggio di CLP per l'utilizzo di propagazione di vincoli.

3.5 Applicazioni, Limitazioni e Tendenze della CP

La CP è stata applicata con successo a problemi di molte aree diverse come l'analisi della struttura del DNA, il time-tabling per ospedali e lo scheduling industriale. La CP si dimostra molto adatta alla risoluzione di problemi reali dato che molti campi di applicazione evocano in modo naturale la descrizione mediante vincoli.

I problemi di assegnamento sono stati forse il primo tipo di applicazione industriale risolto con la CP. Un'altra area tipica di applicazione è quella dell'assegnamento del personale, il quale in genere impone vincoli difficili. Ma l'area di applicazione probabilmente di maggior successo per vincoli di domini finiti è certamente rappresentata dai problemi di scheduling, in cui i vincoli esprimono in modo naturale le limitazioni reali. Queste sono solo le principali aree di applicazione della CP, ma molte altre sono state affrontate con successo.

Tuttavia, questo considerevole utilizzo della CP nella risoluzione di problemi reali mette in luce un certo numero di limitazioni e difetti degli strumenti attualmente disponibili.

Dato che la maggior parte dei problemi risolti dalla CP appartiene alla classe dei problemi $\mathcal{NP}$–ardui, l'identificazione di restrizioni che rendono il problema intrattabile è davvero importante sia dal punto di vista teorico che da quello pratico. Comunque, come con la maggior parte degli approcci a problemi $\mathcal{NP}$–ardui, l'efficienza del programma di CP è ancora imprevedibile e l'intuizione è solitamente la parte più importante di decisione su come e quando usare vincoli. Il problema più comunemente dichiarato da un utente di CP riguarda la stabilità del modello. Anche piccole modifiche in un programma o nei dati possono infatti condurre ad una modifica drammatica nella performance. Sfortunatamente, il processo di esecuzione di debugging per una esecuzione stabile su un certo numero di dati in input, attualmente non è ben compreso.

Un altro problema riguarda la scelta dell'appropriata tecnica di soddisfazione di vincoli da utilizzare per un particolare problema. A volte una rapida ricerca "cieca", come il BT, si rivela più efficiente di una più costosa propagazione di vincoli, e viceversa.

Come già accennato, un problema particolare in molti modelli di CP è rappresentato dal costo di ottimizzazione. A volte, è davvero difficile migliorare una soluzione iniziale, ed un piccolo miglioramento richiede molto più tempo della ricerca della soluzione iniziale. Inoltre, i programmi di CP sono in un certo senso incrementali (essi possono aggiungere vincoli dinamicamente), ma non hanno un supporto per la risoluzione di vincoli on-line, cosa sempre più richiesta nei correnti ambienti in continuo cambiamento.

I difetti del corrente sistema di soddisfazione di vincoli indicano le direzioni per sviluppi futuri. Tra questi, il modelling sembra uno dei più importanti. Attualmente la discussione riguarda l'uso di vincoli globali che incapsulano vin-

coli primitivi all'interno di un pacchetto più efficiente. Una questione molto più generale riguarda i linguaggi di modelling per esprimere problemi di CP. La maggior parte dei pacchetti disponibili in CP consiste in estensioni di un linguaggio di programmazione (CLP) o librerie che sono state usate con linguaggi di programmazione convenzionali (ILOG Solver). Sono stati, ad esempio, introdotti linguaggi di modelling simili a descrizioni algebriche (come il linguaggio *Numerica*, [32]) per semplificare la descrizione dei vincoli e sono stati anche usati linguaggi di modelling visivo (come il *VisOpt VML*) per generare programmi per disegni visivi.

Lo studio di interazioni di diversi metodi di risoluzione di problemi vincolati rappresenta uno dei problemi maggiormente discussi in tutte le aree di ricerca. Gli *algoritmi ibridi* che combinano varie tecniche di risoluzione sono uno dei risultati di tale ricerca [63].

Un'altra interessante area di studio è rappresentata dalla *collaborazione di risolutori* [78] e la relativa combinazione di studi sulla teoria di prova. La combinazione di tecniche di soddisfazione di vincoli con metodi tradizionali di OR come la PLI rappresenta un'altra importante sfida della ricerca attuale.

3.6 Confronto e Integrazione con OR

La classe dei problemi di Ottimizzazione Combinatoria è stata largamente studiata negli anni da diversi campi di ricerca, come quelli della OR, della AI e della CS, ciascuno dei quali caratterizzato da diversi approcci di risoluzione.

Il campo di OR ha significative sovrapposizioni con quello di AI. Ne è un esempio classico la ricerca branch-and-bound e ne sono esempi più recenti gli algoritmi basati sulla tabu search e sul simulated annealing. La CP è invece una emergente e promettente disciplina situata alla confluenza di CS, AI ed OR.

Quello che distingue gli approcci di OR ai problemi $\mathcal{NP}$–ardui è l'uso consistente di metodi continui basati sulla PL e sulla PLI. Dall'altro lato, nella CP l'enfasi è stata minore sulla struttura matematica dell'applicazione particolare e maggiore sul modelling di alto livello, sui metodi e strumenti di soluzione e sull'integrazione di idee da diversi sistemi di vincoli.

Un'altra sostanziale differenza tra i due campi consiste nel fatto che mentre le tecniche di OR vengono applicate solitamente a problemi di ottimizzazione, essendo basate su un *ragionamento di ottimalità*, tradizionalmente i program-

mi di CP sono problemi di realizzabilità, basati cioè su un *ragionamento di realizzabiltà.*

Comunque, esiste una differenza solo superficiale tra la ricerca di una soluzione realizzabile e quella di una soluzione ottima. Infatti, un algoritmo che cerca una soluzione realizzabile può essere trasformato in un algoritmo di ottimizzazione considerando una funzione obiettivo ed imponendo un limite su tale funzione. In qualsiasi momento venga trovata una soluzione, il limite viene aggiornato e quando non viene trovata alcuna soluzione realizzabile, l'ultima soluzione realizzabile trovata è quella ottima. Tuttavia, la CP non ha la disponibilità di strumenti per gestire in modo efficiente una funzione obiettivo, pur ammettendone la definizione in un suo programma.

Negli ultimi anni è andato aumentando l'interesse sulla combinazione di modelli e metodi di ottimizzazione con quelli di soddisfazione di vincoli. L'integrazione è stata inizialmente impedita dalle differenti origini culturali, essendo i primi largamente sviluppati nella comunità di OR e gli altri nelle comunità di CS e di AI. Comunque, i vantaggi risultanti dall'unione hanno rapidamente fatto superare tale barriera e nel tempo sempre più tecniche della OR sono state applicate all'interno di strutture di CP e viceversa.

Ad esempio, i vincoli globali in CP servono sia dal punto di vista dichiarativo come blocchi che costruiscono la dichiarazione del problema, sia dal punto di vista operazionale come componenti software che incapsulano tecniche specializzate di potatura. In molti casi, queste tecniche (algoritmi di filtraggio) hanno la loro origine nella OR e nella matematica discreta.

Un tipico esempio è la tecnica di *edge-finding* per lo scheduling inserita in vincoli globali per modellare disponibilità limitate di risorse (vedere [19],[4] e [1]). Un altro interessante esempio è rappresentato dal vincolo globale `alldifferent` che vincola un insieme di variabili ad essere assegnate a valori diversi [87]. Il problema di potare valori inconsistenti dai domini delle variabili di tale vincolo può essere riscritto come il problema di trovare tutti i matchings bipartiti massimali nel corrispondente grafo di valori. La teoria dei grafi [6] permette di fare questa riscrittura in maniera meno naïve e più efficiente usando un algoritmo di flusso e calcolando componenti fortemente connesse.

Storicamente, i vincoli globali permettono una efficace potatura sulla base di un ragionamento di realizzabilità: i valori sono rimossi dai domini se si dimostrano irrealizzabili. In problemi di ottimizzazione, i vincoli globali possono anche essere usati per un ragionamento di ottimalità: i valori vengono rimossi

dai domini se si dimostrano sub-ottimali (vedere [41] e [84]). Questo secondo tipo di filtraggio può essere ottenuto inserendo rilassamenti o algoritmi con finalità speciali (ottimizzazione) in vincoli globali.

Nella ricerca di tecniche di integrazione tra i due campi è stata rivolta un'attenzione certamente particolare ai vincoli globali. In effetti, il modelling fatto per mezzo di tali vincoli presenta diversi vantaggi:

Modelli Vincoli più espressivi conducono a modelli più piccoli in cui le variabili ed i vincoli hanno un mapping più vicino al problema in esame. Questo certamente semplifica il compito di modelling e di mantenimento.

Inferenza I vincoli globali catturano una struttura e la consegnano intatta al risolutore. Questo permette al risolutore di applicare efficaci algoritmi di inferenza specializzati.

Rilassamenti I rilassamenti possono essere aggiornati dinamicamente, permettendo una formulazione più stretta da mantenere.

Ricerca L'informazione per guidare la ricerca può essere raccolta da questi vincoli, permettendo una guida più precisa.

Visualizzazione I vincoli globali possono avere una visualizzazione, cioè lo stato del vincolo e delle variabili coinvolte può essere mostrato (durante la ricerca) in un modo specifico per il vincolo.

Dalla prospettiva di risoluzione, la più importante caratteristica è il fatto che rilassamenti e inferenza possono essere combinati ed applicati alle stesse strutture. In particolare, questo permette di far incontrare la propagazione di vincoli e la PL, a dispetto della differenza nei modelli tradizionalmente applicati.

Per l'Ottimizzazione Combinatoria, gli algoritmi più completi sono basati su alberi di ricerca, si tratta cioè di algoritmi che definiscono implicitamente un albero in cui i nodi interni corrispondono a soluzioni parziali, i rami sono scelte che partizionano lo spazio di ricerca ed i nodi foglia rappresentano soluzioni complete.

Sia la CP che la PLI contano su ramificazioni per l'enumerazione dello spazio di ricerca. Tuttavia, all'interno di questa struttura di ramificazione esse usano approcci duali alla risoluzione del problema: inferenza e ricerca, rispettivamente.

La CP combina la ramificazione con l'inferenza nella forma di propagazione di vincoli, che rimuove valori irrealizzabili dai domini delle variabili, riducendo così lo spazio da esplorare. Questo viene fatto attraverso regole di propagazione di vincoli incapsulate all'interno di vincoli globali. In questo modo un modello di CP scompone il problema in sotto-strutture, ognuna delle quali è trattata separatamente con algoritmi specializzati. L'effetto della propagazione di vincoli individuali è comunicato attraverso i domini di variabili condivise.

Non si tratta di un metodo di ricerca, cioè di un algoritmo che esamina una serie di labelling completi fino a trovare una soluzione. La PLI, invece, fa esattamente questo, ottenendo labelling completi dalla risoluzione di rilassamenti lineari del problema nell'albero di ramificazione, cioè intrecciando la ramificazione con rilassamenti che permettono di eliminare l'esplorazione di nodi per i quali il rilassamento è irrealizzabile o peggiore della migliore soluzione disponibile.

Inoltre, dando un limite inferiore sulla soluzione, il rilassamento fornisce anche un punto dello spazio attorno al quale la ricerca può essere concentrata. Nel caso di un buon rilassamento questo punto dovrebbe essere vicino alla soluzione ottima del problema. Questo fatto è sfruttato nella tradizionale ricerca di branch-and-bound della PLI.

La PLI presenta, rispetto alla CP, il vantaggio di poter generare piani di taglio che rinforzano il rilassamento lineare. Questa può rappresentare una potente tecnica quando il problema è ben disposto ad una analisi poliedrale. Tuttavia, la PLI presenta anche lo svantaggio che i suoi vincoli devono essere espressi come disequazioni (o equazioni) che coinvolgono variabili di valore intero (altrimenti il rilassamento lineare non è disponibile) e questo pone una severa restrizione sul linguaggio di modelling.

In generale, la PLI risulta davvero efficiente per problemi con buone formulazioni, ma soffre quando il rilassamento è debole o quando la sua struttura ristretta di modelling ha per risultato modelli grandi. La CP, invece, con i suoi vincoli più espressivi, ha modelli più piccoli che sono più vicini alla descrizione del problema reale e si comportano bene con problemi altamente vincolati, anche se manca la "prospettiva globale" dei rilassamenti.

La struttura dei risolutori di PLI comuni in OR non permette di definire strategie di ricerca flessibili per problemi specifici. Questo è dovuto molto probabilmente al fatto che la OR si è sviluppata per essere applicata dalla comunità matematica piuttosto che da quella della CS.

A confronto, la CP è stata maggiormente influenzata dalla CS e dalla progettazione di linguaggi di programmazione piuttosto che dalla matematica. Questo ha influenzato non solo le strategie di risoluzione, ma anche l'utilizzo di inferenza e modelling. In CP risulta disponibile una più vasta gamma di vincoli globali e l'utente può definirne di nuovi in diversi modi. Inoltre, le strategie di ricerca sono maggiormente supportate e sfruttate.

Gli approcci presenti in letteratura sull'integrazione tra CP e PLI sono principalmente di due tipi: l'approccio basato sull'idea di usare ambedue i modelli in parallelo e quello in cui si tenta invece di incorporare un modello nell'altro. Lo scopo principale di tale integrazione consiste nel prendere i vantaggi sia dell'inferenza attraverso la CP che dei rilassamenti attraverso la PLI, al fine di ridurre la dimensione dell'albero di ricerca per un dato problema.

Le chiavi di decisione che devono essere prodotte per l'integrazione sono rappresentate da quattro elementi fondamentali: i modelli, l'inferenza, i rilassamenti e le strategie di ramificazione e ricerca da usare.
Numerosi e diversi tra loro sono stati i tentativi di integrazione tra PLI e CP. Beringer e De Backer [7], ad esempio, propongono l'esplorazione di accoppiamenti di risolutori di CP e di PLI con propagazione di limiti e variabili fissate. Wallace et al. [56] usano la CP insieme con rilassamenti lineari in un singolo albero di ricerca per potare domini e stabilire limiti. Un nodo può fallire o perché la propagazione produce un dominio vuoto o perché il rilassamento lineare è irrealizzabile o ha un valore ottimo peggiore del valore della soluzione ottima. Una procedura sistematica viene usata per creare un modello "ombra" di programmazione lineare mista del modello originario di CP. Esso include vincoli aritmetici e vincoli `alldifferent`.

Il rilassamento lineare in un modello ibrido può anche essere rinforzato aggiungendo piani di taglio come nella PLI pura, ed i piani di taglio possono anche essere azionati da tecniche di consistenza (vedere [18] e [61]). Inoltre, a volte possono essere realizzati rilassamenti lineari più stretti se il rilassamento lineare di un vincolo simbolico è dinamicamente riscritto su cambiamenti di domini. Refalo [86] propone uno schema detto *tight cooperation* che fa proprio questo.

Bockmayr e Kasper [12] propongono una struttura per combinare CP e PLI in cui sono possibili diversi approcci di integrazione o sinergie. Essi indagano su come i vincoli simbolici possono essere incorporati all'interno della PLI, mostrando in particolare come un sistema lineare di disequazioni possa essere

usato in CP incorporandolo come vincolo simbolico.

Carlsson e Ottosson [18] confrontano CP, PLI ed un algoritmo ibrido per un problema di configurazione la cui forma base è un problema intero, le cui estensioni però risultano difficilmente risolvibili da un approccio puro di PLI. Rilassamenti lineari, ricerca branch-and-bound, piani di taglio, preprocessing [89] e consistenza sui limiti [77] vengono sperimentati all'interno di versioni sia pure che miste. Esperimenti computazionali mostrano che per questo problema i rilassamenti lineari ed i piani di taglio sono i fattori più importanti per l'efficienza e che il preprocessing e la consistenza sui limiti aiutano certamente, ma sono significativamente meno importanti di un rilassamento stretto.

Ci sono anche approcci in cui i modelli di CP e di PLI non sono uniti. Ad esempio, Heipcke [58] propone uno schema in cui due diversi modelli (uno di CP ed uno di PLI) vengono risolti separatamente in due alberi di ricerca sincronizzati. I due modelli sono collegati da variabili, le quali forniscono comunicazione di informazioni tra i risolutori.

Hooker, Kim, Otosson e Thorsteinsson [60], invece, sostengono l'utilizzo non di un modello di CP né di PLI, ma piuttosto di un modello specifico per il risolutore ibrido. Tale modello separa grossolanamente il problema in due parti, una parte discreta (FD store) ed una parte continua (LP store). Questo porta alla riduzione di domini sulla parte discreta (propagazione di vincoli), all'inferenza da questa a quella continua (limiti e piani di taglio) e ad un rilassamento naturale (rilassamento lineare). In questo lavoro manca l'inferenza da LP a FD store e tale mancanza comporta il fatto che la comunicazione risulta principalmente unidirezionale. Tuttavia, in un secondo lavoro degli stessi ricercatori [62] tale inferenza è stata studiata e proposta.

Jain e Grossmann [51] presentano uno schema in cui il problema è scomposto in due sotto-parti, una trattata dalla PLI ed una dalla CP. Questo è esemplificato con un problema di scheduling multi-machine in cui l'assegnamento di compiti alle macchine è formalizzato come un programma intero e la sequenza dei lavori sulle macchine assegnate è trattata con la CP. Lo schema di ricerca implementato è una procedura iterativa, in cui prima è risolto all'ottimalità il problema di assegnamento (identificando quale macchina usare per quale lavoro), e poi è risolto un problema di realizzabilità di CP cercando la sequenza conforme a questo assegnamento. Se la sequenza fallisce, vengono aggiunti piani di taglio al programma intero per vietare gli assegnamenti (mantenendo però la sequenza) ed il processo viene iterato.

Per quanto riguarda le estensioni di CP all'interno della PLI, qualche ricerca ha puntato ad incorporare i migliori supporti per vincoli simbolici e logici in PLI. Ad esempio, Hajian et al. (vedere [55] e [54]) mostrano come le disuguaglianze ($X_i \neq X_j$) possano essere trattare più efficientemente nei risolutori di PLI. Inoltre, essi danno un modelling lineare del vincolo `alldifferent`, introducendo variabili *non-zero*, cioè variabili che non possono prendere zero come valore. Questa restrizione non è espressa come vincoli nel modello, ma piuttosto trattata implicitamente attraverso un'estensione dell'algoritmo branch-and-bound.

A parte i vari schemi di ottimizzazione centrati attorno alla PLI ed alla programmazione mista, la comunità di OR ha sviluppato una vasta gamma di algoritmi per problemi specifici, come i problemi di scheduling o di reti di flusso. L'integrazione di questi algoritmi in CP, insieme con gli algoritmi su grafi per la matematica discreta applicata, rappresenta di gran lunga il più grande sforzo di integrazione in CP. Questi algoritmi sono incapsulati in vincoli globali (vedere per esempio, [4]) e le aree di applicazione vanno dallo scheduling (vedere [19],[20] e [1]) e dal routing [21] alla allocazione ed all'imballaggio (vedere [87],[3],[16] e [17]).

I *costi ridotti* di un rilassamento sono stati a lungo usati in OR per inferenza (rafforzamento dei limiti). La linea di ricerca seguita da Focacci, Lodi, Vigo e Milano (vedere [43],[40],[41] e [42]) usa i costi ridotti di un sottoproblema di assegnamento per propagazione in una struttura di CP (vincoli `alldifferent` e `path`). L'importanza di questo approccio risiede nel fatto che la propagazione in questi vincoli ora diventa *orientata all'ottimizzazione*, cioè non vengono rimossi dai domini solo gli elementi irrealizzabili, ma anche quelli sub-ottimali. Questo rimedia parzialmente la mancanza di rilassamenti in CP.

Un'altra tecnica efficiente di OR che è stata oggetto di studio per l'integrazione dei due campi è quella della generazione di colonne. Come visto nel capitolo precedente, si tratta di un approccio di PLI per una larga scala di problemi, in cui le variabili (colonne), che rappresentano soluzioni parziali, vengono generate dinamicamente all'interno di un problema di set partitioning. Recentemente, la CP è stata anche introdotta come metodo per generare colonne [35], mostrando di essere efficace nell'espressione di numerosi vincoli complicati.

In conclusione, per molti problemi (ad esempio, il progressive party problem [14], il warehouse location problem [15], il template design problem [85]

e il change problem [59]) la CP sembra più efficace, mentre per altri problemi (ad esempio, crew scheduling problem e flow aggregation problem [24]) sembra essere migliore la PLI. In molti casi, infine, gli approcci ibridi CP-PLI e le estensioni di PLI basate sulla logica hanno portato a considerevoli vantaggi sia nel modelling che negli algoritmi. Uno tra tutti è il caso del problema del commesso viaggiatore con finestre temporali (TSPTW) risolto efficacemente da diversi approcci ibridi (vedere [47] e [42]).

Capitolo 4

Problema di Vehicle Routing con Finestre Temporali (VRPTW)

4.1 Problemi di Routing

Come definito dal *Logistics Management Council* (una associazione di professionisti nella gestione della catena di approvvigionamento), la logistica è "il processo di pianificazione, implementazione e controllo del flusso efficiente, e dello stoccaggio di beni, servizi e relative informazioni dal luogo di origine a quello di consumo allo scopo di conformarsi alla richiesta del cliente".

Apportando migliorie agli assegnamenti delle rotte ai mezzi utilizzati (assegnamenti di routing), è possibile conseguire significativi risparmi attraverso la diminuzione dei costi di trasporto, la riduzione dei costi di di stoccaggio e di inventario, l'eliminazione di penalità dovute a carichi e consegne intempestivi. Il trasporto è infatti il principale fattore competitivo nella catena di approvvigionamento, dato che esso gioca un ruolo molto importante nella logistica, i cui costi totali dipendono direttamente dalle decisioni di trasporto.

In questo contesto, il problema dell'assegnazione delle rotte e la conseguente pianificazione dei percorsi di distribuzione e consegna divengono cruciali nella gestione della flotta di veicoli di una compagnia di trasporti. Nella realtà molte compagnie ed organizzazioni, certamente non solo quelle nel settore del trasporto, devono affrontare problemi che riguardano il trasporto fisico di materiali, persone o informazioni. Tali problemi sono comunemente detti *problemi*

di routing.

L'aumentata specializzazione nella lavorazione o fabbricazione di un prodotto e la crescente globalizzazione dell'economia mondiale hanno condotto negli ultimi decenni ad un aumento significativo del bisogno, e di conseguenza dell'importanza, del trasporto e ad una particolare attenzione rivolta al costo del trasporto, che rappresenta certamente una porzione significativa dei costi totali di ogni prodotto e servizio.

In un problema di routing "puro" c'è solamente una *componente geografica*, ma problemi di routing più realistici includono anche un aspetto di scheduling, come ad esempio una *componente temporale*.

I problemi di routing e di scheduling sono dunque elementi importanti nella maggior parte dei sistemi logistici e per questo motivo una larga parte della ricerca si è concentrata sullo sviluppo di modelli e metodi atti a produrre buone soluzioni a questi, che sono in realtà complessi problemi di Ottimizzazione Combinatoria.

Come visto nei capitoli precedenti, il campo della OR ha approntato negli ultimi decenni numerosi algoritmi e metodi di ottimizzazione, tra cui metodi esatti, euristiche e metaeuristiche, applicati alla rappresentazione del problema reale mediante un modello lineare di Programmazione Matematica. Molti di questi approcci si sono rivelati estremamente efficienti e per questo motivo trovano largo impiego per la risoluzione di larga parte dei problemi logistici odierni.

Dall'ottimizzazione dei percorsi di una flotta di veicoli, i quali debbono sottostare ad una serie di vincoli dati, prendono le mosse i cosiddetti *Vehicle Routing Problems* (VRPs). Nucleo centrale di tali problemi è la pianificazione di percorsi (le rotte) su cui sono disposti i clienti da raggiungere e servire, in modo che risultino minimizzati i costi di routing e di assegnamento di veicoli ai relativi percorsi.

Più precisamente, il VRP comporta la ricerca di un insieme di rotte per i veicoli di una flotta in modo che ciascun veicolo parta dal deposito e vi ritorni al termine della rotta associata e che allo stesso tempo un dato insieme di clienti sia interamente servito. Ad ogni veicolo deve essere assegnata una rotta che copra un certo numero di clienti in modo che ogni cliente sia visitato esattamente da un veicolo (e quindi esattamente una volta nell'intero processo di distribuzione). Ciascun cliente ha una domanda specifica che deve essere soddisfatta dalla consegna e nessun veicolo può servire più clienti di quanto gli

permetta di fare la propria capacità. L'obiettivo può essere quello di minimizzare la distanza totale percorsa dalla flotta o il numero di veicoli usati o una combinazione di questi obiettivi.

Il VRP è il più importante tra i problemi di routing, ma non è certamente l'unico che sia in grado di rappresentare efficacemente problemi reali di trasporto.

Il più semplice problema di routing, e per questo motivo anche il più studiato fra tutti, è il problema del commesso viaggiatore, il cosiddetto *Travelling Salesman Problem* (TSP), nel quale un commesso (che può essere visto come un veicolo) deve visitare un certo numero di clienti, ognuno una sola volta, partendo dal deposito e tornandovi alla fine del viaggio.

Dato un grafo (che rappresenta una rete di strade) i cui nodi rappresentano i clienti da visitare ed agli archi del quale sono associati dei costi (le distanze tra i nodi congiunti dagli archi), il problema consiste nel cercare la rotta più corta (o quella più economica dal punto di vista dei costi di routing) che parta dal deposito e vi ritorni dopo aver visitato ogni nodo del grafo esattamente una volta. Il grafo contiene anche un nodo che rappresenta il deposito da cui si immagina di far partire il commesso e che si assume come nodo termine del percorso di distribuzione. Il TSP è un problema puramente geografico, nel senso che i vincoli del problema e la funzione obiettivo (che rappresenta la grandezza da ottimizzare) dipendono solamente da una componente geografica.

Nonostante la semplicità descrittiva del problema TSP, trovare il percorso migliore che minimizzi la distanza percorsa o i costi totali e che visiti tutti i nodi del grafo una sola volta, è un problema di difficile risoluzione.

Assumendo di avere a disposizione un cero numero m di commessi, che possono servire un certo insieme di clienti, si ottiene il problema m-TSP , in cui ogni cliente deve essere visitato da un solo (arbitrario) commesso, il quale deve partire dal deposito e farvi ritorno alla fine del proprio percorso. In questo problema si ammette quindi la possibilità di visitare tutti i clienti non con un percorso unico, ma con un insieme di percorsi ciclici, disgiunti tra loro, salvo che fanno capo tutti allo stesso deposito. L'obiettivo è quello di minimizzare la spesa complessiva e l'unica differenza rispetto al TSP è la molteplicità di commessi, cioè di veicoli. Questo tuttavia già basta ad introdurre un'altra considerazione fondamentale, il fatto cioè che il numero di commessi può essere fissato o costituire un ulteriore obiettivo (solitamente da minimizzare).

Anche questo problema, come il TSP, è puramente geografico. Si tratta

però di un problema più complicato, dato che alla scelta della sequenza di clienti da visitare si aggiunge ora la scelta dell'accoppiamento tra cliente e veicolo. Questa maggiore complessità si traduce in pratica nel fatto che la dimensione dei problemi m-TSP che si riescono attualmente a risolvere in modo esatto risulta inferiore rispetto a quella dei TSP risolvibili.

Appare chiaro che il VRP è, in realtà, un m-TSP in cui ad ogni cliente è associata una domanda e nel quale ogni veicolo ha una limitata capacità di carico; per questo motivo viene anche detto *Capacitated Vehicle Routing Problem* (CVRP). Il vincolo di capacità limita l'insieme dei nodi del grafo visitabili da ciascun veicolo e riduce il numero di soluzioni rispetto al TSP.

Il VRP non è puramente geografico dato che la domanda può essere vincolante ed è anche più difficile del TSP, tanto che generalmente è possibile trovare una soluzione ottima solo se il numero di clienti da servire è relativamente piccolo. I metodi raffinati che consentono di risolvere il TSP possono essere estesi a problemi più complessi, divenendo però meno efficaci. Infatti, i più grandi VRP risolti in modo esatto non superano qualche centinaio di nodi, una dimensione talvolta appropriata alle applicazioni, ma non sempre.

Il VRP deve la sua importanza anche al fatto che rappresenta il modello di base per una vasta gamma di problemi di routing, ovvero diverse estensioni del problema di base ciascuna delle quali cerca di rappresentare con la miglior approssimazione possibile un determinato problema reale di trasporto.

In questo lavoro ci concentreremo sulla variante più importante del VRP, quella che comprende le *finestre temporali*, il cosiddetto *Vehicle Routing Problem with Time Windows*, spesso indicato semplicemente con la sigla VRPTW. Tale problema estende quello di base attraverso un vincolo aggiuntivo: i clienti devono essere serviti all'interno di un certo intervallo cronologico rappresentato dalla finestra temporale.

Il VRPTW risponde all'esigenza sempre più impellente e realistica di considerare l'aspetto temporale dei problemi di routing. Le finestre temporali, infatti, modellano una situazione, comunissima nella pratica, nella quale gli utenti di un sistema distributivo non solo avanzano una richiesta di servizio, ma indicano anche gli orari entro i quali tale richiesta deve essere soddisfatta.

Altri problemi di particolare importanza e caratterizzati da vincoli temporali sono il TSPTW, estensione del TSP con l'aggiunta delle finestre temporali sui clienti, ed il *Pickup and Delivery Problem with Time Windows* (PDPTW), variante del VRP in cui i veicoli durante il proprio percorso non solo consegnano

ma ritirano anche le merci all'interno di determinate finestre temporali.

Un'altra variante del VRP è il *Vehicle Routing Problem with Length Constraint* (VRPLC), cioè un VRP con vincoli sulle lunghezze delle distanze percorse in ogni rotta; nello *Split Delivery Problem* (SDP), invece, la domanda di un cliente non è necessariamente coperta da un solo veicolo, ma può essere divisa tra due o più veicoli. Altre varianti considerano più di un deposito, più tipi di mezzi di trasporto per il servizio o più tipi di merci da consegnare.

Quelle nominate sono solo alcune delle estensioni possibili del VRP, sono sufficienti però per far comprendere l'importanza di tale problema di base. Nel seguito di questo lavoro focalizzeremo la nostra attenzione sul *Problema di Routing di Veicoli con Finestre Temporali*, cioè sull'estensione più importante del VRP, il VRPTW, proponendone un modello matematico ed analizzando i principali approcci esatti a tale difficile problema di Ottimizzazione Combinatoria.

4.2 VRPTW

Il VRPTW coinvolge una flotta di veicoli che, partendo da un deposito, deve raggiungere e servire un certo numero di clienti, situati in diverse posizioni geografiche, ciascuno dei quali con specifiche domande e finestre temporali all'interno delle quali deve avvenire il servizio. Come nel VRP, ogni cliente deve essere servito esattamente da un veicolo e quindi una sola volta nell'intero processo di distribuzione. Inoltre, ad ogni veicolo deve essere assegnata una rotta che contenga consegne tali da non superare la capacità del veicolo stesso, il quale al termine del proprio servizio deve tornare al deposito da cui è partito.

Per quanto riguarda il vincolo supplementare, cioè quello temporale, esistono due principali varianti del problema:

- la variante in cui le finestre temporali sono "*hard*", cioè da rispettare rigidamente, nel senso che un veicolo può arrivare in anticipo da un cliente ma in tal caso per poterlo servire deve aspettare l'inizio della relativa finestra temporale e comunque non può mai arrivare dal cliente dopo la fine della finestra;

- la variante in cui le finestre sono "*soft*", cioè possono essere violate al costo però dell'introduzione di un contributo di penalizzazione sulla funzione obiettivo.

In generale, i ricercatori hanno considerato problemi VRPTW con finestre temporali rigide ed una flotta omogenea di veicoli. Tuttavia, con il maturare del campo di ricerca, l'aumentato livello di sofisticazione dei metodi sviluppati ha permesso di trattare anche flotte eterogenee. Per quanto riguarda la dimensione della flotta, questa è stata fissata a priori oppure assunta libera, cioè da determinare simultaneamente con il migliore insieme di rotte per i veicoli; comunque molti metodi possono attualmente ottimizzare la dimensione della flotta durante il processo.

Sebbene l'introduzione delle finestre temporali nel problema VRP renda ovviamente più difficile la costruzione ed il mantenimento di un insieme realizzabile di rotte, tale estensione permette la specificazione di funzioni obiettivo più realistiche rispetto alla "semplice" minimizzazione della distanza totale percorsa. Infatti, oltre a quest'ultima o al numero di veicoli utilizzati, si possono minimizzare la durata totale del viaggio oppure il costo complessivo coinvolto nel routing e nello scheduling, che consiste in costi fissi di utilizzo dei veicoli ed in costi variabili, come quelli di tempo necessario per gli spostamenti, di tempi d'attesa e di carico e scarico.

Nonostante la sua intrinseca complessità, il VRPTW ha ricevuto una considerevole attenzione negli ultimi anni nella comunità della OR. Prima di tutto perché è ancora uno dei problemi di Ottimizzazione Combinatoria più difficili e quindi rappresenta una grande sfida per ogni ricercatore; in secondo luogo per un aspetto più pratico già introdotto, vale a dire per il fatto che la soluzione di questo problema contribuisce direttamente alla riduzione dei costi nell'importante area della logistica.

Il VRPTW è stato studiato ed affrontato usando molte diverse tecniche che includono sia metodi esatti che euristiche e metaeuristiche. Data la difficoltà di realizzazione di metodi esatti veramente efficienti (in molti casi è necessario un tempo proibitivo per trovare una soluzione ottima ad una istanza del problema), la maggior parte della ricerca si è focalizzata sullo sviluppo di diverse ed efficienti tecniche euristiche in grado di fornire in un tempo ragionevole una buona soluzione al problema (ammissibile, ma non certamente ottima). Tra gli approcci euristici e metaeuristici al VRPTW, numerosi sono quelli basati su tabu search, simulated annealing e algoritmi genetici, con risultati davvero considerevoli.

Certamente meno numerosi, sono stati sviluppati anche efficienti approcci esatti al VRPTW ed è proprio di tali approcci che ci occuperemo nel seguito,

facendone una panoramica completa. Inizieremo col presentare gli approcci esatti di OR sviluppati e concluderemo con un approccio ibrido di OR e CP, l'unico presente finora in letteratura per il problema in esame.

4.3 Un Modello Matematico del VRPTW

Descriviamo ora un modello matematico di PLI per il VRPTW che useremo per presentare buona parte degli approcci esatti di OR al problema.

Il VRPTW è dato da una flotta omogenea di veicoli rappresentata dall'insieme $\mathcal{V}$, da un insieme di clienti $\mathcal{C}$ e da un grafo orientato $\mathcal{G}$, cioè una rete orientata che connette i clienti ed il deposito. Posto $|\mathcal{C}| = n$ il grafo consiste in $n+2$ vertici (o nodi), dove i clienti sono rappresentati dai vertici $1, 2, \ldots, n$ ed il deposito è rappresentato da due vertici, dal vertice 0 che rappresenta il deposito da cui partono tutti i veicoli con il carico da consegnare e dal vertice $n+1$ che rappresenta il deposito a cui tornano tutti i veicoli una volta percorsa la relativa rotta.

L'insieme di tutti i vertici $0, 1, \ldots, n+1$ è rappresentato dall'insieme $\mathcal{N}$ e l'insieme $\mathcal{A}$ degli archi del grafo rappresenta le connessioni tra deposito e clienti e tra clienti. Naturalmente, nessun arco termina nel vertice 0 e nessun arco ha origine nel vertice $n+1$. Tutte le rotte iniziano in 0 e terminano in $n+1$. Ad ogni arco (i, j) della rete vengono associati un *costo* c_{ij} ed un *tempo di viaggio* t_{ij}, che può includere il tempo necessario per il servizio ad un cliente i. Un arco non esistente o non realizzabile può essere modellato usando un valore di costo grande; così, ad eccezione dei vertici 0 ed $n+1$, si può assumere che la rete sia completa. Ogni veicolo ha una data capacità q (uguale per ogni veicolo) ed ogni cliente $i \in \mathcal{C}$ ha una domanda d_i.

Ad ogni cliente $i \in \mathcal{C}$ è associata una *finestra temporale* $[a_i, b_i]$ che rappresenta l'intervallo temporale all'interno del quale deve iniziare il servizio al cliente i da parte di un veicolo della flotta. Tali finestre temporali sono da ritenersi rigide, nel senso che un veicolo deve assolutamente arrivare in i ed iniziare il servizio prima del tempo b_i e nel caso in cui arrivi prima del tempo a_i è costretto ad aspettare (senza però alcun costo o penalità) tale tempo per iniziare il servizio.

Anche il deposito ha una finestra temporale $[a_0, b_0]$, identica per il vertice 0 e per il vertice $n+1$, detta anche *scheduling horizon*. I veicoli devono lasciare il deposito di partenza (il vertice 0) all'interno della finestra $[a_0, b_0]$ e ritornare

al deposito (il vertice $n+1$) all'interno della finestra $[a_{n+1}, b_{n+1}]$, con $a_{n+1} = a_0$ e $b_{n+1} = b_0$. Senza perdere di generalità, si può assumere che sia $a_0 = b_0 = 0$, che c_{ij}, q, d_i, a_i, b_i siano interi non negativi e che t_{ij} siano strettamente positivi per ogni $i \in \mathcal{C}$ e per ogni $(i,j) \in \mathcal{A}$.

Assumiamo anche che valga la disuguaglianza triangolare sia per c_{ij} che per t_{ij}, cioè che valgano $c_{ij} \leq c_{ih} + c_{hj}$ e $t_{ij} \leq t_{ih} + t_{hj}$, $\forall h, i, j \in \mathcal{N}$. La disuguaglianza triangolare sui costi può essere assunta senza perdita di generalità, dato che è possibile aggiungere ogni scalare a tutti gli archi di costo c_{ij}, senza cambiare la soluzione ottima, in modo che ogni istanza di VRPTW possa essere trasformata in un'istanza che soddisfa la disuguaglianza triangolare sul costo. Questo non è vero per la disuguaglianza triangolare sul tempo, ma sembra che la maggior parte delle applicazioni soddisfi tale proprietà. In particolare, questo è vero nel caso in cui t_{ij} includa il tempo di servizio necessario per il cliente i. Un'importante conseguenza del fatto che la disuguaglianza triangolare sul tempo sia soddisfatta è che se un veicolo può servire un insieme di clienti $\mathcal{S}$, con $\mathcal{S} \subseteq \mathcal{C}$, allora esso può anche servire ogni sottoinsieme $\mathcal{S} - s$ di clienti, dove $s \in \mathcal{S}$.

Il modello contiene due tipi di variabili decisionali, x ed s. Per ogni arco $(i,j) \in \mathcal{A}$, dove $i, j \in \mathcal{N}, i \neq j, i \neq n+1, j \neq 0$, e per ogni veicolo $k \in \mathcal{V}$ si definisce la componente x_{ijk} di x come segue:

$$x_{ijk} = \begin{cases} 1 & \text{se il veicolo } k \text{ si sposta dal vertice } i \text{ al vertice } j \\ 0 & \text{altrimenti} \end{cases}$$

La componente s_{ik} della variabile decisionale s è definita per ogni vertice $i \in \mathcal{N}$ e per ogni veicolo $k \in \mathcal{V}$ e rappresenta il tempo in cui il veicolo k inizia a servire il cliente i. Nel caso in cui il veicolo k non serva il cliente i, s_{ik} non ha alcun significato. Avendo assunto $a_0 = 0$, risulta $s_{0k} = 0$ $\forall k \in \mathcal{V}$ e $s_{n+1,k}$ denota il tempo di arrivo del veicolo k al deposito.

L'obiettivo è quello di designare un insieme di rotte di costo minimo, una per ogni veicolo, in modo tale che:

- ogni cliente sia servito esattamente una volta,
- ogni rotta abbia origine nel vertice 0 e termine nel vertice $n+1$,
- vengano osservati i vincoli di finestre temporali e di capacità.

Il VRPTW può essere matematicamente dichiarato come:

$$\min \sum_{k\in\mathcal{V}} \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}} c_{ij}\, x_{ijk} \qquad s.a \tag{4.1}$$

$$\sum_{k\in\mathcal{V}} \sum_{j\in\mathcal{N}} x_{ijk} = 1 \qquad \forall\, i \in \mathcal{C} \tag{4.2}$$

$$\sum_{i\in\mathcal{C}} d_i \sum_{j\in\mathcal{N}} x_{ijk} \leq q \qquad \forall\, k \in \mathcal{V} \tag{4.3}$$

$$\sum_{j\in\mathcal{N}} x_{0jk} = 1 \qquad \forall\, k \in \mathcal{V} \tag{4.4}$$

$$\sum_{i\in\mathcal{N}} x_{ihk} - \sum_{j\in\mathcal{N}} x_{hjk} = 0 \qquad \forall\, h \in \mathcal{C}, \forall\, k \in \mathcal{V} \tag{4.5}$$

$$\sum_{i\in\mathcal{N}} x_{i,n+1,k} = 1 \qquad \forall\, k \in \mathcal{V} \tag{4.6}$$

$$s_{ik} + t_{ij} - K(1 - x_{ijk}) \leq s_{jk} \qquad \forall\, i, j \in \mathcal{N}, \forall\, k \in \mathcal{V} \tag{4.7}$$

$$a_i \leq s_{ik} \leq b_i \qquad \forall\, i \in \mathcal{N}, \forall\, k \in \mathcal{V} \tag{4.8}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall\, i, j \in \mathcal{N}, \forall\, k \in \mathcal{V} \tag{4.9}$$

La funzione obiettivo (4.1) dichiara che i costi totali di routing dovranno essere minimizzati. I vincoli (4.2) dichiarano che ogni cliente deve essere visitato esattamente una volta, cioè deve essere assegnato esattamente ad un veicolo. Nel seguito indicheremo tali vincoli come *vincoli di assegnamento*. I vincoli (4.3) dichiarano che nessun veicolo viene caricato più di quanto sia permesso dalla sua capacità. Le tre equazioni successive, (4.4), (4.5) e (4.6), sono vincoli di flusso ed assicurano rispettivamente che ogni veicolo $k \in \mathcal{V}$ parta dal deposito (cioè dal vertice 0), che dopo aver raggiunto un cliente il veicolo parta nuovamente e che alla fine esso torni al deposito (cioè al vertice $n+1$). Comunque, dato che l'arco $(0, n+1)$ è incluso nella rete, è permesso anche il viaggio "vuoto". Si noti che il vincolo (4.6) è ridondante; tuttavia, esso è mantenuto nel modello per mettere in evidenza la struttura di rete. Le disequazioni (4.7) dichiarano che un veicolo k che stia viaggiando dal cliente i al cliente j non può arrivare da quest'ultimo prima del tempo $s_{ik} + t_{ij}$, somma del tempo in cui k comincia il servizio al cliente i e di quello necessario per spostarsi da i al cliente j. Il valore K che compare in tale vincolo è uno scalare grande. Infine, i vincoli (4.8) assicurano che tutte le finestre temporali relative ai clienti siano soddisfatte, mentre i vincoli (4.9) sono i vincoli di interezza sulla variabile decisionale x.

Il modello permette di considerare un numero fissato di veicoli o un limite superiore dato da $|\mathcal{V}|$ sul numero di veicoli. Se questo limite superiore è sufficientemente alto, significa che il numero di veicoli disponibili è illimitato. Un limite inferiore sul numero permesso di veicoli è facilmente aggiunto. Un numero libero di veicoli è modellato ponendo $c_{0,n+1} = 0$. Se il numero di veicoli è fissato a $|\mathcal{V}|$ si può porre $c_{0,n+1}$ oppure $t_{0,n+1}$ ad un valore grande il quale assicura che l'arco $(0, n+1)$ non sarà usato in alcuna soluzione ottima. Un'altra opzione di modelling consiste nel permettere un numero libero di veicoli, ma mettendo un costo c_v su ogni veicolo usato. Questo può essere fatto ponendo $c_{0,n+1} = -c_v, \forall\, k \in \mathcal{V}$. Se c_v è sufficientemente grande, il modello avrà come obiettivo primario quello di minimizzare il numero di veicoli ed in un secondo momento quello di minimizzare i costi di viaggio.

Omettendo i vincoli (4.7) e (4.8) il modello proposto rappresenta un VRP. Questo può essere ottenuto settando $a_i = 0$ e $b_i = M$ (dove M è uno scalare grande) per ogni cliente i. In tal caso bisogna però aggiungere i vincoli di eliminazione di sottoprercorsi. Se si omettono i vincoli sopra e si ha anche $|\mathcal{V}| = 1$, il modello rappresenta un TSP. Mantenendo invece i vincoli temporali ed omettendo quelli di capacità, cioè i (4.3), si ottiene un m-TSPTW e nel caso in cui $|\mathcal{V}| = 1$ un TSPTW.

Infine, rimuovendo i vincoli di assegnamento (4.2) il problema diventa un *Elementary Shortest Path Problem with Time Windows and Capacity Constraints* (ESPPTWCC) per ogni veicolo, cioè il problema di trovare il percorso più corto che parta dal deposito e vi ritorni, che non violi i vincoli temporali e di capacità e che visiti i clienti sulla rotta al massimo una volta. Dato che la flotta di veicoli è assunta omogenea, i veicoli sono identici tra loro e quindi lo sono anche tutti i problemi ESPPTWCC che si ottengono rimuovendo i vincoli di assegnamento. Omettendo anche i vincoli temporali o quelli di capacità, oppure entrambi si ottengono rispettivamente i problemi ESPPCC, ESPPTW e ESPP. Si può definire anche un rilassamento di ESPPTWCC, il SPPTWCC, come il problema di trovare il percorso più corto che soddisfi i vincoli temporali e di capacità senza però la richiesta che ogni cliente non possa essere visitato più di una volta. In corrispondenza, omettendo i vincoli di tempo, di capacità o entrambi, si ottengono rispettivamente i problemi SPPCC, SPPTW e SPP.

4.4 Complessità Computazionale

Dato che il VRPTW contiene come casi speciali diversi problemi di ottimizzazione $\mathcal{NP}$–ardui (come il TSP ed il VRP), esso stesso è un problema $\mathcal{NP}$–arduo in senso forte.

La seguente proposizione è dovuta a Savelsbergh [88]:

Proposizione 1. *Provare la realizzabilità del TSPTW è un problema $\mathcal{NP}$–completo in senso forte. Anche se il tempo di viaggio è simmetrico, cioè se $t_{ij} = t_{ji}\ \forall i, j \in \mathcal{N}, i \neq j$, provare la realizzabilità del TSPTW rimane un problema $\mathcal{NP}$–completo in senso forte.*

Una conseguenza di tale proposizione è che anche trovare una soluzione realizzabile al VRPTW con un numero fissato $|\mathcal{V}|$ di veicoli è un problema $\mathcal{NP}$–arduo in senso forte. Se il numero di veicoli disponibili è illimitato, il problema di realizzabilità è facile, dato che esso equivale a determinare se una soluzione che consiste delle rotte deposito-cliente i-deposito, $\forall\, i \in \mathcal{C}$, è realizzabile. Questo può essere fatto in un tempo $O(n)$.

Per quanto riguarda i problemi di shortest path, è noto che il problema base SPP è polinomiale, può essere infatti risolto dall'agoritmo di Bellman-Ford-Moore [81] in un tempo $O(nm)$, dove n ed m rappresentano rispettivamente il numero dei vertici e quello degli archi. Per quanto riguarda le varianti di tale problema, valgono le proposizioni seguenti:

Proposizione 2. *SPPTWCC, SPPTW e SPPCC sono problemi $\mathcal{NP}$–ardui, ma risolvibili con algoritmi pseudo-polinomiali se:*

1. $t_{ij} \geq 0, \forall\, i, j \in \mathcal{N}$, *e*
2. $d_i > 0, \forall\, i \in \mathcal{C}$.

Proposizione 3. *ESPPTWCC, ESPPTW, ESPPCC e ESPP sono problemi $\mathcal{NP}$–ardui in senso forte.*

4.5 Approcci Esatti di OR

Il primo articolo apparso in letteratura che ha proposto un algoritmo esatto nel campo della OR per la risoluzione del VRPTW è stato pubblicato da Kolen, Kaan e Trienekens [71] nel 1987. Da allora ha preso il via un importante filone

di ricerca che ha portato a diversi metodi esatti per lo stesso problema, i quali possono essere classificati in tre categorie distinte a seconda del principio su cui è basato l'algoritmo risolutivo:

- metodi di programmazione dinamica;
- metodi basati sul rilassamento lagrangiano;
- metodi di generazione di colonne.

Il metodo presentato da Kolen et al. [71] calcola limiti inferiori usando la PD ed il rilassamento dello spazio degli stati. Le decisioni di ramificazione sono prese su assegnamenti rotta-cliente. Questo rappresenta l'unico approccio di PD al VRPTW presente in letteratura ed il problema più grande risolto con tale metodo contiene 15 clienti.

Fischer, Jörnsten e Madsen [38] nel 1997 descrivono un algoritmo basato sul rilassamento K-tree (un albero di supporto con l'aggiunta di K archi) del VRPTW. I vincoli di capacità sono trattati introducendo un vincolo il quale richiede che qualche insieme di clienti $\mathcal{S}$, con $\mathcal{S} \subset \mathcal{C}$, deve essere servito da almeno $k(\mathcal{S})$ veicoli. Questo vincolo viene rilassato in modo lagrangiano ed il problema risultante è ancora un problema K-tree con costi degli archi modificati. Le finestre temporali sono trattate in modo simile: viene infatti generato e rilassato in modo lagrangiano un vincolo, il quale richiede che non possono essere usati tutti gli archi in un cammino che viola il tempo. Tale metodo rientra nella seconda categoria ed ha risolto qualche problema coinvolgente fino a 100 clienti.

Kohl e Madsen [70] lo stesso anno presentano, invece, un approccio di cammino più corto (shortest path) con vincoli laterali seguito da rilassamento lagrangiano. Il metodo presentato rilassa i vincoli di assegnamento ed il modello viene così scomposto in un sottoproblema per ogni veicolo, cioè in un certo numero di ESPPTWCC tutti uguali tra loro (dato che i veicoli sono assunti identici tra loro). Il problema principale consiste nel trovare i moltiplicatori di Lagrange ottimi, cioè i moltiplicatori di Lagrange che producono il miglior limite inferiore. Tale problema è risolto con un metodo che utilizza sia l'ottimizzazione del sub-gradiente che un metodo bundle. Con tale approccio Kohl e Madsen sono riusciti a risolvere problemi coinvolgenti fino a 100 clienti.

In una panoramica di problemi di routing vincolati nel tempo, Dumas, Desrosiers, Solomon e Soumis ([29]) si concentrano su metodi basati sulla de-

composizione di Dantzig-Wolfe, concludendo che tale approccio produce risultati computazionali migliori rispetto ad altri approcci. Questo è dovuto al fatto che l'informazione ottenuta nel sottoproblema è sfruttata meglio nel problema principale della decomposizione di Dantzig-Wolfe (un rilassamento lineare di un problema di tipo set covering), dato che tutti i cammini generati possono essere sfruttati in tale problema. Gli autori notano anche che la decomposizione di Dantzig-Wolfe fornisce informazioni migliori per progettare il metodo di branch-and-bound. In [70], Kohl e Madsen mostrano come tale informazione possa essere usata in uno schema di rilassamento lagrangiano.

Al momento, gli algoritmi di maggior successo per il VRPTW sono basati sulle decomposizioni di shortest path del problema. L'osservazione fondamentale è che solo i vincoli (4.2) "legano" insieme i veicoli. Il problema che si ottiene omettendo tali vincoli, e quindi costituito dai vincoli (4.3),..., (4.9), è un ESPPTWCC per ogni veicolo. Questo problema è $\mathcal{NP}$–arduo in senso forte, ma esistono algoritmi di PD veramente efficienti per il problema lievemente rilassato SPPTWCC; esiste, in particolare, un algoritmo pseudo-polinomiale in grado di risolvere la maggior parte delle istanze del SPPTWCC di dimensione moderata.

Due tipi di decomposizione sono state computazionalmente investigate: la *decomposizione di Dantzig-Wolfe* e la *variable splitting*. In ambedue i casi, lo studio del VRPTW porta ad avere come sottoproblema un SPPTWCC. I metodi basati sul primo tipo di decomposizione rientrano nella categoria dei metodi basati sulla generazione di colonne, mentre quelli basati sulla variable splitting vengono inseriti nella categoria dei metodi basati sul rilassamento lagrangiano.

Desrosiers, Desrochers e Soumis [26] (vedere anche [28]) nel 1992 hanno applicato la generazione di colonne ad un certo numero di VRPTW con un numero illimitato di veicoli. Come notato in [28] questa generazione di colonne è nei fatti equivalente alla decomposizione di Dantzig-Wolfe e quindi denoteremo l'approccio presentato in [26] con quest'ultima dicitura, anche se essa non è direttamente usata dagli autori.

La decomposizione di Dantzig-Wolfe sfrutta il fatto che solo i vincoli (4.2) legano insieme i veicoli. Il metodo parte con una soluzione costituita da $|\mathcal{C}|$ cammini, ognuno dei quali visita un solo cliente. Il problema principale (*master problem*) consiste nel trovare un insieme di cammini di costo minimo, tra tutti i cammini generati, il quale allo stesso tempo assicuri che tutti i clienti vengano

serviti. Il numero di volte in cui un cammino è usato non è necessariamente un intero, ma può essere ogni numero nell'intervallo $[0, 1]$.

Dato che i cammini generati non sono cammini elementari, cioè lo stesso cliente può essere servito più di una sola volta su un cammino, il problema principale è il rilassamento lineare di un problema di tipo set partitioning. Per ragioni computazionali, Desrosiers et al. risolvono invece il rilassamento lineare di un problema di tipo set covering. Questo non cambia il risultato finale nel caso in cui valga la disuguaglianza triangolare sia sul tempo che sui costi.

I moltiplicatori del simplesso della soluzione ottima sono usati per modificare i costi nei sottoproblemi, i quali sono dei SPPTWCC e sono tutti identici (basta quindi risolverne solo uno). Se viene trovato un cammino con costo marginale negativo, esso viene incluso nell'insieme di cammini nel problema principale. Dato che si potrebbero trovare diversi cammini di costo marginale negativo, ogni iterazione può generare diversi cammini.

Il metodo termina quando nel sottoproblema non può essere generato alcun cammino di costo marginale negativo. A questo punto, tutti i cammini usati nella soluzione ottima al problema principale avranno costo marginale nullo, mentre tutti gli altri cammini avranno costo marginale non negativo. Se il problema principale risulta avere una soluzione intera, allora questa soluzione è ottima per il VRPTW. Altrimenti, si trova un limite inferiore sul valore ottimo e si applica qualche strategia di branch-and-bound per trovare la soluzione ottima. Tale metodo ha risolto problemi coinvolgenti fino a 100 clienti.

Soumis, Desrosiers, Kohl, Madsen e Solomon [30] introducono qualche anno più tardi (1999) delle disuguaglianze valide, cioè dei tagli (path cuts), per produrre dei limiti inferiori migliori per il VRPTW. Essi sviluppano anche un efficace algoritmo di separazione per trovare tali tagli. I tagli vengono incorporati quando necessario nel problema principale di un approccio di decomposizione di Dantzig-Wolfe. Il sottoproblema è un SPPTWCC. Si applica poi una procedura di branch-and-bound per ottenere soluzioni intere: si ramifica prima sul numero di veicoli se questo è frazionario, e poi sulle variabili di flusso. L'algoritmo è stato in grado di risolvere problemi coinvolgenti fino a 100 clienti ed uno con 150.

Madsen, Jörnsten e Sørensen [66] hanno proposto nel 1986 la decomposizione del VRPTW mediante il metodo di variable splitting. In tale metodo le variabili in alcuni vincoli vengono rinominate. Viene quindi introdotto e rilas-

sato in modo lagrangiano un nuovo tipo di vincolo, il quale associa le variabili originali e quelle nuove. Questo scompone il problema in due o più problemi indipendenti. Nel VRPTW questo significa che l'espressione $\sum_j x_{ijk}$ è sostituita dalla variabile binaria y_{ik} in qualche vincolo. Viene quindi introdotto e poi rilassato in modo lagrangiano il vincolo $\sum_j x_{ijk} = y_{ik}, \forall\, i \in \mathcal{N}, \forall\, k \in \mathcal{V}$. Ora il problema risulta separato in due problemi, uno nella variabili x_{ijk} e s_{ik} ed uno nella variabili y_{ik}. Il metodo di variable splitting è dunque un *metodo di decomposizione lagrangiana* (vedere [67] e [53]).

Per il VRPTW ci sono tre modi naturali di separare il problema, si può cioè considerare:

1. Un modello che consiste in un SPPTW, costituito dai vincoli (4.4), ...,(4.9), ed in un Generalized Assignment Problem (GAP), costituito dai vincoli (4.2) e (4.3).

2. Un modello che consiste in un SPPTWCC, costituito da tutti i vincoli tranne i (4.2), ed in un Semi Assignment Problem (SAP), costituito solo dai vincoli (4.2).

3. Un modello in cui i vincoli (4.3) vengono duplicati e modellati in ambedue i problemi, ottenendo così un SPPTWCC ed un GAP.

Il primo modello è stato implementato da Olsen [83] nel 1988 e si è dimostrato in grado di risolvere problemi coinvolgenti fino a 16 clienti, quindi con un risultato non particolarmente rilevante. Il secondo modello è stato descritto da Fisher et al. [38] e messo a punto anche da Halse [57], il quale è riuscito a risolvere problemi coinvolgenti fino a 100 clienti ed anche uno con 105. L'ultimo modello, invece, non è stato implementato.

Ricerche condotte alla fine degli anni '80 [31] hanno mostrato che risolvere il sottoproblema SPP come un programma intero spesso chiude lo scarto di interezza tra soluzione frazionaria del rilassamento lineare del sottoproblema e la soluzione intera del VRPTW. Dunque, in molti casi, non è necessario applicare un metodo branch-and-bound una volta trovati i moltiplicatori ottimi. Quando invece la ramificazione è necessaria, gli autori menzionati hanno usato differenti strategie di ramificazione. Per esempio, Kohl e Madsen [70] ramificano sulle finestre temporali, Halse [57] sulle variabili di assegnamento veicolo-cliente, mentre Desrosiers et al. [26] sul numero di veicoli.

Analizziamo ora tali approcci esatti di OR al VRPTW più nel dettaglio, secondo la suddivisione proposta nelle tre categorie sopra con riferimento al modello matematico precedentemente presentato.

4.5.1 Approccio di Programmazione Dinamica

Un approccio esatto di PD al VRPTW è presentato per la prima ed unica volta nel 1987 da Kolen et al. [71]. L'articolo è ispirato ad uno precedente (1981) in cui Christofides, Mingozzi e Toht [22] usano il paradigma della PD per risolvere il VRP.

L'algoritmo di Kolen et al. [71] usa il metodo di branch-and-bound per raggiungere l'ottimalità, cioè per minimizzare la distanza totale percorsa. La PD è inserita nel branch-and-bound per il calcolo di limiti inferiori.

Il Metodo di Branch-and-Bound

La descrizione del metodo branch-and-bound parte con la regola di ramificazione. Ogni nodo α nell'albero di ricerca corrisponde a tre insiemi: $F(\alpha)$ che è l'insieme di rotte realizzabili fissate che partono dal deposito e vi terminano, $P(\alpha)$ che è una rotta parzialmente costruita che parte dal deposito, e $C(\alpha)$ che rappresenta l'insieme di clienti che non possono essere i prossimi nella rotta $P(\alpha)$. Inizialmente, naturalmente, gli insiemi $F(\alpha)$ e $C(\alpha)$ sono vuoti e l'insieme $P(\alpha)$ è costituito solo dal deposito.

La ramificazione nell'albero di branch-and-bound è fatta selezionando un cliente che non sia vietato, cioè un cliente $i \notin C(\alpha)$, e che non appaia in alcuna rotta, fissata o parziale, cioè $i \notin F(\alpha) \cup P(\alpha)$. Le decisioni di ramificazione sono prese su assegnamenti rotta-cliente. Si ramifica creando due nodi $\alpha^{'}$ e $\alpha^{''}$, generando così due rami: un ramo in cui la rotta parzialmente costruita $P(\alpha)$ viene estesa da i (e risulta $C(\alpha) = C(\alpha^{'})$) ed uno in cui i viene vietato come prossimo cliente nella rotta, in cui cioè i viene aggiunto all'insieme $C(\alpha)$ $(C(\alpha^{''}) = C(\alpha) \cup \{i\})$. Se $i = 0$, la rotta parziale estesa sarà aggiunta all'insieme delle rotte fissate e viene cominciata una nuova rotta parziale $(P(\alpha) = (0), C(\alpha) = \emptyset)$.

In ogni nodo α dell'albero di ricerca, il metodo calcolerà un limite inferiore su tutte le estensioni possibili realizzabili della soluzione parziale caratterizzata da $F(\alpha)$, $P(\alpha)$ e $C(\alpha)$, rilassando la condizione che ogni cliente non ancora su una rotta debba essere servito esattamente una volta. Infatti, nel seguito

mostreremo come calcolare l'estensione più economica della soluzione parziale corrente ad un insieme di rotte in modo che il carico totale su queste rotte sia uguale a $D = \sum_{i \in \mathcal{N}} d_i$ e che ogni rotta abbia un diverso ultimo cliente.

Data la natura del limite inferiore, la selezione euristica del cliente i sarà quella di selezionare il prossimo cliente con il quale la rotta parziale $P(\alpha)$ è stata estesa nel calcolo del limite inferiore. Se non c'è alcuna rotta parziale ($P(\alpha) = \emptyset$), ne viene cominciata una nuova selezionando il cliente che appare con maggior frequenza come il primo cliente nelle rotte calcolate per il limite inferiore. All'inizio ed ogni volta che accade un arresto della procedura, la preferenza è data al cliente con la domanda più grande, dato che questa scelta riduce la dimensione dei grafi orientati che appaiono nei calcoli del limite inferiore.

Calcolo del Limite Inferiore

Ad ogni nodo dell'albero di branch-and-bound viene usata la PD per calcolare un limite inferiore su tutte le soluzioni realizzabili definite da $F(\alpha), P(\alpha)$ e $C(\alpha)$.

Per prima cosa si discute il caso del nodo radice, in cui si ha $F(\alpha) = \emptyset, C(\alpha) = \emptyset$ e $P(\alpha) =$ deposito. Si costruisce un grafo orientato con vertici $v(i, d, k)$ per $i = 0, 1, \ldots, n, d = 0, 1, \ldots, D$ e $k = 0, 1, \ldots, m$, dove n è il numero di clienti, m il numero di veicoli e D la somma di tutte le domande d_i dei clienti. Dunque, associato ad ogni nodo dell'albero di branch-and-bound c'è un insieme di rotte.

Un cammino orientato da $v(0, 0, 0)$ a $v(i, d, k)$ nel grafo corrisponde ad un insieme di k rotte con un carico totale pari a d e con differenti ultimi clienti visitati (ognuno dei quali in $\{1, 2, \ldots, i\}$). Le lunghezze degli archi nel grafo orientato saranno definiti in modo che la lunghezza di questo cammino orientato sia uguale alla lunghezza totale delle corrispondenti rotte. Il limite inferiore è dato allora dal minimo su $k = 1, 2, \ldots, m$ delle lunghezze dei cammini più corti da $v(0, 0, 0)$ a $v(n, D, k)$. Si noti che non ci sono vincoli che forzino i clienti (non appartenenti a $F(\alpha)$ o a $P(\alpha)$) ad essere visitati da alcuna delle rotte generate. Dunque il minimo risultante è un limite inferiore.

Dinamicamente si tenta di estendere un insieme di k rotte con carico d e l'insieme degli ultimi clienti $\{1, 2, \ldots, i\}$ all'insieme in cui gli ultimi clienti

provengono da $\{1, 2, \ldots, i+1\}$. Ci sono allora due possibilità, cioè due modi diversi di fare tale estensione:

- Il cliente $i+1$ non è incluso come punto finale di alcuna rotta. Questo approccio è rappresentato da un arco che va da $v(i,d,k)$ a $v(i+1,d,k)$ di lunghezza nulla.
  Si noti che un cliente $i+1$ che non sia punto finale di una rotta potrebbe ancora essere membro di una delle altre rotte generate dalla funzione $F(i,d)$ descritta sotto.

- Inserire il cliente $i+1$ come ultimo cliente in una rotta di carico totale d'. Questo approccio è rappresentato da un arco che va da $v(i,d,k)$ a $v(i+1,d+d',k+1)$ di lunghezza $F(i+1,d')$ per ogni possibile valore di d'. Generalmente, la funzione $F(i,d)$ è definita come la lunghezza minima di una rotta realizzabile con carico totale d ed ultimo cliente i. Questo problema è un SPP con vincoli laterali. Esso viene rilassato per permettere che un cliente sia servito più di una volta ed è risolto da una versione "estesa" dell'algoritmo di Dijkstra.
  Le rotte associate ad un dato vertice $v(i,d,k)$ sono le rotte date dal calcolo di $F(i,d)$ e dall'estensione prodotta usando gli archi nel grafo.

Se ci si trova non nel nodo radice ma in un nodo arbitrario dell'albero di branch-and-bound, occorre distinguere due casi:

1. $P(\alpha) = \emptyset$;

2. $P(\alpha) \neq \emptyset$.

Se $P(\alpha) = \emptyset$ si adatta il problema per il numero $\hat{k}$ di rotte già generate (cioè, $\hat{k} = |F(\alpha)|$), il loro carico $\hat{d}$ e l'insieme di clienti $\hat{I}$ già usati in queste rotte. L'algoritmo di PD sopra descritto può allora essere usato su questo problema ridotto, cioè su un grafo con vertici $v(i,d,k)$ per $i = 0, \ldots, n - |\hat{I}|$ (dopo la rinumerazione), $d = \hat{d}, \ldots, D$ e $k = \hat{k}, \ldots, m$. Il limite inferiore è dato dal minimo su $k = \hat{k}+1, \ldots, m$ di tutte le lunghezze dei cammini più corti da $v(0,\hat{d},\hat{k})$ a $v(n-|\hat{I}|, D, k)$.

Nel caso in cui $P(\alpha) \neq \emptyset$, esattamente una delle rotte che appaiono nel calcolo del limite inferiore è un'estensione di $P(\alpha)$. Ora, sia $\bar{F}(i,d)$ la lunghezza minima di una tale estensione con carico totale d ed ultimo cliente i ($\bar{F}(i,d)$ è calcolata nello stesso modo di $F(i,d)$). Come prima, il problema può essere

ridotto secondo $\hat{k}, \hat{d}$ e $\hat{I}$. Il grafo orientato è ora esteso per contenere i vertici $v(i,d,k)$ e $\bar{v}(i,d,k)$ ed i seguenti archi:

- Archi di lunghezza nulla da $v(i,d,k)$ a $v(i+1,d,k)$ e da $\bar{v}(i,d,k)$ a $\bar{v}(i+1,d,k)$. Questo equivale a non utilizzare il cliente $i+1$ nelle rotte.
- Archi di lunghezza $F(i+1,d')$ da $v(i,d,k)$ a $v(i+1,d+d',k+1)$ e da $\bar{v}(i,d,k)$ a $\bar{v}(i+1,d+d',k+1)$ per ogni possibile valore di d'.
- Archi di lunghezza $\bar{F}(i+1,d')$ da $v(i,d,k)$ a $\bar{v}(i+1,d+d',k+1)$.

L'ultimo tipo di arco corrisponde all'aggiunta di una singola rotta che estende $P(\alpha)$. Il limite inferiore è dato dal minimo su $k = \hat{k}+1,\ldots,m$ delle lunghezze dei cammini più corti da $v(0,\hat{d},\hat{k})$ a $\bar{v}(n-|\hat{I}|,D,k)$.

Con tale approccio si sono risolti problemi coinvolgenti fino a 15 clienti, un risultato quindi non particolarmente interessante.

4.5.2 Metodi Basati sul Rilassamento Lagrangiano

La seconda categoria menzionata contiene, come visto nella breve panoramica degli approcci presentata precedentemente, un certo numero di articoli che usano approcci leggermente diversi:

- l'approccio di cammino più corto (shortest path) seguito da rilassamento lagrangiano [70];
- l'approccio di K-tree seguito da rilassamento lagrangiano [38];
- il metodo di variable splitting seguito da rilassamento lagrangiano (vedere [66],[83],[38],[76] e [57]).

Durante le ultime due decadi il rilassamento lagrangiano si è dimostrato un importante strumento in PLI per l'Ottimizzazione Combinatoria. Esso è usato per ottenere limiti più stretti sull'ottimo mediante la risoluzione di un problema rilassato, cioè il sottoproblema. Il problema principale (master problem) consiste nell'ottimizzare i moltiplicatori di Lagrange, in modo da avere limiti più stretti possibile.

Il rilassamento lagrangiano può essere applicato al VRPTW in diversi modi. Si può considerare, ad esempio, di rilassare i vincoli temporali e di capacità, cioè i vincoli (4.7), (4.8) e (4.3). Questo produce un problema di rete di flusso

lineare, il quale possiede la cosiddetta *proprietà di interezza*, vale a dire che tutti i punti estremi del politopo costituito da tutte le soluzioni al rilassamento continuo del problema sono interi, purché lo siano i dati. Il limite può essere calcolato molto velocemente, ma è probabile che esso non sia veramente forte a meno che la capacità non sia vincolante e le finestre temporali siano veramente strette. Rilassare solo i vincoli di capacità o quelli temporali non sembra essere ragionevole, dato che il problema rilassato generalmente non risulta più facile di quello originale.

Approccio di Shortest Path

In alternativa, si può usare il rilassamento lagrangiano dei vincoli di assegnamento, cioè dei vincoli (4.2) che assicurano che ogni cliente sia servito. Questo scompone il VRPTW in un certo numero di SPPs. Questo tipo di rilassamento lagrangiano è stato proposto per la prima volta da Kohl e Madsen [70] nel 1997. Il problema principale consiste nel trovare i moltiplicatori di Lagrange ottimi ed il sottoproblema è un SPPTWCC. I moltiplicatori di Lagrange ottimi vengono trovati usando un metodo che sfrutta i benefici di metodi di sub-gradiente come pure un metodo bundle.

Tale approccio rappresenta un approccio innovativo nella categoria dei metodi basati sul rilassamento lagrangiano per il VRPTW. Il problema viene infatti scomposto come nella decomposizione di Dantzig-Wolfe, ma il modo in cui le variabili duali (in questo contesto i moltiplicatori di Lagrange) vengono ottimizzate è certamente nuovo nel campo di ricerca.

Rilassando i vincoli di assegnamento, la funzione obiettivo con i termini di penalità aggiunti diventa

$$\min \left\{ \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} (c_{ij} - \lambda_j)\, x_{ijk} + \sum_{j \in \mathcal{C}} \lambda_j \right\} \tag{4.10}$$

e quindi il problema rilassato è costituito da (4.10) e dai vincoli (4.3),..., (4.9). Qui λ_j è il moltiplicatore di Lagrange associato al vincolo che assicura che il cliente j sia servito. Indicheremo i costi modificati, detti anche *costi marginali*, con $\hat{c}_{ij} = c_{ij} - \lambda_j$, per ogni arco $(i, j) \in \mathcal{A}$. Il moltiplicatore λ_j riflette i costi di servire un cliente e generalmente sarà positivo; però, dato che i vincoli (4.2) sono vincoli di uguaglianza, i moltiplicatori possono prendere anche valori negativi.

Per ogni dato valore del vettore dei moltiplicatori $\underline{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_{|\mathcal{C}|})$, il modello risultante si scompone in un sottoproblema per ogni veicolo. Comunque, i $|\mathcal{V}|$ sottoproblemi sono tutti identici dato che lo sono i relativi insiemi di vincoli (i veicoli sono tutti identici) e che i costi degli archi sono indipendenti dal veicolo. Il sottoproblema è un ESPPTWCC e dato che i costi degli archi vengono modificati sottraendo il relativo moltiplicatore di Lagrange, il grafo può anche contenere cicli negativi.

Come già detto, il ESPPTWCC è un problema veramente difficile da risolvere, ma nella maggior parte dei casi si riesce a risolvere un suo rilassamento, il SPPTWCC, mediante un algoritmo di PD. Il SPPTWCC ammette che un cammino possa servire uno stesso cliente più di una sola volta. Risolvere il SPPTWCC invece del ESPPTWCC può diminuire il limite inferiore ottenuto, ma risultati sperimentali (vedere [26] e [70]) mostrano che il limite ottenuto è ancora di qualità eccellente.

I moltiplicatori di Lagrange associati ai vincoli di assegnamento possono essere interpretati come un prezzo dato per servire il cliente associato al vincolo. Ogni veicolo sceglie il proprio cammino, minimizzando il costo del cammino modificato dai moltiplicatori dei clienti serviti. Se il numero di veicoli disponibili è illimitato o limitato superiormente, ogni veicolo ha l'opzione di usare il cammino $(0) - -(n+1)$, cioè nessuna rotta a nessun costo. Se il numero di veicoli è limitato inferiormente, per esempio dal vincolo $\sum_k \sum_{j \in \mathcal{C}} x_{0jk} \geq v_{\min}$, questo vincolo viene rilassato in modo lagrangiano. Il corrispondente moltiplicatore di Lagrange modifica il costo degli archi che hanno origine in 0 e terminano in j, con $j \in \mathcal{C}$, mediante la formula $\hat{c}_{ij} = c_{ij} - \lambda_j - \gamma_{\min}$, dove $\gamma_{\min}$ indica il moltiplicatore associato al vincolo sul minimo numero di veicoli.

Se il SPPTWCC ha un unico ottimo, tutti i veicoli sceglieranno lo stesso cammino dato che essi sono tutti identici. A meno che il problema contenga un solo veicolo (cioè $|\mathcal{V}| = 1$), questa non può essere una soluzione realizzabile per il VRPTW, dato che i clienti che si trovano sullo stesso cammino più corto vengono serviti $|\mathcal{V}|$ volte, mentre tutti gli altri clienti non vengono serviti per niente. Comunque, la soluzione al SPPTWCC in generale non è unica, quando sono stati trovati i migliori moltiplicatori di Lagrange.

Per ogni vettore $\underline{\lambda}$ di moltiplicatori, il valore della funzione obiettivo della soluzione ottima al problema rilassato, che indichiamo con $f(\underline{\lambda})$, dà un limite inferiore sul valore ottimo per il VRPTW ([48]). Se $\hat{z}$ indica il costo marginale del cammino più corto, cioè se $\hat{z} = \sum_i \sum_j \hat{c}_{ij} x_{ijk} = \sum_i \sum_j (c_{ij} - \lambda_j) x_{ijk}$ per

una soluzione ottima al problema rilassato per ogni veicolo k, allora (4.10) produce la formula $f(\underline{\lambda}) = |\mathcal{V}|\hat{z} + \sum_i \lambda_i$. Indicheremo con $\underline{\lambda}^*$ un valore di $\underline{\lambda}$ che massimizza f.

Proposizione 4. *Assumiamo che f sia limitata superiormente. Se $\underline{\lambda} = \underline{\lambda}^*$ allora per ogni nodo $i \in \mathcal{C}$ esiste un cammino di costo marginale minimo che serve il cliente i.*

Dimostrazione Sia $\underline{\lambda} = \underline{\lambda}^*$. Sia z_i il costo marginale del cammino più corto che passa per il nodo $i \in \mathcal{C}$, cioè $z^* = \min_{i \in \mathcal{C}}(z_i)$. Assumiamo che il nodo i non sia in alcun cammino più corto, cioè $z_i - \epsilon = z^*, \epsilon > 0$. Ora aumentiamo λ_i con ϵ, cioè $\lambda_i := \lambda_i + \epsilon$. Questo diminuisce z_i a z^*, ma non cambia z^*. Dato che z^* è invariato, lo è anche il primo termine della funzione obiettivo rilassata, ma il secondo termine è aumentato di ϵ. Questo contraddice l'assunzione che $\underline{\lambda}$ sia ottima, e dunque mostra che i deve essere in un cammino più corto se $\underline{\lambda} = \underline{\lambda}^*$. ■

Se esiste un insieme di cammini di costo minimo, che allo stesso tempo copre ogni cliente esattamente una volta, allora è stata trovata una soluzione realizzabile e ottima. Se questo è il caso, può in principio essere determinata risolvendo un problema di set partitioning con colonne corrispondenti ai cammini di costo marginale minimo e righe corrispondenti ai clienti. Se il numero di veicoli è fissato o limitato questo deve essere incluso come vincolo. Se e solo se esiste una soluzione realizzabile a questo problema di set partitioning, il VRPTW è stato risolto all'ottimalità. Questa procedura non è direttamente implementabile, dato che generalmente non si conoscono *tutti* i cammini di costo marginale minimo, ma solo uno. Questo suggerisce il fatto che un'implementazione naïve di rilassamento lagrangiano fornisce solo un limite inferiore sull'ottimo, ma non è capace di produrre una soluzione intera ottima (o informazioni per una procedura branch-and-bound).

Il problema principale consiste nel cercare i moltiplicatori $\underline{\lambda}^*$ che producono il miglior limite inferiore, cioè nel risolvere il modello

$$\max_{\underline{\lambda} \in \mathbb{R}^n} f(\underline{\lambda}).$$

La funzione f è concava e non differenziabile. Per ogni dato valore delle variabili x, f è lineare in $\underline{\lambda}$. Il numero di variabili x è limitato da $|\mathcal{V}||\mathcal{N}|^2$. Dato che esse possono prendere solo i valori 0 e 1, il numero di soluzioni possibili nelle variabili x è limitato da $2^{|\mathcal{V}||\mathcal{N}|^2}$, che è generalmente molto grande ma

finito. Dato che il ESPPTWCC è rilassato al SPPTWCC, ogni arco della rete può essere usato diverse volte (in caso di cicli), così x_{ijk} non è limitato da 1, ma è ancora intero e limitato, così l'analisi non è fondamentalmente cambiata da questa. Dato che il numero di soluzioni nelle variabili x è finito, f è il minimo di un numero molto grande di funzioni lineari. Di conseguenza f è lineare a tratti. Massimizzare f è certamente un compito non banale, ma possono essere applicati diversi metodi [94]. Se f è limitata superiormente, il problema originale è chiaramente irrealizzabile. Comunque, l'opposto non è vero: il problema può essere irrealizzabile anche se f è limitata superiormente.

Spesso lo scarto di interezza tra la soluzione frazionaria e quella intera del VRPTW viene in larga parte chiuso nel SPPTWCC. In molti casi non è nemmeno necessario applicare un metodo branch-and-bound, una volta trovati i moltiplicatori di Lagrange ottimi. Quando invece la ramificazione è necessaria, gli autori scelgono di ramificare sulle finestre temporali.

Con tale approccio Kohl e Madsen sono riusciti a risolvere problemi coinvolgenti fino a 100 clienti.

Approccio K-Tree

Fisher et al. [38] lo stesso anno (1997) presentano un algoritmo per risolvere all'ottimalità il VRPTW in cui il problema è formulato come un problema di K-tree con grado $2K$ sul deposito. Un K-tree per un grafo contenente $n+1$ vertici è un insieme di $n+K$ archi che attraversano il grafo. Non formalmente, il VRPTW potrebbe essere descritto come il problema di trovare un K-tree con grado $2K$ sul deposito, grado 2 sui clienti e soggetto a vincoli di tempo e di capacità. Un K-tree con grado $2K$ sul deposito diventa quindi uguale a K rotte.

Per definire il problema introduciamo le variabili binarie x ed y:

$$x_{ij} = \begin{cases} 1 & \text{se il veicolo viaggia direttamente dal cliente } i \text{ al cliente } j \\ 0 & \text{altrimenti} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{se } x_{ij} = 1 \text{ oppure } x_{ji} = 1 \\ 0 & \text{altrimenti} \end{cases}$$

Siano X l'insieme di tutte le variabili x_{ij} ed Y l'insieme di tutte le variabili y che definiscono un K-tree con grado $2K$ sul deposito, cioè:

$$Y = \left\{ y : y_{ij} \in \{0,1\}, y \text{ definisce un } K\text{-tree con } \sum_{i+1}^{n} y_{0i} = 2K \right\}.$$

Siano poi $k(S)$ un limite inferiore sul numero di veicoli richiesti per servire i clienti in $S \subseteq \mathcal{C}$, $\hat{S} = \mathcal{N} \setminus S$ e $P = \langle i_1, i_2, \ldots, i_{m_p} \rangle$ un cammino nel quale non c'è modo di distribuire i clienti soddisfacendo le finestre temporali.
Allora il problema è definito come segue:

$$\min_{x \in X, y \in Y} \sum_{i,j \in \mathcal{N}, i \neq j} c_{ij} x_{ij} \qquad s.a \tag{4.11}$$

$$\sum_{i \in \mathcal{N}, i \neq j} x_{ij} = \begin{cases} 1 & \text{se } j \in \mathcal{C} \\ k & \text{altrimenti} \end{cases} \tag{4.12}$$

$$\sum_{j \in \mathcal{N}, j \neq i} x_{ij} = \begin{cases} 1 & \text{se } i \in \mathcal{C} \\ k & \text{altrimenti} \end{cases} \tag{4.13}$$

$$\sum_{i \in \mathcal{S}} \sum_{j \in \hat{S}} x_{ij} \geq k(S) \qquad \forall\, S \subseteq \mathcal{C}, |S| \geq 2 \tag{4.14}$$

$$\sum_{h=1}^{m_p - 1} x_{i_h, i_{h+1}} \leq m_p - 2 \qquad \forall\, p \in P \tag{4.15}$$

$$y_{ij} = x_{ij} + x_{ji} \qquad \forall\, i, j \in \mathcal{N} \tag{4.16}$$

$$x_{ij} \in \{0, 1\} \tag{4.17}$$

$$y_{ij} \in \{0, 1\} \tag{4.18}$$

Tutti i vincoli, eccetto quelli che assicurano che al massimo un arco congiunge i clienti i e j, vengono allora rilassati in modo lagrangiano. Il problema è allora risolto con un problema di K-tree di grado (vincolato) minimo come sottoproblema ed i moltiplicatori di Lagrange sono fissati usando l'approccio di sub-gradiente. Queste idee erano già state abbozzate da Fisher in precedenza [36] come estensione di un algoritmo ottimo per il VRP, ma senza essere allora implementate. Dato che i vincoli in (4.14) e (4.15) sono in numero esponenziale, solo un sottoinsieme di questi viene generato e dualizzato per ottenere un rilassamento lagrangiano.

Come detto, per dati moltiplicatori di Lagrange il problema principale è un problema di K-tree con vincolo di grado minimo e Fisher propone un algoritmo polinomiale per risolverlo [37]. Tale algoritmo inizializza i moltiplicatori di Lagrange, costruisce un albero di estensione minima ed aggiunge i K archi non utilizzati di costo minimo. Se il deposito non è incidente a $2K$ archi, allora il K-tree viene modificato da una serie di scambi di archi fino a che il deposito arriva ad avere grado $2K$. Gli archi incidenti al deposito vengono

rimossi. Se i vincoli rilassati sono violati, allora vengono aggiornati i moltiplicatori di Lagrange e se avvengono più iterazioni si torna al principio della procedura, altrimenti si effettua il branch-and-bound e poi si torna all'inizio della procedura. Se, invece, i vincoli rilassati non sono violati, allora si ottiene una soluzione ottima al problema principale.

Questo metodo è stato in grado di risolvere all'ottimalità diversi problemi coinvolgenti fino a 100 clienti.

Metodo di Variable Splitting

Il metodo di variable splitting per la risoluzione del VRPTW è stato proposto per la prima volta nel 1986 da Jörnsten et al. [66], ma senza risultati computazionali. Due anni più tardi in [76] Madsen menziona, senza però implementare o testare, quattro diversi approcci di variable splitting. In seguito, nella sua Tesi di Dottorato, Halse [57] (nel 1992) analizza tre di questi approcci (il quarto risulta non essere competitivo con gli altri), implementandone però uno solo.

Nel metodo di variable splitting le variabili in qualche vincolo vengono "rinominate" ed un nuovo vincolo, che associa le variabili originali e quelle nuove, viene aggiunto e rilassato in modo lagrangiano. Questo scompone il problema in due o più sottoproblemi indipendenti. Si può anche scegliere di includere certi vincoli in diversi o in tutti i sottoproblemi.

Supponiamo che il metodo di variable splitting scomponga il problema in un sottoproblema caratterizzato dall'insieme di vincoli A ed in un sottoproblema caratterizzato dall'insieme di vincoli B. LR_A ed LR_B siano il miglior limite inferiore ottenibile da un ordinario rilassamento lagrangiano rispettivamente sugli insiemi di vincoli A e B. Guignard e Kim [53] provano che il limite inferiore LD ottenibile dal metodo di variable splitting è almeno buono quanto il migliore tra LR_A ed LR_B, cioè $LD \geq \max(LR_A, LR_B)$, considerando un problema (primale) di minimizzazione. Inoltre, essi provano che vale l'uguaglianza se almeno uno dei due sottoproblemi possiede la proprietà di interezza. In altre parole, il limite ottenuto dal metodo di variable splitting può solo essere migliore del miglior rilassamento lagrangiano, se la variable splitting produce due problemi interi "difficili".

Nel primo approccio di variable splitting al VRPTW descritto da Halse

[57], il metodo è eseguito sostituendo $\sum_j x_{ijk}$ con y_{ik}. Il vincolo

$$y_{ik} = \sum_{j \in \mathcal{N}} x_{ijk} \quad \forall\, i \in \mathcal{N}, \forall\, k \in \mathcal{V} \tag{4.19}$$

viene introdotto nel problema.

Prima di tutto, si riscrivono i vincoli (4.2) e (4.3) usando y_{ik} al posto di x_{ijk}:

$$\sum_{k \in \mathcal{V}} y_{ik} = 1 \quad \forall\, i \in \mathcal{C} \tag{4.20}$$

$$\sum_{i \in \mathcal{C}} d_i y_{ik} \leq q \quad \forall\, k \in \mathcal{V} \tag{4.21}$$

Si introducono anche i vincoli

$$y_{ik} \in \{0, 1\} \tag{4.22}$$

Si noti che (4.19) è il solo vincolo che associa (4.20) e (4.21) a (4.4),. . ., (4.9). Se vengono rilassati solo i vincoli (4.19) si ottiene la funzione obiettivo

$$\sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} c_{ij} x_{ijk} + \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{C}} \lambda_{ik} (y_{ik} - \sum_{j \in \mathcal{N}} x_{ijk}) \tag{4.23}$$

da minimizzare.

Il modello del problema è quindi dato dalla minimizzazione della funzione (4.23), soggetta ai vincoli (4.20), (4.21) e (4.22) riguardanti le nuove variabili y ed i vincoli (4.4),. . ., (4.9) del modello di partenza, sulle variabili x ed s. Il problema può essere ora diviso in due sottoproblemi, uno nelle variabili x ed s ed uno nelle variabili y. Il primo sottoproblema è caratterizzato da

$$\min \left\{ \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} (c_{ij} - \lambda_{ik}) x_{ijk} \right\}$$

soggetta ai vincoli di rete e a quelli temporali, cioè ai vincoli (4.4), . . ., (4.9). L'altro sottoproblema è caratterizzato invece da

$$\min \left\{ \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{C}} \lambda_{ik} y_{ik} \right\}$$

soggetta ai vincoli di assegnamento, a quelli di capacità ed ai vincoli sui valori realizzabili per le y, cioè ai vincoli (4.20), (4.21) e (4.22). Il problema nelle variabili x ed s è un SPPTW, mentre quello nelle variabili y è un GAP.

Il SPPTW è un problema che, rispetto alla difficoltà computazionale, è strettamente legato al problema SPPTWCC per il quale esiste un algoritmo efficiente di PD. Il GAP è un problema di Ottimizzazione Combinatoria piuttosto difficile da risolvere, ma esistono diversi metodi per farlo.

Questo primo approccio è stato implementato nel 1988 da Olsen [83], che è riuscito in questo modo a risolvere problemi coinvolgenti fino a 16 clienti. Si tratta di quindi un approccio sicuramente non competitivo con gli altri presentati fino ad ora, la maggior parte dei quali in grado di risolvere problemi anche con 100 clienti.

Un altro modo di scomporre il problema è quello descritto da Fisher et al. [38] e messo a punto anche da Halse [57], che consiste nello spostare i vincoli di capacità (4.21) dal secondo sottoproblema, quello nelle variabili y, al primo cioè quello nelle variabili x ed s. Il primo sottoproblema diventa in questo modo un SPPTWCC, mentre il secondo diventa un SAP, cioè un GAP senza vincoli di capacità. Per il primo problema si può ricorrere al solito algoritmo per il SPPTWCC, mentre il secondo può essere facilmente risolto per controllo.

Con tale approccio Fisher et al. [38] sono stati in grado di risolvere problemi coinvolgenti fino a 100 clienti, mentre Halse [57] è riuscito ad ottenere risultati migliori risolvendo un numero maggiore di problemi ed anche un problema con 105 clienti.

L'ultimo approccio descritto da Halse in [57] consiste nel duplicare i vincoli di capacità e nell'inserirli in ognuno dei due sottoproblemi. Si ottengono in questo modo un SPPTWCC ed un GAP. Tale approccio non è stato però implementato.

Dato che tutti i metodi costruiti sul rilassamento lagrangiano usano rilassamenti (risolvono, per esempio, SPPTW o SPPTWCC al posto di ESPPTW o ESPPTWCC), in essi deve necessariamente essere implementata anche una struttura di branch-and-bound. Kohl [69]) mostra che se esistono soluzioni realizzabili al problema VRPTW, allora il limite inferiore ottenuto da GAP e ESPPTWCC non è migliore di quello ottenuto da SAP e ESPPTWCC, dimostrando quindi che risolvere il più difficile GAP al posto del più facile SAP non produce limiti migliori.

4.5.3 Metodi di Generazione di Colonne

Il Primo Approccio Sviluppato

Desrosiers et al. [26] nel 1992 hanno applicato la generazione di colonne, che noi chiameremo decomposizione di Dantzig-Wolfe, al VRPTW con un numero illimitato di veicoli disponibili. Essi risolvono con la decomposizione di Dantzig-Wolfe il rilassamento lineare della formulazione di set partitioning del VRPTW: colonne realizzabili vengono aggiunte al bisogno risolvendo un SPPTWCC usando la PD. La soluzione ottenuta fornisce generalmente un eccellente limite inferiore che viene usato in un algoritmo di branch-and-bound per risolvere la formulazione intera di set partitioning.

La generazione di colonne è un metodo che evita di considerare esplicitamente tutte le variabili di un problema. Dato, ad esempio, un programma lineare con un grande numero di variabili, il metodo consente di risolvere il problema considerando solo un piccolo sottoinsieme X' dell'insieme X delle variabili. Il problema risultante è il problema principale (master problem) ed una volta risolto questo, ci si chiede se esista qualche variabile in $X \backslash X'$ che possa essere usata per migliorare la soluzione trovata.

La teoria della dualità fornisce come condizione necessaria che la scelta appropriata sia una variabile con costo ridotto negativo e l'algoritmo del simplesso tenta di trovare una tale variabile calcolando esplicitamente il costo ridotto di tutte le variabili. L'idea della generazione di colonne è invece quella di trovare le variabili con costi ridotti negativi senza enumerare esplicitamente tutte le variabili. La ricerca per le variabili con costi ridotti negativi è eseguita nel sottoproblema.

Nell'applicazione al VRPTW, la decomposizione di Dantzig-Wolfe sfrutta il fatto che solo i vincoli (4.2) legano insieme i veicoli. Il metodo parte con una soluzione (iniziale di base) che consiste in $|\mathcal{C}|$ cammini, ognuno dei quali visita un cliente. Il problema principale consiste nel trovare un insieme di cammini di costo minimo, tra tutti i cammini generati, il quale allo stesso tempo assicuri che tutti i clienti siano visitati. Se il numero di veicoli è limitato, questo vincolo può anche essere incorporato nel problema principale. Il numero di volte in cui un cammino è usato non è necessariamente un intero, ma può essere ogni numero reale nell'intervallo $[0, 1]$.

Dunque, il problema principale è il rilassamento continuo di un problema di set partitioning, possibilmente con vincoli extra. Per ragioni implementative,

gli autori risolvono invece un problema di set covering, dato che il sottoproblema può generare rotte contenenti cicli. Inoltre, il rilassamento lineare di un modello di tipo set covering è numericamente molto più stabile di quello del modello di tipo set partitioning. Comunque, considerare la formulazione di set covering del VRPTW piuttosto di quella di set partitioning non cambia il risultato finale se la disuguaglianza triangolare risulta soddisfatta sia sul tempo che sui costi (come abbiamo assunto in questo lavoro).

L'idea di base è quindi quella di scomporre il problema in insiemi di clienti visitati dallo stesso veicolo, vale a dire in un insieme di rotte, e selezionare l'insieme ottimo di rotte tra tutti quelli possibili.

Per formulare il VRPTW sulla rete $\mathcal{G}$ come un problema di set partitioning, indichiamo con R l'insieme di rotte realizzabili per il VRPTW e con c_r il costo della rotta r, cioè il costo di visitare i clienti presenti su tale rotta (e quindi la somma del costo degli archi della rotta). Infine, definiamo le costanti δ_{ir} e la variabile binaria x come segue:

$$\delta_{ir} = \begin{cases} 1 & \text{se il cliente } i \text{ appartiene alla rotta } r \\ 0 & \text{altrimenti} \end{cases}$$

$$x_r = \begin{cases} 1 & \text{se la rotta } r \text{ è usata} \\ 0 & \text{altrimenti} \end{cases}$$

Il problema principale, cioè il problema di set partitioning, può essere allora matematicamente dichiarato come segue:

$$\begin{aligned} \min \sum_{r \in \mathcal{R}} c_r x_r \qquad & s.a \\ \sum_{r \in \mathcal{R}} \delta_{ir} x_r = 1 \qquad & \forall\, i \in \mathcal{C} \\ x_r \in \{0,1\} \qquad & \forall\, r \in \mathcal{R} \end{aligned}$$

Le colonne rappresentate dalle variabili corrispondono alle rotte realizzabili. Dato che il loro numero è estremamente grande per tutti i problemi tranne quelli di dimensione veramente piccola, il problema di set partitioning non può essere risolto direttamente, cioè con approcci che coinvolgono l'enumerazione esaustiva delle colonne; invece, si usa un metodo di generazione di colonne.

Per introdurre il modello di tipo set covering usato dagli autori al posto di quello di set partitioning, indichiamo con γ_{ir} una costante che prende un valore intero se la rotta $r \in R$ visita il cliente $i \in \mathcal{C}$, 0 altrimenti. La costante γ_{ir} indica

che un cliente i può essere visitato più di una volta dalla rotta r. In questa formulazione c_r e x_r hanno lo stesso significato assunto nella formulazione di set partitioning. Infine, siano X_d e X_c due variabili intere addizionali: la prima definisce il numero di rotte, mentre la seconda rappresenta la distanza totale percorsa. Si noti che X_c è intero solo se c_r è intero per ogni $r \in R$. Si soddisfa tale condizione prendendo c_{ij} intero per $(i,j) \in \mathcal{A}$.

Il problema di set covering può allora essere matematicamente dichiarato come segue:

$$\min \sum_{r \in R} c_r x_r \qquad s.a \tag{4.24}$$

$$\sum_{r \in R} \gamma_{ir} x_r \geq 1 \qquad \forall\, i \in \mathcal{C} \tag{4.25}$$

$$\sum_{r \in R} x_r - X_d = 0 \tag{4.26}$$

$$\sum_{r \in R} c_r x_r - X_c = 0 \tag{4.27}$$

$$x_r \in \{0,1\} \qquad \forall\, r \in R \tag{4.28}$$

$$X_d \geq 0, \quad X_d \in \mathbb{Z} \tag{4.29}$$

$$X_c \geq 0, \quad X_c \in \mathbb{Z} \tag{4.30}$$

Tale modello è qualche volta detto "problema principale ristretto", dato che l'insieme delle colonne è ristretto alle colonne generate (invece che comprendere tutte le colonne realizzabili).

La funzione obiettivo (4.24) ed i vincoli (4.25) e (4.28) formano un problema di set covering che seleziona un insieme di rotte di costo (distanza) minimo tale che per ogni cliente i esista almeno una rotta che visiti tale cliente. I vincoli (4.26) e (4.27) assicurano invece che siano interi rispettivamente il numero di rotte e la distanza totale percorsa.

Ottimizzare il programma lineare usando le colonne correnti costituisce la prima fase della procedura di generazione di colonne. Nella seconda fase, viene risolto un sottoproblema per trovare la colonna di costo marginale minimo. Se è trovata una colonna con costo marginale negativo, questa variabile viene aggiunta a quelle note e si ritorna alla prima fase; altrimenti, la soluzione corrente è ottima.

Nella prima fase viene usato il metodo del simplesso per risolvere il rilassamento lineare del problema di set covering. La risoluzione del problema

produce una soluzione primale che può essere intera o meno. Se essa è intera, allora è stata trovata una soluzione realizzabile (ma non necessariamente ottima) al VRPTW. Si ottiene anche una soluzione duale di moltiplicatori del simplesso. Il valore ottimo della funzione obiettivo del problema principale (ad ogni iterazione intermedia) può essere interpretata solo come il corrente limite superiore sul migliore limite inferiore ottenibile (dalla decomposizione di Dantzig-Wolfe).

Il metodo del simplesso fornisce le variabili duali π_i $(i \in \mathcal{C}), \pi_d$ e π_c, associate rispettivamente ai vincoli (4.25), (4.26) e (4.27), necessari per la soluzione del sottoproblema. I moltiplicatori del simplesso della soluzione ottima del problema principale vengono usati per modificare i costi degli archi nei sottoproblemi. Il costo marginale della rotta r è dato quindi da:

$$\hat{c}_r = c_r - \sum_{i \in \mathcal{C}} \pi_i \gamma_{ir} - \pi_d - \pi_c c_r.$$

Il costo marginale di un arco (i, j) è definito da

$$\hat{c}_{ij} = (1 - \pi_c) c_{ij} - \pi_i, \quad \forall\, (i, j) \in \mathcal{A}.$$

I sottoproblemi sono degli ESPPTWCC, tutti identici tra loro; basta quindi risolverne solo uno. Il sottoproblema viene quindi rilassato diventando un SPPTWCC. Se si trova un cammino con costo marginale negativo, allora esso viene incluso nell'insieme di cammini nel problema principale. Dato che si potrebbero trovare diversi cammini di costo marginale negativo, ogni iterazione può generare diversi cammini [69].

Il metodo termina quando nel sottoproblema non può essere generato alcun cammino di costo marginale negativo. A questo punto, l'algoritmo del simplesso fornisce la soluzione ottima del rilassamento lineare della formulazione di tipo set covering. Il valore della funzione obiettivo del problema dà un limite inferiore sul valore ottimo della funzione obiettivo del VRPTW. Tutti i cammini utilizzati nella soluzione ottima al problema principale (corrispondenti alle variabili di base nel problema principale) avranno costo marginale nullo, mentre tutti gli altri cammini avranno costo marginale non negativo. Dato che $|\mathcal{C}|$ variabili saranno variabili di base, almeno $|\mathcal{C}|$ rotte avranno costo marginale nullo. Se il problema principale risulta avere una soluzione intera, essa è ottima per il VRPTW; altrimenti si ha un limite inferiore sul valore ottimo della funzione obiettivo del VRPTW e deve essere utilizzata una qualche

procedura di branch-and-bound per trovare la soluzione ottima. Si noti che è essenziale ramificare sulle variabili del problema originale, cioè sulle variabili x ed s del modello per il VRPTW, dato che una decisione di ramificazione del tipo $x_r = 0$ non è computazionalmente implementabile direttamente [69].

Tale approccio ha reso possibile la risoluzione di problemi coinvolgenti fino a 100 clienti.

Lo Sviluppo di Disuguaglianze Valide

Nel 1999 Soumis et al. [30] introducono una forte disuguaglianza valida, la cosiddetta *2-path cut*, per produrre limiti inferiori migliori per il VRPTW, da incorporare nel problema principale di un approccio di decomposizione di Dantzig-Wolfe simile a quello presentato da Desrosiers et al. [26]. Essi sviluppano anche un efficiente algoritmo per trovare tali disuguaglianze.

I contributi principali di questo lavoro sono i seguenti:

- La formulazione generale di disuguaglianze k-path e l'utilizzo del tipo specifico di disuguaglianze 2-path per il VRPTW.

- Lo sviluppo di un algoritmo di separazione per trovare disuguaglianze 2-path. Tale algoritmo sfrutta il fatto che sebbene il TSPTW sia $\mathcal{NP}$–arduo, piccole istanze sono generalmente facilmente risolvibili.

- Illustrare come innestare i tagli (e/o rami) in un metodo di generazione di colonne.

- Mettere in luce l'efficacia delle disuguaglianze 2-path per mezzo di risultati computazionali migliori di quelli precedentemente pubblicati.

Quasi tutti i metodi di ottimizzazione per problemi difficili di PLI sfruttano un rilassamento (e spesso una decomposizione) del problema per calcolare limiti sul valore ottimo dell'obiettivo. Spesso la soluzione del problema rilassato non risulta realizzabile per il problema originale, così non si può ottenere direttamente dal rilassamento alcuna soluzione ottima.

Le tecniche di branch-and-bound hanno rappresentato uno strumento fondamentale per ottenere soluzioni realizzabili intere e provarne l'ottimalità. Nel recente passato, una notevole attenzione è stata rivolta a metodi branch-and-cut basati sul miglioramento di un limite stringendo il rilassamento il più possibile

prima della ramificazione. Il limite viene stretto incorporando disuguaglianze valide violate nella descrizione poliedrale dello spazio delle soluzioni.

Da tempo sono noti metodi generali per calcolare disuguaglianze valide per programmi interi, ma algoritmi molto più efficienti possono essere ottenuti sfruttando la struttura del particolare problema da risolvere.

La forma generale di una disuguaglianza valida per il VRPTW è data da

$$\sum_{k\in\mathcal{V}}\left(\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\alpha_{ijk}x_{ijk}+\sum_{i\in\mathcal{N}}\beta_{ik}s_{ik}\right)\geq\gamma,$$

dove α e β sono appropriati vettori e γ una costante data.

Gli autori considerano solo disuguaglianze valide definite sulle variabili x; inoltre, dato che i veicoli sono tutti identici, essi scelgono di aggregarli alle variabili x e questo conduce alla seguente disuguaglianza valida piuttosto semplice:

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\alpha_{ij}x_{ij}\geq\gamma. \tag{4.31}$$

Il problema di separazione consiste nel trovare α_{ij} e γ in modo tale che la (4.31) sia una disuguaglianza valida e che essa sia violata dalla soluzione corrente $\bar{x}_{ij}$, valga cioè

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\alpha_{ij}\bar{x}_{ij}<\gamma.$$

Per presentare le disuguaglianze k-path per il VRPTW definiamo una rete di flusso $\mathcal{F}=(\mathcal{N},\mathcal{A}')$, dove $\mathcal{A}'=\{(i,j)\in\mathcal{A}:\bar{x}_{ij}>0\}$ e la capacità del flusso dell'arco $(i,j)\in\mathcal{A}'$ è data dalla soluzione corrente $\bar{x}_{ij}$. Per ogni insieme di valori delle variabili $\bar{x}_{ij}$ e per ogni insieme non vuoto di clienti, $S\subseteq\mathcal{C},|S|\geq 1$, si definisce *flusso in* S la quantità

$$\bar{x}(S)=\sum_{i\in\bar{S}}\sum_{j\in\mathcal{C}}\bar{x}_{ij},\quad\forall S\subseteq\mathcal{C},$$

dove $\bar{S}=\mathcal{N}\setminus S$.

Il vincolo di eliminazione di sottocammini non può essere formulato in termini della variabile originale del problema x come $x(S)\geq 1,\forall\, S$. Per il VRPTW si sostituisce il lato destro della disuguaglianza con $k(S)$, che rappresenta il più piccolo numero di veicoli necessari per servire tutti i clienti in S. Il vincolo $x(S)\geq k(S)$ è detto taglio *k-path*.

Comunque, i tagli k-path non sembrano essere davvero utili computazionalmente dato che:

- Il numero di insiemi S cresce esponenzialmente.
- Calcolare $k(S)$ è un problema $\mathcal{NP}$–arduo in senso forte.

Conviene allora limitarsi a considerare insiemi che soddisfino $\bar{x}(S) < 2$ e $k(S) > 1$. In questo modo il problema diventa più maneggevole. Dalla definizione, $k(S)$ è intero, così $k(S) > 1$ implica $k(S) \geq 2$. In altre parole, si tenta di identificare insiemi di clienti S che richiedono almeno due veicoli per il servizio, ma sono correntemente (nella soluzione frazionaria) serviti da meno di due veicoli. Le disuguaglianze valide associate sono i tagli *2-path*.

Verificare se $k(S) > 1$ per un particolare S equivale a determinare se la capacità di un veicolo è sufficiente, il che può essere fatto in tempo lineare, e verificare se il corrispondente TSPTW è realizzabile. La realizzabilità del TSPTW è un problema $\mathcal{NP}$–arduo in senso forte, ma nell'applicazione presente in [30] $|S|$ risulta veramente piccolo, così che il problema è relativamente facile da risolvere con la PD [28]. Si ha quindi a disposizione un algoritmo veloce, sebbene non polinomiale, per determinare se $k(S) > 1$.

Per trovare insiemi S che soddisfino $\bar{x}(S) < 2$ vengono proposti due metodi: un'euristica ed un metodo ottimo. Nel caso in cui la rete sia aciclica, l'euristica diventa un metodo esatto trovando tutti gli insiemi S che soddisfano $\bar{x}(S) < 2$. Nessuno dei due metodi ha complessità polinomiale nel caso peggiore, ma il loro comportamento nel contesto studiato risulta comunque ragionevole. Una volta trovate le disuguaglianze valide, queste possono essere inserite in un approccio di decomposizione di Dantzig-Wolfe.

Tale approccio è stato in grado di risolvere problemi coinvolgenti fino a 100 clienti.

4.6 Approccio Esatto Ibrido di OR e CP

Nel 2004 Gendreau et al. [39] propongono un metodo esatto ibrido per la risoluzione del VRPTW. L'idea alla base di questo lavoro è nata dalla collaborazione di diversi studiosi di OR e di AI durante un vasto progetto risalente alla seconda metà degli anni '90 e riguardante lo sviluppo di approcci ibridi dei due campi per la risoluzione di complessi problemi di Ottimizzazione Combinatoria.

Nello stesso contesto e gruppo di ricerca, nel 1997 Guadin [52] presenta un altro progetto incentrato sull'applicazione del paradigma puro della CP ad un problema di vehicle routing con finestre temporali molto complicato, in un ambiente di trasporto multi-modale, in cui cioè sono possibili diverse modalità di trasporto. Questa caratteristica del problema affrontato esclude tale approccio dalla nostra discussione, incentrata in particolare su metodi esatti per problemi di routing con flotte omogenee. Comunque, la dissertazione di Guadin si dimostra molto interessante e mostra chiaramente gli ottimi risultati ottenibili con l'utilizzo del paradigma di CP nella risoluzione di problemi complessi. A tale lavoro va anche il merito di aver messo in luce le maggiori particolarità del paradigma della CP: la particolare espressività, che permette di descrivere in modo naturale problemi anche molto complessi, e la sua grande flessibilità, caratteristica strettamente connessa alla prima. Tali risultati hanno contribuito al crescente interesse di quel periodo per la CP e per la sua possibile integrazione con la OR, dal quale è scaturito anche il metodo ibrido in esame.

Un anno più tardi rispetto alla pubblicazione di Guadin, gli stessi autori del metodo ibrido in esame propongono un algoritmo esatto di CLP per il TSPTW [47]. Questo progetto coinvolge l'applicazione abbastanza semplice della metodologia della CP, presentando però un elemento chiave, cioè l'uso di regole derivate da precedenti studi di OR sul TSPTW per ridurre le finestre temporali ed eliminare gli archi non realizzabili dalla rete sottostante.

Negli algoritmi tradizionali di OR queste regole di eliminazione di archi e riduzione di finestre temporali sono applicate "a priori" sui dati del problema per ridurre l'insieme realizzabile di una istanza, prima di iniziare la procedura di risoluzione. Nella struttura della CP è possibile invece modellare queste regole come vincoli addizionali ed applicarli così in modo dinamico durante l'esecuzione dell'algoritmo. Di conseguenza, esse diventano molto più efficaci ed il risultante algoritmo si dimostra assai efficiente e competitivo con gli altri algoritmi per il TSPTW sviluppati nel campo della OR, in particolare con quelli basati sulla PD.

Questo lavoro è stato il primo che utilizzasse la CP pubblicato sulle riviste di OR ed è stato successivamente esteso per trattare problemi di TSPTW con finestre temporali frammentate, un problema assai difficile da trattare con le tecniche standard di OR. Tale estensione ha mostrato ancora una volta la grande flessibilità degli approcci di CP, dato che sono state sufficienti poche

modifiche dell'algoritmo in [47] per trattare strutture di finestre temporali molto più complicate.

Nello sviluppo del metodo ibrido per il VRPTW in esame, molti sono i riferimenti a questo approccio al TSPTW ed anche ad una struttura ibrida di ottimizzazione proposta qualche anno prima da Fahle et al. [35], la cosiddetta *Programmazione con Vincoli basata sulla generazione di colonne*, che usa la CP per risolvere i sottoproblemi della generazione di colonne.

La novità presentata in tale struttura è che il sottoproblema può essere un arbitrario CSP, il che presenta due grandi vantaggi: prima di tutto, si generalizza la classe dei sottoproblemi permettendo così l'uso della generazione di colonne anche se il sottoproblema non si riduce ad un programma lineare intero misto; in secondo luogo, si possono sfruttare le tecniche di soddisfazione di vincoli per risolvere il sottoproblema.

L'idea di base è davvero semplice. Il problema principale è un problema intero misto con un insieme di vincoli lineari ed una funzione lineare di costo; le colonne (o variabili) non sono date esplicitamente. Il sottoproblema è un arbitrario CSP. Per ogni soluzione del sottoproblema esiste una variabile del problema principale. Naturalmente, occorre conoscere i coefficienti della variabile in tutti i vincoli lineari del problema principale e nella sua funzione di costo. Per ognuno di questi coefficienti a_{ij} si introduce una variabile y_i corrispondente nel sottoproblema. Inoltre, si introduce una variabile z per il coefficiente c_j nella funzione di costo. Data una soluzione v_j del sottoproblema, il coefficiente a_{ij} della variabile x_i nell'i-esimo vincolo lineare è allora ottenuto come il valore della variabile y_i nella soluzione data. Rappresentare i coefficienti attraverso le variabili del sottoproblema permette anche di assicurare che le soluzioni del sottoproblema abbiano costi ridotti negativi. Data una soluzione del rilassamento lineare del problema principale, si considerano i valori duali λ_i di ogni vincolo i. Allora si introduce semplicemente un vincolo lineare nel sottoproblema che è formulato su z, sulle y_i e su quelle che usano i valori duali come coefficienti.

Nel passato, tale struttura è stata usata per risolvere problemi di scheduling in cui il grafo associato è in modo naturale aciclico ed i risultati sono stati veramente considerevoli. Il metodo in esame cerca invece di usare tale struttura per risolvere problemi in cui il grafo associato è ciclico per natura, come per i problemi di routing, risolvendo il ESPP, sottoproblema derivante dall'approccio di generazione di colonne, attraverso la CP.

La maggior parte dei metodi di generazione di colonne fa uso della PD per risolvere il SPP in cui i vincoli elementari (i vincoli che assicurano che il cammino non può attraversare lo stesso nodo più di una volta) sono stati rilassati. Tale approccio è veramente efficiente, ma dato che il problema ammette pesi negativi sugli archi, il cammino prodotto può contenere cicli (dato che cicli di costo negativo diminuiscono la funzione obiettivo). Dato che i problemi di routing sono ciclici per natura, la maggior parte dei metodi che riguardano i casi ciclici rendono prima di tutto il grafo associato aciclico. Sfortunatamente, questa trasformazione richiede che le diverse risorse (le finestre temporali, le capacità, etc.) siano discrete e la dimensione del grafo risultante è solitamente assai impressionante (pseudo-polinomiale nel peso delle risorse).

Da qui il tentativo di applicare anche a questi casi la struttura di CP basata sulla generazione di colonne. Gli autori considerano un dominio di applicazione di routing, concentrandosi sui *cyclic resource constrained shortest path problems*, noti anche con il nome di *resource constrained Profitable Tour Problems* (PTPs). L'obiettivo è quello di costruire un tour che minimizzi la somma delle distanze percorse e massimizzi l'ammontare totale di compensi (qui, valori duali) raccolti. Questi obiettivi sono in conflitto dato che più compensi raccolti implicano più distanze percorse. L'obiettivo combinato è così quello di minimizzare la lunghezza totale delle rotte meno la somma di tutti i valori duali raccolti.

Lo scopo del lavoro di Gendreau et al. [39] è quello di mostrare che i metodi di CP possono identificare cammini elementari di costo ridotto negativo lavorando sul grafo originale ciclico. L'uso della CP permette anche l'aggiunta di ogni forma di vincoli al problema originale (il che non è possibile nel caso dell'approccio di PD).

La motivazione originale alla base dell'utilizzo di una struttura di CP basata sulla generazione di colonne [35] per risolvere problemi di assegnamento di equipaggi di aerei, è stata quella che per alcuni problemi la modellazione attraverso metodi puri di OR è risultata molto complessa. Così, l'uso della CP per risolvere il sottoproblema in un approccio di generazione di colonne (tipico metodo di OR) ha fornito sia il potente strumento di decomposizione della generazione di colonne sia la flessibilità di modelling tipica della CP.

Nel modello del SPP, gli autori di [35] propongono l'uso di un singolo insieme di variabili Y (cioè una variabile il cui valore finale è un insieme) che contiene i nodi che devono essere inclusi in un cammino di costo ridotto

negativo. Dato che il problema discusso in questa struttura è per natura aciclico (la rete sottostante è orientata nel tempo), è facile calcolare in tempo polinomiale il cammino più corto che copre i nodi in Y. Viene anche introdotto un vincolo speciale per migliorare la potatura e l'efficienza del metodo nel suo complesso. Questo vincolo, il quale assicura che i nodi in Y siano parte di un cammino realizzabile, rafforza anche la consistenza del limite risolvendo un SPP sia sugli insiemi richiesti che su quelli possibili di Y. Un'implementazione incrementale dell'algoritmo del shortest path assicura che il filtraggio sia fatto efficientemente.

Problema Principale

Siano r una rotta realizzabile nel grafo originale (che contiene N clienti), R l'insieme di tutte le possibili rotte r, c_r il costo di visitare tutti i clienti in r, $A = (a_{ir})$ una matrice booleana che esprime la presenza del cliente $i \in \{1, \ldots, n\} = \mathcal{C}$ in r ed x_r la variabile booleana che specifica se la rotta r è scelta ($x_r = 1$) o meno ($x_r = 0$).

Il problema di set partitioning (S) è definito quindi come segue:

$$\begin{aligned} \min \sum_{r \in R} c_r x_r \qquad & s.a \\ \sum_{r \in R} a_{ir} x_r = 1 \qquad & \forall\, i \in \mathcal{C}, \\ x \in \{0,1\}^n & \end{aligned}$$

Questa formulazione, comunque, pone qualche problema. Prima di tutto, dato che è impraticabile costruire e memorizzare l'insieme R vista la sua dimensione davvero grande, solitamente si lavora con un insieme parziale R' che viene arricchito iterativamente risolvendo un sottoproblema. In secondo luogo, la formulazione di set partitioning è difficile da risolvere quando R' è piccolo ed ammette valori duali negativi che possono essere problematici per il sottoproblema (un duale negativo significa un costo marginale negativo per servire un nodo).

Per questi motivi, come fatto da Desrosiers et al. [26], al posto della formulazione (S) viene usata come problema principale la seguente formulazione

rilassata di set covering (M):

$$\begin{aligned} \min & \sum_{r \in R'} c_r x_r \qquad s.a \\ & \sum_{r \in R'} a_{ir} x_r \geq 1 \qquad \forall\, i \in \mathcal{C}, \\ & x \geq 0 \end{aligned}$$

Per arricchire R', è necessario trovare nuove rotte che offrano un modo migliore di visitare i clienti che contengono, rotte che presentino un costo ridotto negativo. Il costo ridotto di una rotta è calcolato rimpiazzando il costo d_{ij} di un arco (la distanza tra i due clienti i e j) con il costo ridotto di tale arco $c_{ij} = d_{ij} - \lambda_i$, dove λ_i è il valore duale associato al cliente i. Il valore duale associato ad un cliente può essere interpretato come il costo marginale di visitare quel cliente nella corrente soluzione ottima. L'obiettivo del sottoproblema è allora quello di identificare un cammino di costo ridotto negativo, cioè un cammino per il quale la somma della distanza percorsa è inferiore alla somma dei costi marginali (valori duali). Un tale cammino rappresenta un nuovo e migliore modo di visitare i clienti da servire.

La soluzione ottima di (M) è identificata quando non esiste alcun cammino di costo ridotto negativo. Comunque, questa soluzione può essere frazionaria dato che (M) è un rilassamento di (S) e così non rappresenta la soluzione ottima di (S), ma piuttosto un limite inferiore su questa. Se questo è il caso, è necessario usare uno schema di ramificazione per identificare una soluzione intera.

Modello del Sottoproblema

Dato che il problema considerato in [39] è ciclico, non può essere usato un semplice insieme di variabili per registrare le soluzioni (come proposto in [35]) dato che la costruzione di una soluzione completa dall'insieme di visite incluse richiederebbe di risolvere un TSP. Occorre quindi un nuovo modello.

Sia $N = \{0, 1, \ldots, n\}$ l'insieme di tutti i clienti, compreso il nodo 0 che rappresenta il deposito di partenza, e sia $n + 1$ il nodo che rappresenta il deposito in cui tornano i veicoli dopo il servizio. Sia quindi $N' = \{1, \ldots, n+1\}$ l'insieme di tutti i nodi eccetto il deposito iniziale.

Per descrivere il modello del sottoproblema consideriamo i seguenti parametri:

- d_{ij}, che rappresenta la distanza dal nodo i al nodo j;
- t_{ij}, che rappresenta il tempo necessario per spostarsi da i a j;
- a_i e b_i, che rappresentano rispettivamente i limiti sinistro e destro della finestra temporale relativa al cliente i;
- l_i, che rappresenta il carico da portare al nodo i, cioè la domanda del nodo i;
- λ_i, che rappresenta il valore duale associato al nodo i;
- C, che rappresenta la capacità del veicolo;
- $c_{ij} = d_{ij} - \lambda_i$, che rappresenta il costo ridotto per andare dal nodo i al nodo j.

Consideriamo poi le seguenti variabili:

- $Pr_i \in N \quad \forall\, i \in N'$,
  che rappresenta il predecessore diretto del nodo i;
- $S_i \in N' \quad \forall\, i \in N$,
  che rappresenta il successore diretto del nodo i;
- $T_i \in [a_i, b_i] \quad \forall\, i \in N \cup \{n+1\}$,
  che rappresenta il tempo di visita del nodo i;
- $L_i \in [0, C] \quad \forall\, i \in N \cup \{n+1\}$,
  che rappresenta il carico del veicolo prima della visita al nodo i.

La variabile S identifica il *successore diretto* nel cammino di ogni nodo del grafo, ma per brevità ci riferiremo a tale variabile solo come variabile *successore*. Un nodo i lasciato fuori dal cammino scelto è rappresentato da un cappio e la variabile S_i è in questo caso fissata al valore i. Allo stesso modo ci riferiremo alla variabile Pr come *predecessore* al posto di *predecessore diretto*.

L'obiettivo è quello di minimizzare la somma dei costi ridotti dell'intero percorso, vale a dire

$$\min \sum_{i \in N} c_{iS_i},$$

soggetta ai vincoli:

- `AllDifferent`(S),
  che è il vincolo di conservazione di flusso;

- `NoSubTour`(S),
  che è il vincolo di eliminazione dei sottopercorsi;

- $T_i + t_{iS_i} \leq T_{S_i} \quad \forall\, i \in N$,
  che sono i vincoli di finestre temporali;

- $L_i + l_i = L_{S_i} \quad \forall\, i \in N$,
  che sono i vincoli di capacità.

Gli ultimi due insiemi di vincoli impongono il rispetto della capacità dei veicoli e delle finestre temporali dei clienti propagando l'informazione sul tempo e sul carico quando un nuovo arco diventa noto.

Il vincolo `AllDifferent`(S) è usato per esprimere la conservazione del flusso nella rete. La natura della variabile decisionale già impone che ogni nodo abbia esattamente un arco uscente, ma è anche necessario assicurare che ogni nodo abbia allo stesso modo esattamente un arco entrante. Per far questo è necessario assicurare che nessuna coppia di nodi abbia lo stesso successore ed è proprio quello che impone il vincolo `AllDifferent`(S). Questa proprietà è ottenuta risolvendo un problema di matching in un grafo bipartito e mantenendo la consistenza su archi in un modo incrementale (come descritto in [87]).

Il vincolo `NoSubTour`(S) è preso dal lavoro precedente degli autori che presenta un algoritmo di CLP per il TSPTW [47]. Per ogni catena di clienti, si conserva memoria del primo e dell'ultimo visitati, e quando due catene vengono unite (quando una variabile è fissata ed un nuovo arco è introdotto) si aggiornano le informazioni riguardanti il primo e l'ultimo cliente della catena e si rimuove il valore del primo dal dominio della variabile successore dell'ultimo (impedendo così che si formi un ciclo).

Vincoli Addizionali del Sottoproblema

Per migliorare il tempo di soluzione vengono introdotti anche dei vincoli ridondanti, i quali non modificano l'insieme soluzione ma permettono di migliorare la potatura ed il filtraggio.

Il primo tipo di vincoli ridondanti, preso ancora una volta da [47], esegue un filtraggio basato sul fatto di restringere le finestre temporali. Questi vincoli mantengono per ogni nodo i il tempo di partenza più tardi possibile ed il nodo j associato a tale tempo. Quando il dominio di S_i viene modificato, il vincolo prima di tutto verifica che j sia ancora nel dominio di S_i ed in tal caso non esegue il filtraggio. Questa implementazione permette che il calcolo del tempo realizzabile sia veramente efficiente dato che essa previene controlli ridondanti di soluzioni realizzabili note.

Il secondo tipo di vincoli ridondanti è rappresentato da una famiglia di vincoli che riduce il numero di nodi esplorati dell'albero di ricerca riducendo il numero di archi del grafo del sottoproblema. L'idea è quella di eliminare archi che non saranno certamente presenti nella soluzione ottima del PTP. Una tale pratica è nota come *cost-based filtering* o *optimization constraints* ed è stata introdotta da Focacci, Lodi e Milano in [42].

Nell'approccio in esame vengono proposti due vincoli che eliminano archi che possono ridurre in modo significativo la dimensione del grafo originale sulla base sella seguente idea: se il valore duale associato ad un cliente non è sufficientemente grande, allora può non valere la pena di visitare questo cliente. Ancora, questi vincoli sono validi se e solo se vale la disuguaglianza triangolare per le risorse. Altrimenti, la visita di un cliente intermedio potrebbe produrre risparmi in qualche risorsa, permettendo così la visita di clienti extra. Dato che questa disuguaglianza non vale quando $i = j$, il vincolo non è stato definito sui cappi.

Il primo tipo di vincolo di eliminazione di archi è definito come segue: dato un arco (i, j), se per tutti gli altri clienti k che sono elementi del dominio della variabile successore S_j di j è sempre più economico andare direttamente da i a k piuttosto che passare attraverso j (cioè, $d_{ij} + d_{jk} - \lambda_j > d_{ik}$) , allora l'arco (i, j) può essere eliminato dal grafo del sottoproblema dato che esso non sarà mai parte di una soluzione ottima:

$$\forall\, i \in N, \forall\, j \in S_i :$$

$$j \neq i \quad \text{impone che}$$

$$\big(\forall\, k \in S_j :\, k \neq i \neq j\ (d_{ij} + d_{jk} - \lambda_j > d_{ik})\big) \Longrightarrow S_i \neq j.$$

Il secondo tipo di vincolo di eliminazione di archi è definito come segue: dato un arco (i, j), se per tutti gli altri k clienti la cui variabile successore S_k

include i è sempre più economico andare direttamente da k a j piuttosto che passare attraverso i (cioè, $d_{ki} + d_{ij} - \lambda_i > d_{kj}$), allora l'arco (i, j) può essere eliminato dal grafo del sottoproblema dato che esso non sarà mai parte di una soluzione ottima:

$$\forall\, i \in N, \forall\, j \in S_i :$$

$$j \neq i \quad \text{impone che}$$

$$\big(\forall\, k \in Pr_i :\, k \neq i \neq j\ (d_{ki} + d_{ij} - \lambda_i > d_{kj})\big) \Longrightarrow S_i \neq j.$$

Questi vincoli possono essere applicati prima che sia intrapresa la ricerca per identificare un cammino di costo ridotto negativo, e potrebbero essere così usati in congiunzione con ogni metodo che riguardi il PTP (anche un approccio di PD che risolve un rilassamento del PTP). Comunque, dato che si sta usando il paradigma della CP si può ottenere un'ulteriore potatura applicando questi vincoli durante la ricerca.

Per girare in modo efficiente, questi vincoli di eliminazione di archi devono essere implementati in modo incrementale. Vediamo cosa significa questo per il primo tipo di vincolo, ma il ragionamento può essere facilmente ricondotto al secondo tipo. Per ogni arco (i, j) è mantenuto il valore di k, un successore di j per il quale $d_{ij} + d_{jk} - \lambda_j \leq d_{ik}$. Quando il dominio di S_j viene modificato, il vincolo rende valide due condizioni prima di eseguire qualsiasi filtraggio: prima di tutto, l'arco (i, j) deve ancora essere un possibile arco nella soluzione, cioè il valore j deve ancora essere presente nel dominio di S_i; in secondo luogo, se il valore di k è ancora nel dominio di S_j allora non può esserci alcun filtraggio e l'algoritmo di filtro ritorna. Solo quando l'arco (i, j) è ancora presente e k non è più un possibile successore di j il vincolo cerca un nuovo valore di k; quando non ne viene trovato alcuno, allora l'arco viene rimosso.

Strategie di Ricerca per il Sottoproblema

Per costruire la soluzione è necessario definire delle euristiche di selezione di variabile e di valore. Le strategie di selezione proposte in [39] sono basate sulla cosiddetta *Programmazione con Vincoli basate sulla programmazione dinamica* proposta da Focacci e Milano [44].

In [44] viene definita l'architettura di *vincolo globale orientato all'ottimizzazione*. Accanto alle variabili decisionali del problema $X_1, \ldots, X_n$ coinvolte nel vincolo, un parametro addizionale del vincolo è la variabile Z che rappresenta la funzione obiettivo del problema ed una funzione di costo.

Concettualmente, un vincolo globale di CP incorpora una componente software contenete un algoritmo di filtraggio che pota i domini sulla base di un *ragionamento di realizzabilità*: un valore viene rimosso dal dominio di una variabile se si dimostra irrealizzabile ed ogni vincolo interagisce con gli altri vincoli attraverso la modifica del dominio della variabile coinvolta. Intuitivamente, si vuole eseguire una potatura basata su un *ragionamento di ottimalità*: un valore viene rimosso dal dominio di una variabile se si dimostra sub-ottimale. A questo scopo, si estende l'architettura concettuale di vincolo globale di CP con due componenti addizionali: una componente di ottimizzazione ed un algoritmo di filtraggio basato sul costo.

La componente di ottimizzazione è un risolutore in grado di calcolare la soluzione ottima del problema che è tipicamente un rilassamento del problema rappresentato dal vincolo globale. Questo rilassamento dipende dalla semantica dichiarativa del vincolo globale e dalla funzione obiettivo del problema. In generale, la componente di ottimizzazione è basata su tecniche di OR, su algoritmi con finalità speciali per problemi strutturati o su algoritmi di PL con finalità generali.

La componente di ottimizzazione dovrebbe restituire tre parti di informazione:

- la soluzione ottima s^* del problema rilassato;
- il valore LB della soluzione ottima, il quale rappresenta un limite inferiore sul problema corrispondente al vincolo globale e quindi sul valore della funzione obiettivo totale;
- una funzione $grad(V, v)$ che misura il costo dell'assegnamento variabile-valore, cioè dell'assegnamento del valore v alla variabile V.

Queste informazioni vengono sfruttate sia per finalità di filtraggio che per guidare la ricerca attraverso rami promettenti (in termini di costo) dell'albero.

Una prima (banale) propagazione viene eseguita per imporre la condizione $LB \leq Z$, dove Z è il dominio della variabile che rappresenta la funzione obiettivo (nell'ipotesi di problema di minimizzazione).

Più interessante risulta la propagazione dalla funzione gradiente $grad(X, v)$ verso le variabili decisionali $X_1, \ldots, X_n$. La funzione gradiente fornisce una valutazione ottimistica sul costo di ogni assegnamento variabile-valore. Data questa informazione, si può calcolare una valutazione ottimistica sulla soluzione

ottima di un problema in cui una data variabile è assegnata ad un dato valore, cioè la valutazione $LB_{X=v}$ (si può fare tale calcolo per tutte le variabili e tutti i valori appartenenti ai loro domini). Così, se questa valutazione è più grande della miglior soluzione trovata fino al momento, il valore può essere rimosso dal dominio della variabile. Più formalmente, per ogni valore j del dominio di ogni variabile X_i si può calcolare un valore di limite inferiore del sottoproblema generato se il valore j è assegnato a X_i, $LB_{X_i=j} = LB + grad(X_i, j)$. Se $LB_{X_i=j}$ è più grande o uguale al limite superiore del dominio di Z, j può essere rimosso dal dominio di X_i.

La componente di ottimizzazione e l'algoritmo di filtraggio comunicano attraverso un mapping tra le variabili di CP coinvolte nel vincolo globale ed il modello sfruttato dal risolutore di ottimizzazione.

In tale struttura di CP basata sulla PD viene utilizzata la nozione di *variabili condizionali*, introdotte per modellare metodi di PD in CP. Esse sono usate per definire grandezze che possono avere o non avere senso nella definizione di un problema. Una variabile condizionale è molto simile ad una variabile tradizionale, eccetto per il fatto che essa è associata ad un vincolo di definizione: quando questo vincolo è soddisfatto, la variabile condizionale si comporta come una variabile regolare; quando il vincolo diventa violato, la variabile viene svuotata e tutti i vincoli che la coinvolgono sono banalmente soddisfatti.

Per una definizione più formale, una variabile condizionale V^c è definita dalla coppia (D^c, C^c), dove D^c è il dominio di valori possibili e C^c è un vincolo, anche indicato con $C^c(V^c)$. Il dominio D^c deve contenere almeno un valore se e solo se il vincolo C^c è rispettato, mentre è vuoto altrimenti. La variabile è detta *vera* se il suo vincolo di definizione è vero, *falsa* altrimenti.

Possono essere definiti diversi operatori che coinvolgono le variabili condizionali. Consideriamo prima di tutto il vincolo di uguaglianza definito tra una variabile V di dominio D ed una variabile condizionale $V^c = (D^c, C^c)$: il vincolo $V^c = V$ vale se C^c è falso oppure se C^c è vero e le due variabili prendono lo stesso valore. Il vincolo di uguaglianza può così essere propagato come segue:

$$\begin{aligned} i \notin D \quad &\Longrightarrow \quad V^c \neq i \quad \forall\, i \in D^c, \\ C^c \quad &\Longrightarrow \quad (\, i \notin D^c \Rightarrow V \neq i \quad \forall\, i \in D\,). \end{aligned}$$

In modo simile, questo vincolo di uguaglianza può essere applicato a due variabili condizionali $V_1^c = (D_1^c, C_1^c), V_2^c = (D_2^c, C_2^c)$: il vincolo $V_1^c = V_2^c$ vale se

$C_1^c \wedge C_2^c$ è falso oppure se $C_1^c \wedge C_2^c$ è vero e le due variabili assumono lo stesso valore.

Le variabili condizionali possono anche essere combinate tra loro per mezzo di operatori aritmetici ed il concetto di variabili condizionali è stato anche usato in CP per implementare disgiunzioni esclusive, le quali dicono che solo una variabile dell'insieme di variabili condizionali deve essere vera.

Un vincolo globale può essere definito come segue: $xunion(V^c_{[1,\ldots,k]}, y, x)$, dove y ed x sono variabili regolari e $V^c_{[1,\ldots,k]}$ è un vettore di k variabili condizionali. Il vincolo assicura che esattamente una delle k variabili condizionali sarà vera, imponendo inoltre che tale variabile eguagli la variabile y e definendo la variabile x come l'indice al quale essa è collocata nell'array (cioè, $x \in [1, \ldots, k]$).

A questo punto, in aggiunta al modello del sottoproblema presentato precedentemente, vengono introdotte nuove variabili e nuovi vincoli per implementare un grafo del rilassamento dello spazio degli stati del problema originale. La prima variabile P indica la *posizione* nella quale ogni nodo è visitato, P_i indica quindi la posizione del nodo i nella soluzione ed è settata a 0 quando il nodo i non è visitato nel cammino ottimo (eccetto per il nodo sorgente, che è detto essere visitato in posizione 0).

Per ogni nodo i, viene poi introdotta una nuova variabile condizionale $S^c_{ip} = (N', P_i = p) \quad \forall p \in N$, la quale rappresenta il nodo direttamente successivo al nodo i nel cammino ottimo se i viene visitato nella posizione p. Le variabili condizionali sono qui necessarie dato che S^c_{ip} non ha alcun senso se i non è visitato alla posizione p. Vengono introdotte anche simili variabili condizionali, T^c_{ip} e L^c_{ip}, rispettivamente in relazione alla dimensione temporale e di capacità del problema. Le variabili originali S_i, T_i, L_i sono uguali alla disgiunzione esclusiva delle n variabili condizionali $S^c_{ip}, T^c_{ip}, L^c_{ip}$ dato che il nodo i può essere visitato in una sola posizione. Tali variabili e vincoli aggiuntivi permettono che la ricerca proceda nella modalità della PD.

Le nuove variabili del problema sono quindi le seguenti:

- $P_i \in N \quad \forall i \in N$;
- $S^c_{ip} \in (N', P_i = p) \quad \forall i, p \in N$;
- $T^c_{ip} \in ([a_i, b_i], P_i = p) \quad \forall i, p \in N$;
- $L^c_{ip} \in ([0, C], P_i = p) \quad \forall i, p \in N$.

I nuovi vincoli sono invece i seguenti:

- $P_0 = 0$;
- $S^c_{i0} = i \quad \forall\, i \in N \setminus \{0\}$;
- $S^c_{ip} \neq i \quad \forall\, i \in N, \forall\, p \in N \setminus \{0\}$;
- $xunion(S_{i[0,\ldots,n]}, S_i, P_i) \quad \forall\, i \in N$;
- $xunion(T_{i[0,\ldots,n]}, T_i, P_i) \quad \forall\, i \in N$;
- $xunion(L_{i[0,\ldots,n]}, L_i, P_i) \quad \forall\, i \in N$;
- $(P_j = p) \;\Rightarrow\; \exists i \neq j :\; S^c_{i(p-1)} = j \quad \forall\, i, p \in N$;
- $T^c_{ip} + t_{iS^c_{ip}} \leq T_{S^c_{ip}p+1} \quad \forall\, i \in N$;
- $L^c_{ip} + l_i = L_{S^c_{ip}p+1} \quad \forall\, i \in N$.

Una volta aggiunti tali vincoli al modello originale, è possibile propagare informazioni limite su tutte le nuove variabili (quelle indicizzate dal valore di posizione) nella modalità della PD (come descritto in [44]). Questa informazione è poi usata per guidare il processo di ramificazione sulle variabili originarie (S). La strategia di selezione di variabile tenta di costruire il cammino più corto dal nodo sorgente all'ultimo nodo, selezionando sempre la variabile successore dell'ultimo nodo i che è stato aggiunto al cammino.

L'euristica di selezione del valore consiste semplicemente nel fissare S_i al più promettente valore del dominio di S_{ip}, dove p è la posizione in cui i è stato inserito; S_i viene quindi fissato al valore j che minimizza d_{ij}.

Dato che trovare il cammino ottimo di costo ridotto negativo non è importante in una struttura di generazione di colonne, il problema è trattato come un CSP e solo le prime k soluzioni vengono incluse nell'insieme R' delle possibili rotte.

Limiti Inferiori per il Sottoproblema

Per potare efficientemente l'albero di ricerca occorre essere in grado di calcolare limiti inferiori ad ogni nodo. Sfortunatamente, anche se la letteratura è prolifica in termini di limiti inferiori per il TSP, non ne esiste alcuno per il PTP. Tuttavia, è abbastanza semplice trasformare un PTP in un TSP asimmetrico

[25], cioè in un ATSP, aggiungendo N nodi e $2N$ archi (si noti che in questo nuovo grafo $c_{ij} = d_{ij}$). Questa porzione extra del grafo costituisce un cammino fittizio che permette la visita di nodi lasciati non visitati dalla soluzione del PTP. Il costo di visita di un nodo attraverso tale cammino fittizio è settato al costo del suo valore duale associato. Il valore obiettivo della soluzione ottima risultante del ATSP sarà superiore al valore della soluzione ottima al PTP per una costante pari alla somma di tutti i valori duali ($\sum_{i \in \{1,\ldots,n\}} \lambda_i$).

Ben noti limiti inferiori del ATSP possono allora essere applicati al grafo trasformato una volta rilassati i vincoli relativi alle risorse. Una soluzione ottima al problema di assegnamento (*Assignment Problem*, AP) è un limite inferiore per il ATSP dato che esso è ottenuto rilassando il vincolo `NoSubTour`. Efficienti metodi primale-duale (come l'algoritmo Ungherese) possono essere usati per risolvere il problema AP e fornire costi ridotti, con i quali può essere eseguito qualche ulteriore filtraggio di domini. Come descritto in [43], il costo ridotto $c^{'}$ può essere interpretato come il costo addizionale che incorre se un arco nuovo viene introdotto nella soluzione. Gli archi (i, j) il cui costo ridotto aggiunto al limite inferiore supera il corrente limite superiore ($LB + c^{'}_{ij} > UB$) possono così essere eliminati dinamicamente durante la ricerca.

Branch-and-Price per Risolvere il Problema

La soluzione ottima al problema principale è ottenuta una volta provato che non esiste alcun cammino di costo ridotto negativo. Sfortunatamente, questa soluzione non è sempre intera e si rende così necessario uno schema di ramificazione, detto branch-and-price (un metodo consistente nel "ramificare" e nel "valutare"), per chiudere lo scarto di interezza tra soluzione frazionaria del problema principale e la soluzione intera del VRPTW.

Non è utile ramificare sulle variabili del problema principale dato che queste variabili non possono essere forzate a prendere il valore 0. Anche fissando x_r a 0 si potrebbe non prevenire in modo efficiente che l'algoritmo di CP generi ancora la stessa rotta r e la aggiunga all'insieme $R^{'}$. Si scelgono quindi come variabili di ramificazione le variabili B, variabili successore simili a quelle usate per descrivere il sottoproblema. Sia f^r_{ij} un valore booleano che indica se j è il successore di i nella rotta r (vale 1 se questo è vero, 0 altrimenti) e f_{ij} il flusso che attraversa l'arco (i, j).

La strategia di ramificazione è la seguente:

1. *Nodo 0.* Si itera tra il problema principale ed il sottoproblema fino a quando non esistono più cammini di costo ridotto negativo.

2. *Limitazione superiore.* Se la corrente soluzione del programma principale è intera ed il suo valore è migliore della miglior soluzione trovata, allora si aggiorna il limite superiore, si memorizza la soluzione corrente e si torna indietro.

3. *Ramificazione.* Una volta trovata la soluzione ottima del problema principale, si identifica la variabile maggiormente frazionaria dato che deve essere fissata la successiva variabile di ramificazione (B). Per fare questo, prima si calcola il flusso che attraversa ogni arco, $f_{ij} = \sum_{r \in R'} f_{ij}^r x_r$, e poi si conta per ogni cliente i il numero o_i di archi uscenti di flusso positivo. Infine, si seleziona per la ramificazione la variabile B_i che è associata con il massimo valore di o_i e si ramifica sul valore j che massimizza f_{ij}.

4. *Esplorazione del problema principale.* Nel caso di arresto nel criterio di selezione di un valore deve essere impiegato un arresto della strategia. Dato che, nel caso presente, il problema principale può essere risolto in modo davvero efficiente, esso può essere usato per stimare l'impatto di decisioni di ramificazione. Viene così temporaneamente imposto il risultato di selezionare ogni valore sul problema principale, che viene poi risolto. La strategia di ramificazione seleziona poi il valore che ha generato il valore meno basso del problema principale.

5. *Filtraggio.* Una volta eseguita una decisione di ramificazione, è importante applicarla in tutto il resto dell'algoritmo. La prima misura da prendere è quella di prevenire la selezione di ogni colonna che violi decisioni precedentemente prese. Per fare questo, si fissa la seguente variabile nel modello di Set Covering:

$$x_r = 0 \quad \forall\, r \in R^{'}, \forall\, i \in N, \forall\, j \notin B_i : f_{ij}^r = 1.$$

   È anche cruciale assicurare che le decisioni di ramificazione siano prese in considerazione al livello del sottoproblema. Si aggiungono quindi i seguenti vincoli al modello del sottoproblema:

$$j \notin B_i \Rightarrow j \notin S_i \quad \forall\, i, j \in \{1, \ldots, N\}.$$

6. *Limitazione inferiore.* Proprio come al passo 1, si itera tra il problema principale ed il sottoproblema fino a quando non esistono più cammini di costo ridotto negativo, prendendo in considerazione i nuovi vincoli di filtraggio. Se il limite inferiore ottenuto è più grande del limite superiore, allora si torna indietro e si cancella la decisione di ramificazione precedentemente presa. Altrimenti, si torna al passo 2.

L'approccio ibrido al VRPTW proposto da Gendreau et al. [39] è stato in grado di risolvere molti problemi di taglia piccola (intorno ai 25 clienti) ed alcuni di taglia medio-grande (50-100 clienti).

Conclusioni

I metodi esatti di OR presentati nel capitolo precedente si sono rivelati, in generale, abbastanza efficienti nel trattare il VRPTW, come pure la grande varietà di problemi ad esso collegati. Alcuni dei metodi e degli algoritmi presentati sono stati applicati con successo nella pratica. Tali metodi esatti hanno contato sullo sfruttamento intelligente di strutture speciali del problema ed hanno sicuramente beneficiato dei costanti avanzamenti nella tecnologia di calcolo.

Tutti i risultati computazionali degli approcci presentati sono stati ottenuti dagli autori con riferimento ad un insieme di problemi di prova sviluppato da Solomon [91]. Solomon presenta due insiemi di problemi: l'insieme 1 che ammette approssimativamente da 5 a 10 clienti per rotta (a causa di vincoli di tempo e di capacità) e l'insieme 2 che ammette anche più di 30 clienti per rotta. L'insieme 2 non è ancora stato considerato da metodi esatti di ottimizzazione perché ritenuto troppo difficile e quindi i problemi di prova fino ad ora sono stati presi solamente dall'insieme 1. Tale insieme è suddidiviso in tre diversi insiemi di problemi che si distinguono per la caratterizzazione dei dati geografici dei problemi: nell'insieme di problemi R1 i dati geografici vengono generati in modo casuale, nell'insieme C1 i dati geografici si presentano divisi in gruppi e nell'insieme di problemi RC1 alcuni dati vengono generati casualmente ed altri sono raggruppati. I problemi in C1 sono 27, quelli in R1 36 e quelli in RC1 24; in totale quindi, nell'insieme di problemi 1 ci sono 87 problemi. Il cliente da servire è identico per tutti i problemi all'interno della stessa tipologia (R, C, o RC). I problemi differiscono tra loro per la larghezza delle finestre temporali: alcuni problemi presentano delle finestre temporali veramente strette, mentre altri hanno delle finestre talmente grandi da essere a malapena vincolanti. Il calcolo del costo e del tempo di viaggio non è stato specificato da Solomon ed ogni ricercatore ha usato diverse convenzioni. Questo fatto complica il confronto tra diversi approcci a causa delle diverse convenzioni usate. Tutti i

problemi di prova sono problemi euclidei con 100 clienti, considerati problemi di dimensione grande. Tuttavia, vengono generati anche problemi con 25 o 50 clienti, prendendo i primi 25 o i primi 50 clienti dell'insieme di 100 clienti; tali problemi sono considerati rispettivamente di dimensione piccola e media.

L'unico approccio di programmazione dinamica al VRPTW, cioè quello di Kolen et al. [71], parzialmente ispirato ad un lavoro precedente sul VRP [22], è stato il primo approccio esatto al VRPTW nel caso di deposito singolo e flotta omogenea e risale ormai al 1987. Il metodo non si è rivelato particolarmente efficiente, riportando soluzioni ottime per pochi problemi fino a 15 clienti dall'insieme di dati di Solomon. Le limitazioni di questo approccio sono essenzialmente dovute alla dimensione dello spazio degli stati, anche se questa è stata ridotta dall'uso di una procedura di rilassamento dello spazio stesso. Inoltre, questo metodo non può facilmente incorporare nuovi vincoli (molteplicità di veicoli o di depositi), se non usando uno spazio degli stati allargato.

Per quanto riguarda gli approcci basati sul rilassamento lagrangiano, in genere essi sono stati usati per il VRPTW allo scopo di minimizzare la distanza totale percorsa per una flotta di dimensione fissata. Usando una struttura di K-tree, Fisher et al. [38] sono riusciti a risolvere 12 problemi in C1, di cui 5 problemi con 25 clienti, 5 con 50 e 2 con 100. Nella maggior parte dei problemi in C1 lo scarto di interezza è piuttosto piccolo, mentre nei problemi R1 esso è grande e solo i problemi più piccoli sono stati risolti. Nessun problema è invece stato risolto in RC1. Tale metodo non si dimostra competitivo con quello di variable splitting proposto dagli stessi autori, con il quale sono riusciti a risolvere 21 problemi, di cui 15 problemi in C1 (6 con 25 clienti,5 con 50 e 4 con 100) e 6 in R1 (5 con 25 clienti e 1 con 50). Lo stesso approccio di variable splitting è stato messo a punto anche da Halse [57] con risultati migliori, con un totale di 33 problemi risolti, di cui 20 in C1, 10 in R1 e 3 in RC1.

Un altro approccio basato sul rilassamento lagrangiano è quello di Kohl e Madsen [70], i quali nel 1997 propongono un algoritmo che sfrutta il rilassamento lagrangiano dei vincoli di assegnamento. Tale approccio, come quelli di variable splitting e decomposizione di Dantzig-Wolfe, divide il problema in un problema principale (trovare i moltiplicatori di Lagrange ottimi) ed in un sottoproblema (un ESPPTWCC). Tuttavia, tale approccio crea istanze relativamente facili del sottoproblema in confronto alla decomposizione di Dantzig-Wolfe e si rivela molto più semplice dell'approccio di variable splitting, dato

che il numero di moltiplicatori di Lagrange è molto più piccolo. Il metodo è stato testato solo sui 27 problemi dell'insieme C1, dato che tali problemi sono noti richiedere ramificazioni davvero piccole ed il lavoro presentato è incentrato sul calcolo di limiti inferiori e non contiene alcun nuovo risultato su una strategia di ramificazione. Tutti i problemi sono stati risolti all'ottimalità e 25 su 27 sono stati risolti senza ramificare. Tale metodo ha permesso di risolvere problemi fino a 100 clienti (9 con 25, 9 con 50 e 9 con 100), compresi problemi fino a quel momento irrisolti, mostrandosi quindi molto competitivo con altri approcci presenti in letteratura. Il tempo di calcolo in genere è di 8 volte inferiore a quello nell'approccio di variable splitting di Halse [57] (che è migliore solo per sette istanze tra le più facili).

Al momento, gli algoritmi ottimi che usano la ramificazione e i tagli sulle soluzioni ottenute attraverso la decomposizione di Dantzig-Wolfe sono i migliori nel campo di ricerca sul VRPTW. L'approccio di Desrosiers et al. [26] presentato nel 1992 trova soluzioni ottime per 29 problemi con 25 clienti, 14 con 50 e 7 con 100 clienti: un totale di 50 problemi, di cui 21 in C1 (9 con 25 clienti, 7 con 50 e 5 con 100 clienti), 21 in R1 (12 con 25 clienti, 7 con 50 e 2 con 100) e 8 in RC1 (tutti con 25 clienti).

I risultati computazionali rivelano anche che il rilassamento lineare del modello di tipo set covering fornisce un eccellente limite inferiore primale che a sua volta permette la derivazione efficiente di una soluzione ottima mediante il branch-and-bound. Per 27 problemi su 87 tentati, il limite inferiore è uguale al valore ottimo; per gli altri problemi, lo scarto di interezza medio tra la soluzione del rilassamento del problema di set covering e la soluzione ottima del VRPTW è 1.5 %.

In termini di tempo di calcolo, il metodo di generazione di colonne è molto più efficiente per problemi con le finestre più vincolanti. In questi casi, i sottoproblemi che generano rotte realizzabili sono facili da risolvere con la PD dato che gli insiemi di stati realizzabili sono relativamente piccoli. Inoltre, anche in questi casi il rilassamento lineare fornisce limiti eccellenti (al contrario, i limiti prodotti da altri rilassamenti generalmente deteriorano se le finestre temporali diventano sempre più strette). Infine, il numero di colonne generato è assai grande dato che ogni volta che il sottoproblema è risolto vengono introdotte nel modello molte colonne di costo marginale negativo. Nonostante ciò, tale procedura di generazione di colonne riduce il tempo totale di CPU rispetto agli approcci basati sul rilassamento lagrangiano.

L'algoritmo ottimo di Soumis et al. [30] sviluppato nel 1999 è stato invece in grado di risolvere all'ottimalità un numero considerevole (69 su 87) di problemi di Solomon coinvolgenti fino a 100 clienti, alcuni fino a quel momento irrisolti, ed un problema con 150 clienti: in totale 70 problemi, di cui 27 in C1 (9 con 25 clienti, 9 con 50 e 9 con 100), 24 in R1 (12 con 25 clienti, 9 con 50 e 3 con 100) e 18 in RC1 (8 con 25 clienti, 8 con 50 e 2 con 100), oltre a quello con 150 clienti. Tale algoritmo si è dimostrato più veloce di altri precedentemente considerati in letteratura (10 volte più veloce di quelli presentati in [26] e [57]) ed i risultati computazionali mostrano l'efficacia delle disuguaglianze valide sviluppate nel rafforzare i limiti inferiori per il VRPTW.

Bisogna notare che gli algoritmi di generazione di colonne e gli approcci di variable splitting hanno esibito risultati computazionali assai simili. Il primo metodo però si è dimostrato lievemente superiore nella qualità delle soluzioni rispetto al secondo, riuscendo a risolvere diversi problemi in più rispetto all'altro. In aggiunta, il metodo di generazione di colonne si è rivelato molto più efficiente in termini di tempo di calcolo.

Ci sono diverse ragioni che spiegano la performance dell'approccio di generazione di colonne rispetto a quella dei diversi approcci basati sul rilassamento lagrangiano per il VRPTW. In primo luogo, l'algoritmo del simplesso usato per risolvere il problema principale nella decomposizione di Dantzig-Wolfe fa uso di una grande quantità di informazioni (per esempio, tutte le colonne aggiunte) ed aggiusta rapidamente i moltiplicatori, mentre l'ottimizzazione del sub-gradiente nel secondo tipo di approccio dipende solo dall'iterazione precedente. In secondo luogo, il metodo di decomposizione di Dantzig-Wolfe è primale e permette l'utilizzo di molte euristiche per una convergenza rapida. Infine, tale metodo fornisce maggiori informazioni per progettare meglio uno schema di branch-and-bound per chiudere lo scarto di interezza tra soluzione frazionaria e intera. Per esempio, decisioni di ramificazione prese su variabili di flusso, su variabili temporali o su vincoli globali sono state progettate ed implementate nell'approccio di decomposizione di Dantzig-Wolfe. La maggior parte di queste decisioni di ramificazione possono essere trasferite ad approcci lagrangiani, migliorando così la loro performance. Questa è proprio la direzione presa da Kohl e Madsen [70] che riportano risultati davvero promettenti con un approccio di shortest path, riuscendo a risolvere problemi di dimensione grande ed anche a ridurre drasticamente il tempo di CPU richiesto per risolvere un certo numero di problemi.

Una proprietà empiricamente osservata nei metodi di generazione di colonne è quella che il valore della soluzione intera del problema di set partitioning risultante dall'uso della decomposizione di Dantzig-Wolfe è molto vicino al suo rilassamento lineare. Bramel e Simchi-Levi [14] hanno mostrato che lo scarto presente tra soluzioni intera e frazionaria diventa arbitrariamente piccolo tendendo a zero con l'aumentare del numero di clienti. Questo rende il processo di branch-and-bound più efficiente.

Riassumiamo in una tabella i risultati computazionali degli approcci esatti di OR al VRPTW analizzati in questo lavoro, ad esclusione dei due approcci che si sono rivelati piuttosto scarsi, cioè quello di PD [71] ed uno di variable splitting [83]. Per ogni approccio riportiamo il numero di istanze risolte di varia dimensione (con 25, 50 e 100 clienti) ed il numero totale di problemi risolti dall'insieme di Solomon. Si noti che per l'approccio di Halse [57] non siamo stati in grado di reperire i dati esatti sulle istanze risolte (nelle relative colonne appare quindi un " - "), ma solo il numero totale di problemi risolti. Infine, il valore 1 aggiunto nella penultima colonna dell'approccio di Soumis et al. [30] rappresenta un problema risolto con 150 clienti.

APPROCCIO	25 CL.	50 CL.	100 CL.	TOTALE
Fisher et al. (K-Tree) [38]	5	5	2	12
Fisher et al. (V.S) [38]	11	6	4	21
Kohl & Madsen [70]	9	9	9	27
Halse [57]	-	-	-	33
Desrosiers et al. [26]	29	14	7	50
Soumis et al. [30]	29	26	14 + 1	70

Tra gli approcci esatti di OR al VRPTW presentati nel capitolo precedente, quindi, uno dei migliori è certamente quello basato sulla generazione di colonne proposto da Desrosiers et al. [26]. Appare naturale usare tale metodo per il confronto con l'approccio ibrido di OR e CP basato sulla generazione di colonne, presentato da Gendreau et al. [39] e descritto nell'ultima sezione del capitolo precedente. Tale scelta è determinata anche dal fatto che i due approcci propongono la stessa decomposizione, cioè la stessa definizione del problema principale e del sottoproblema, del VRPTW. Confrontiamo l'approccio ibrido anche con l'approccio di Soumis et al. [30], il migliore tra tutti gli approcci esatti di OR presentati per quanto riguarda i risultati computazionali.

Anche il metodo ibrido è stato testato sui problemi di Solomon ed è stato in grado di risolvere tutti i problemi di dimensione piccola ed alcuni di quelli di dimensione media e grande. In particolare, il metodo ha risolto all'ottimalità tutti i problemi con 25 clienti, così come in [26] ed in [30], il 55 % dei problemi con 50 clienti, contro il 48 % in [26] ed il 90 % in [30], ed il 28 % dei problemi con 100 clienti, contro il 24 % in [26] ed il 52 % in [30]. Infatti dei 27 problemi in C1 è stato in grado di risolverne 25, di cui 9 con 25 clienti, 9 con 50 e 7 con 100; dei 36 in R1 è stato in grado di risolverne 19, di cui 12 con 25 clienti, 6 con 50 e 1 con 100; dei 24 in RC1 ne ha risolti 9, di cui 8 con 25 clienti ed 1 con 50. In totale, l'approccio ibrido è stato in grado di risolvere 53 problemi dall'insieme di dati di Solomon, contro i 50 risolti in [26] ed i 70 risolti in [30]. Riassumiamo il risultato del confronto fra i tre approcci, comprese le percentuali di risoluzione sulle diverse dimensioni dei problemi, nella tabella seguente:

APPROCCIO	25 CL.	50 CL.	100 CL.	TOTALE
Desrosiers et al. [26]	29 (100%)	14 (48%)	7 (24%)	50
Gendreau et al. [39]	29 (100%)	16 (55%)	8 (28%)	53
Soumis et al. [30]	29 (100%)	26 (90%)	14 + 1 (52%)	70

In termini di risultati computazionali, il metodo ibrido si posiziona quindi tra l'approccio di generazione di colonne di Desrosiers et al. [26] e quello di Soumis et al. [30], il quale resta il migliore in assoluto.

Durante il progetto del metodo ibrido, l'enfasi è stata posta sulla crescente flessibilità piuttosto che sulla riduzione del tempo di calcolo. Il metodo ibrido è infatti più lento dell'approccio puro di OR proposto da Desrosiers et al. [26] e quindi anche di quello presentato da Soumis et al. [30], anche considerando la differenza nella performance dei computer negli anni (tra [26] e [39] intercorrono ben 12 anni). Vale la pena, però, notare che il metodo ibrido è risultato molto più veloce dell'approccio puro di CP al VRPTW (con flotta eterogenea) presentato da Guadin [52] nel 1997.

I risultati, in termini del numero di problemi risolti, ottenuti dal metodo ibrido sono confrontabili con quelli forniti in letteratura da simili algoritmi di OR, ma il paradigma della CP produce un approccio certamente più flessibile. Per esempio, i vincoli di precedenza o ogni tipo di vincoli logici sui clienti o sui veicoli possono essere facilmente supportati nella struttura di CP.

Il fatto che i risultati computazionali dell'approccio ibrido al VRPTW rispetto agli approcci puri di OR non siano particolarmente incoraggianti, spie-

ga il motivo per cui l'approccio ibrido studiato è l'unico presente in letteratura per il VRPTW. Più in generale, il fatto che le tecniche di CP siano più lente per quanto riguarda i tempi di calcolo rispetto a quelle di OR, spiega lo scarso sviluppo di approcci puri o ibridi di CP per problemi di Ottimizzazione Combinatoria molto difficili, al pari del problema VRPTW studiato in questo lavoro.

Per altri tipi problemi più facili, invece, gli approcci ibridi o puri di CP si sono rivelati molto efficaci, riuscendo a competere con i risultati ottenuti per gli stessi problemi da approcci puri di OR. Questo spiega, ad esempio, i diversi metodi risolutivi puri ed ibridi di CP proposti in letteratura negli ultimi anni per il TSPTW (come, ad esempio, [47] e [42]), un difficile problema di Ottimizzazione Combinatoria strettamente correlato al VRPTW, ma un pò più facile da affrontare e risolvere.

Nel vasto campo dei problemi di Ottimizzazione Combinatoria, la ricerca sta sviluppando quindi metodi di risoluzione differenti, puri o ibridi sia di OR che di CP, ognuno dei quali dotato di particolari caratteristiche, vantaggi e svantaggi, cercando di individuare i campi di applicazione più adatti per ciascuno di essi.

La ricerca sull'integrazione di OR e CP ha ottenuto fino ad ora importanti risultati che hanno certamente messo in luce la caratteristica più importante del paradigma della CP, vale a dire la flessibilità. Insieme alla sua particolare espressività, la flessibilità rappresenta uno dei maggiori punti di forza della CP. La CP permette infatti di descrivere in modo naturale problemi anche molto complessi e di trattare abilmente vincoli molto complicati che solitamente si incontrano in problemi reali. Questo aspetto rende certamente la CP un utile strumento che può essere utilizzato anche da utenti che non siano specialisti della metodologia dell'ottimizzazione. Si pensi, ad esempio, alla facilità di aggiunta di vincoli nel paradigma della CP, in contrapposizione alla difficoltà di modifica di un programma in OR. Tale flessibilità è per lo più una conseguenza del modo di CP di mantenere separate la specificazione del modello e le componenti algoritmiche.

Notevoli sono stati gli sforzi fatti in questi anni nella comunità di OR per raggiungere simili risultati in un ambiente tradizionale di Programmazione Matematica e lo sviluppo di linguaggi e sistemi di modelling (come GAMS, AMPL e ABACUS) può essere interpretato come un tentativo di competere con la CP su queste questioni. La maggior parte degli stessi ricercatori di

OR, tuttavia, è convinta del fatto che per quanto riguarda l'espressività e la flessibilità risulti vincitrice la CP sulla OR.

Molto spesso, però, tale flessibilità non è accompagnata da risultati computazionali pienamente soddisfacenti. Per questo motivo non si può dire che la OR possa o debba cedere il posto a questo paradigma emergente. Le tecniche di OR restano sempre tecniche molto efficienti ed indispensabili per la risoluzione di molti problemi difficili di Ottimizzazione Combinatoria. Per questo motivo, si tende a considerare la CP non come il rimpiazzo della ormai "obsoleta" OR, ma piuttosto come un insieme di nuove ed efficaci tecniche di risoluzione che possono essere aggiunte agli strumenti già disponibili per i ricercatori di OR e quindi tecniche di cui la OR può solo beneficiare. Anzi, il punto di forza della OR è proprio il suo appoggio su un insieme diversificato di modelli e tecniche che possono essere selettivamente applicati, da soli o in combinazione, in varie strutture e vari ambienti.

Bibliografia

[1] E.H.L. Aarts, W.P.M. Nuijten.
A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling.
European Journal of Operational Research, 1995.

[2] Adler, A. Ulkucu.
On the Number of Iterations in Dantzig-Wolfe Decomposition.
Decomposition of Large Scale Problems, 181–187,
Himmelblau, editor, 1973.

[3] A. Aggoun, N. Beldiceanu.
Extending CHIP in order to Solve Complex Scheduling and Placement Problems.
Mathematical and Computer Modelling 17 (7), 57–73, 1993.

[4] P. Baptiste, C. Le Pape, W. Nuijten.
Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling.
1st Joint Workshop on Artificial Intelligence and Operational Research, 1995.

[5] R. Barták.
On-Line Guide to Constraint Programming.
Prague, 1998,
http://kti.mff.cuni.cz/ bartak/constraints/

[6] C. Berge.
Graphes et Hypergraphes.
Dunod, Paris, 1970.

[7] H. Beringer, B. De Backer.
Combinatorial Problem Solving in Constraint Logic Programming with Cooperating Solvers.
C. Beierle, L. Plümer, editors,
Logic Programming: Formal Methods and Practical Applications,
Studies in Computer Science and Artificial Intelligence 8, 245–272, Elsevier, 1995.

[8] C. Bèssiere, E. Freuder, J.C. Régin.
Using Constraint Meta-Knowledge to Reduce Arc-Consistency Computation.
Artificial Intelligence 107, 125–148, 1999.

[9] C. Bèssiere, E. Freuder, J. Larrosa, P. Meseguer.
On Forward Checking for Non Binary Constraint Satisfaction.
Proceedings of CP'99, 1999.

[10] C. Bèssiere, J.C. Régin.
Arc Consistency for General Constraint Networks: Preliminary Results.
Proceedings of IJCAI'97, 1997.

[11] C. Bèssiere, J.C. Régin.
Enforcing Arc Consistency for Global Constraints by Solving Subproblems on the Fly.
Proceedings of CP'99, 1999.

[12] A. Bockmayr, T. Kasper.
Branch-and-Infer: A Unifying Framework for Integer and Finite Domain Constraint Programming.
INFORMS Journal on Computing 10 (3), 287–300, 1998.

[13] S.C. Brailford, P.M. Hubbard, B.M. Smith, H.P. Williams.
The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared.
Constraints 1, 119–138, 1996.

[14] J. Bramel, D. Simchi-Levi.
On the Effectiveness of Set Partitioning Formulations for the Vehicle Routing Problem.
Graduate School of Business, Columbia University.

[15] J.P. Carillon, P. Van Hentenryck.
Generality Versus Specificity: An Experience with AI and OR Techniques.
Proceedings of 7th National Conference on Artificial Intelligence (AAAI88), 660–664, 1998.

[16] M. Carlsson, G. Ottosson.
Using Global Constraints for Frequency Allocation.
Astec Technical Report, Uppsala University,
Computing Science Department, 1996.

[17] M. Carlsson, G. Ottosson.
Anytime Frequency Allocation with Soft Constraints.
CP96 Pre-Conference Workshop on Applications, 1996.

[18] M. Carlsson, G. Ottosson.
A Comparison of CP, IP and Hybrids for Configuration Problems.
Technical Report, Swedish Institute of Computer Science, 1999

[19] Y. Caseau, F. Laburthe.
Improved CLP Scheduling with Task Intervals.
Proceedings of 11th International Conference on Logic Programming,
MIT Press, 1994.

[20] Y. Caseau, F. Laburthe.
Solving Various Weighted Matching Problems with Constraints.
Principles and Practice of Constraint Programming, vol. 1330,
Lecture Notes in Computer Science, 17–31, 1997.

[21] Y. Caseau, F. Laburthe.
Solving Small TSPs with Constraints.
Proceedings of the 14th International Conference on Logic Programming,
316–330, Cambridge, MIT Press, 1997.

[22] N. Christofides, A. Mingozzi, P. Toht.
Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxation.
Mathematical Programming 20 (3), 255–282, 1981.

[23] G.B. Dantzig, P. Wolfe.
Decomposition Principle for Linear Programs.
Operations Research 8, 101–111, 1960.

[24] K. Darby-Dowman, J. Little.
Properties of Some Combinatorial Optimization Problems and Their Effect on the Performance of Integer Programming and Constraint Logic Programming.
INFORMS Journal on Computing 10, 276– 286, 1998.

[25] M. Dell'Amico, F. Maffioli, P, Varbrand.
On Prize-Collecting Tours and the Asymmetric Travelling Salesman Problem.
International Transactions in Operational Research 2 (3), 297–308, 1995.

[26] M. Desrochers, J. Desrosiers, M. Solomon.
A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows.
Operations Research 40 (2), 342–354, 1992.

[27] M. Desrochers, J. Desrosiers, F. Soumis.
Routing with Time Windows by Column Generation.
Networks 14, 545–565, 1984.

[28] J. Desrosiers, Y. Dumas, E. Gélinas, M.M. Solomon.
An Optimal Algorithm for the Travelling Salesman Problem with Time Windows.
Operations Research 43, 367–371, 1995.

[29] J. Desrosiers, Y. Dumas, M.M. Solomon, F. Soumis.
Time Constrained and Scheduling.
Network Routing 8, Handbooks in Operations Research and Management Science, 35–139, North–Holland, Amsterdam, 1995.

[30] J. Desrosiers, N. Kohl. O.B.G. Madsen, M.M. Solomon, F. Soumis.
2-Path Cuts for the Vehicle Routing Problem with Time Windows.
Transportation Science 33, 101–116, 1999.

[31] M. Desrosiers, M. Sauvé, F. Soumis.
Lagrangian Relaxation Methods for Solving Minimum Fleet Size Multiple Travelling Salesman Problem with Time Windows.
Management Science 34, 1005–1022, 1988.

[32] Y. Deville, P. van Hentenryck, L. Michel.
Numerica: A Modelling Language for Global Optimization.
MIT Press, Cambridge, Mass., 1997.

[33] Y. Deville, P. van Hentenryck, V. Saraswat.
Design, Implementations and Evaluation of the Constraint Language cc(FD).
Technical Report CS–93–02, Brown University, 1992.

[34] G.L. Elliot, R.M. Haralick.
Increasing Tree Search Efficiency for Constraint Satisfaction Problems.
Artificial Intelligence 14, 263–314, 1980.

[35] T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben.
A Framework for Constraint Programming Based Column Generation.
CP'99 Principles and Practice of Constraint Programming,
Lecture Notes in Computer Science, vol.1713, 261–274, Springer-Verlag, 1999.

[36] M.L. Fisher.
Optimal Solution of Vehicle Routing Problems Using Minimum K-Trees.
Operations Research 42 (4), 626–642, 1994.

[37] M.L. Fisher.
A Polinomial Algorithm for the Degree-Constrained Minimum K-Tree Problem.
Operations Research 42 (4), 775–779, 1994.

[38] M.L. Fisher, K.O. Jörnsten, O.B.G. Madsen.
Vehicle Routing with Time Windows: Two Optimization Algorithms.
Operations Research 45 (3), 488–492, 1997.

[39] F. Focacci, M. Gendreau, G. Pesant, L.M. Rousseau.
Solving VRPTWs with Constraint Programming Based Column Generation.
Annals of Operations Research 130, 199–216, 2004.

[40] F. Focacci, A. Lodi, M. Milano.
Integration of CP and OR Methods for Matching Problems.
CP-AI-OR'00 Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 1999.

[41] F. Focacci, A. Lodi, M. Milano.
Cost-Based Domain Filtering.
Principles and Practice of Constraint Programming,
vol. 1713 of Lecture Notes in Computer Science, Springer, 1999.

[42] F. Focacci, A. Lodi, M. Milano.
Solving TSP with Time Windows with Constraints.
16th International Conference on Logic Programming, 1999.

[43] F. Focacci, A. Lodi, M. Milano, D. Vigo.
Solving TSP through the Integration of OR and CP Techniques.
CP98 Workshop on Large Scale Combinatorial Optimisation and Constraints, 1998.

[44] F. Focacci, M. Milano.
Connections And Integration of Dynamic Programming and Constraint Programming.
Proceedings of 3rd Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Ashford, Kent, UK, 125–138, 2001.

[45] L.R. Ford, D.R. Fulkerson.
A Suggested Computation for Maximal Multicommodity Network Flows.
Management Science 5, 97–101, 1958.

[46] J. Gaschnig.
Performance Measurement and Analysis of Certain Search Algorithms.
CMU–CS–79–124, Carnegie-Mellon University, 1979.

[47] M. Gendreau, G. Pesant, J.Y. Potvin, J.M. Rousseau.
An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows.
Transportation Science 32 (1), 1998.

[48] A.M. Geoffrion.
Lagrangean Relaxation and Its Uses in Linear Programming.
Mathematical Programming Study 2, 82–114, 1974.

[49] P.C. Gilmore, R.E. Gomory.
A Linear Programming Approach to the Cutting Stock Problem.
Operations Research 9, 849–859, 1961.

[50] R.E. Gomory.
Outline of an Algorithm for Integer Solution to Linear Programs.
Bulletin Amer. Math. Soc. 64, 5, 1958.

[51] I.E. Grossmann, V. Jain.
Algorithms for Hybrid MILP/CLP Models for a Class of Optimization Problems.
Technical Report, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, 1999.

[52] E. Guadin.
Contribution de la Programmation par Contraintes au Trasport: Définition et Résolution d'un Modèle Complexe de Gestion de Flotte.
PhD Thesis, Université Pris 7, 1997.

[53] M. Guignard, S. Kim.
Lagrangean Decomposition: A Model Yielding Stronger Lagrangean Bounds.
Mathematical Programming 39, 215–228, 1987.

[54] M. Hajian.
Dis-equality Constraints in Linear/Integer Programming.
Technical Report, IC-Parc, 1996.

[55] M. Hajian, B. Richards, R. Rodošec.
Introduction of a New Class of Variables to Discrete and Integer Programming Problems.
Baltzer Journals, 1996.

[56] M. Hajian, R. Rodošec, M. Wallace.
A New Approach to Integrating Mixed Integer Programming and Constraint Logic Programing.
Baltzer Journals, 1997.

[57] K. Halse.
Modelling and Solving Complex Vehicle Routing Problems.
PhD. Dissertion no. 60, Institute of Mathematical Statistics and Operations Research, Technical University of Denmark, DK-2800 Lyngby, Denmark, 1992.

[58] S. Heipcke.
Integrating Constraint Programming Techniques into Mathematical Programming.
Proceedings of 13th European Conference on Artificial Intelligence (ECAI-98), Henri Prade, editor, John Wiley & Sons, 1998.

[59] S. Heipcke.
Comparing Constraint Programming and Mathematical Programming Approaches to Discrete Optimization: The Change Problem.
Annals of Operations Research 50, 581–595, 1999.

[60] J.N. Hooker, H. Kim, G. Ottosson, E.S. Thorsteinsson.
On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimisation.
Proceedings of 16th National Conference on Artificial Intelligence (AAAI-99), MIT Press, 136–141, 1999.

[61] J.N. Hooker, H, Kim, G. Ottosson, E.S. Thorsteinsson.
A Scheme for Unifying Optimization and Constraint Programming.
Knowledge Engineering Review, 1999.

[62] J.N. Hooker, G. Ottosson, E.S. Thorsteinsson.
Mixed Global Constraints and Inference in Hybrid CLP-IP Solvers.
Proceedings 5th International Conference on Principles and
Practice of Constraint Programing (CP-99), 1999.

[63] E. Jacquet-Lagreze.
Hybrid Methods for Large Scale Optimization Problems:
An OR Perspective.
Proceedings of Practical Application of Constraint Technology
(PACT98), London, UK, 1998.

[64] J. Jaffar, J.L. Lassez.
Constraint Logic Programming.
Proceedings of the Conference on Principle of Programming Language,
1997.

[65] M.D. Johnston, P. Laird, S. Minton.
Minimising Conflicts: A Heuristic Repair Method for
Constraint Satisfaction and Scheduling Problems.
Artificial Intelligence 58 (1–3), 161–206, 1992.

[66] K. Jörnsten, O.G.B. Madsen, B. Sørensen.
Exact Solution of the Vehicle Routing and Scheduling Problem
with Time Windows by Variable Splitting.
Research Report 5/1986, Institute of Mathematical Statistics and
Operations Research, Technical University of Denmark,
DK-2800 Lyngby, Denmark, 1986.

[67] K. Jörnsten, M. Näsberg, P. Smeds.
Variable Splitting - A New Approach to Some Mathematical Programming
Models.
Report LITH-MAT-85-04, Department of Mathematics,
Linköping Institute of Technology, Sweden, 1985.

[68] H. Kautz, B. Selman.
Domain-Independent Exstensions to GSAT:
Solving Large Structured Satisfiability Problems.
IJCAI–93, 1993.

[69] N. Kohl.
Exact Methods for Time Constrained Routing and Related Scheduling Problems.
PhD Thesis, IMM–PHD–1995–16, Department of Mathematical Modelling, Technical University of Denmark, 1995.

[70] N. Kohl, O.G.B. Madsen.
An Optimization Algorithm for the Vehivle Routing Problem with Time Windows Based on Lagrangean Relaxation.
Operations Research 45 (3), 395–406, 1997.

[71] A.W. Kolen, A.H.G. Rinnoy Kaan, H.W.J.M. Trienekens.
Vehicle Routing with Time Windows.
Operations Research 35 (2), 266–273, 1987.

[72] V. Kumar.
Algorithms for Constraint Satisfaction Problems: A Survey.
AI Magazine 13 (1), 32–44, 1992.

[73] E.W. Lawler, D.E. Wood.
Branch-and-Bound Methods: A Survey.
Operations Research 14, 699–719, 1996.

[74] J. W. Lloyd.
Foundations of Logic Programming–Second Extended Edition.
Springer–Verlag, 1987.

[75] A.K. Mackworth.
Consistency in Networks of Relations.
Artificial Intelligence 8 (1), 99–118, 1977.

[76] O.G.B. Madsen.
Variable Splitting and Vehicle Routing Problems with Time Windows.
Technical Report 1A/1988, Department of Mathematical Modelling, Technical University of Denmark, 1988.

[77] K. Marriot, P.J. Stuckey.
Programming with Constraints: An Introduction.
MIT Press, 1998.

[78] E. Monfroy.
Solver Collaboration for Constraint Logic Programming.
PhD Thesis, Université Henri Poincaré-Nancy I, 1996.

[79] U. Montanary.
Networks of Constraints Fundamental Properties and Applications to Pictur Processing.
Information Sciences 7, 95–132, 1974.

[80] J.L. Nazareth.
Computer Solution of Linear Programs.
Oxford University Press, Oxford, 1987.

[81] G.L. Nemhauser, R.A. Wolsey.
Integer and Combinatorial Optimization.
Wiley-Interscience, New York, USA, 1988.

[82] N.J. Nilsson.
Principles of Artificial Intelligence.
Tioga, Palo Alto, 1980.

[83] B. Olsen.
Routing with Time Constraints: Exact Solution.
M.Sc. Thesis (in Danish), Institute of Mathematical Statistics and Operations Research, Technical University of Denmark, DK-2800 Lyngby, Denmark, 1988.

[84] G. Ottosson, E.S. Thorsteinsson.
Linear Relaxations and Reduce-Cost Based Propagation of Continuos Variable Subscripts.
Annals of Operations Research, 2000.

[85] L. Proll, B. Smith.
Integer Linear Programming and Constraint Programming Approaches to a Template Design Problems.
INFORMS Journal on Computing 10, 265–275, 1998.

[86] P. Refalo.
Tight Cooperation and its Application in Piecewise Linear Optimization.
J. Jaffar, editor, Principles and Practice of Constraint Programming, vol. 1713, Lecture Notes in Computer Science, Springer, 1999.

[87] J.C. Régin.
A Filtering Algorithm for Constraints of Difference in CSPs.
Proceedings of 12th National Conference on Artificial Intelligence (AAAI-94), 362–367, 1994.

[88] M.W.P. Savelsbergh.
Local Search in Routing Problems with Time Windows.
Annals of Operations Research 4, 285–305, 1985.

[89] M.W.P. Savelsbergh.
Preprocessing and Probing for Mixed Integer Programming Problems.
ORSA Journal on Computing 6, 445–454, 1994.

[90] A. Schrijver.
Theory of Linear Programming.
John Wiley & Sons, Chichester, 1986.

[91] M.M. Solomon.
Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints.
Operations Research 35 (2), 254–265, 1987.

[92] E. Tsang.
Foundations of Constraint Satisfaction.
Academic Press, London, 1995.

[93] P. Van Hentenryck.
A Logic Language for Combinatorial Optimisation.
Annals of Operations Research, 1989.

[94] J. Zowe.
Nondifferentiable Optimization.
Computational Mathematical Programming vol. F15, Springer-Verlag, Berlin, Germany, 1985.