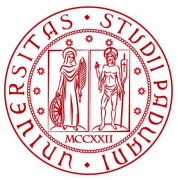

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Triennale in Matematica

Modelli di programmazione matematica
per il problema del Green Vehicle Routing

Relatore:
Prof. Luigi De Giovanni

Laureando: Maria Verzotto
Matricola: 1201614

Anno Accademico 2023/2024

19/07/2024

Indice

1 Introduzione **5**

2 Definizione e formulazione del problema **7**
2.1 Notazione 7
2.2 Formulazione di Erdogan e Miller-Hooks 8
2.3 Formulazione senza nodi fittizi 10
2.4 Formulazione senza nodi fittizi con più ricariche consecutive 12
2.5 Formulazione di tipo set partitioning 14

3 Metodi per il GVRP basati sui piani di taglio **17**
3.1 Tagli di Gomory 18
3.2 Esempi di Tagli specifici per problemi VRP 20

4 Metodi per il GVRP basati su Branch and Cut **23**
4.1 Branch and Bound 23
4.2 Branch and cut 26
4.3 Aspetti implementativi 27

5 Metodi per il GVRP basati su Branch and Price **29**
5.1 Metodo di Generazione di Colonne 30
5.2 Algoritmo di etichettatura bidirezionale 31
5.3 Algoritmo Branch and Price 33
5.4 Algoritmo Branch-and-Price-and-Cut 35

6 Valutazione delle performance **37**
6.1 Risultati di algoritmi $B\&C$ 37
6.2 Risultati di algoritmi $B\&P$ e $B\&P\&C$ 38

7 Conclusioni **41**

Capitolo 1

Introduzione

Il Green Vehicle Routing Problem (GVRP), viene presentato per la prima volta da Miller-Hooks et al. in [18]: si tratta di un problema di ottimizzazione che ha cominciato a essere oggetto di studio negli ultimi anni, ovvero da quando si è cominciato a prestare attenzione alle emissioni di CO_2 e all'utilizzo dei veicolo elettrici. Si tratta di un problema NP-hard che può essere inquadrato nella classe più ampia dei Vehicle Routing Problem (VRP).
Il VRP è anch'esso un problema NP-hard presentato per la prima volta da Dantzig et al. [10] che richiede l'identificazione di un multi insieme di cicli che coprano i nodi di un grafo a costo minimo.
Secondo la descrizione elaborata da Miller-Hooks [18], il GVRP aggiunge la particolarità dei veicoli a carburanti alternativi (*Alternative Fuel Vehicle* - AFV), che pertanto richiedono stazioni apposite per potersi ricaricare (*Alternative Fuel Station* - AFS). Nell'attuale contesto delle reti di trasporto tali stazioni, che chiameremo anche stazioni di ricarica, non sono ancora molto diffuse e pertanto è necessario programmare il tragitto tenendo in considerazione il bisogno di fermarsi presso suddette stazioni quando necessario.
Il GVRP viene quindi definito verbalmente come la ricerca di un itinerario ottimale per un insieme di veicoli che, partenti da un punto comune chiamato deposito, devono visitare e servire un insieme di clienti sparsi lungo il territorio, prestando attenzione alla propria autonomia e fermandosi, se necessario, presso stazioni di ricarica.
Dati come input un grafo pesato che rappresenta l'insieme delle strade, i clienti e le stazioni di ricarica, il problema consiste nel trovare un insieme di cicli non-orientati (uno per ogni veicolo) che compiano un tragitto minimo e che partano e tornino presso il deposito fermandosi se necessario presso le stazioni di ricarica, in modo da non esaurire il carburante e minimizzare una funzione di costo.
L'insieme di tali cammini deve servire tutti i clienti nel grafo esattamente una volta, mentre possono fermarsi presso le stazioni di ricarica quante volte necessario.
Possiamo trovare molte varianti del GVRP, a partire dal *Green Vehicle Routing Problem with Time Windows* (GVRPTW) (Rezaei et al. [39]) , o dal *Time Dipendent Green Vehicle Routing Problem* (Dridi et al. [33], Gao et al. [22]).
Le soluzioni proposte vengono generalmente suddivise in tre categorie: euristiche (Sweda et al. [42]) metaeuristiche (Liu et al. [32]) ed esatte, che studieremo in questa tesi (Desaulniers et al. [13], Dridi et al. [33], Behnkee et al. [4]). Per un'analisi più approfondita sulle varie sotto-categorie del GVRP e delle varie soluzioni fare riferimento a Garside at

al. [23]. In questa tesi passeremo in rassegna alcune soluzioni proposte da altri lavori sviluppati per risolvere questo problema. Nel Capitolo 2 verranno analizzate diverse formulazioni proposte per il GVRP. Nel Capitolo 3 verrà analizzato l'algoritmo dei piani di taglio e riportati alcuni tagli utili per il GVRP.
Nel Capitolo 4 viene illustrato l'algoritmo Branch-and-cut (unione degli algoritmi dei piani di taglio e del Branch-and-Bound) Nel Capitolo 5 viene infine riportato l'algoritmo Branch-and-Price, insieme alla sua implementazione detta Branch-Price-and-Cut. Infine nel Capitolo 6 presentiamo un'analisi su come i vari algoritmi si comportano in una serie di istanze, sulla base dei risultati computazionali presentati in letteratura.

Capitolo 2

Definizione e formulazione del problema

Il GVRP viene comunemente definito attraverso una formulazione di programmazione lineare intera mista (MILP).
Andremo ora a presentare alcuni modelli MILP sviluppati per il GVRP:

- Formulazione con nodi fittizi [18];
- Formulazione senza nodi fittizi [29];
- Formulazione senza nodi fittizi e che permetta più visite consecutive presso AFS [6];
- Formulazione set partitioning.

La prima formulazione risulta essere tra le più comuni, ma è molto pesante da risolvere con risolutori MILP a causa dell'utilizzo di nodi fittizi. Sono state pertanto proposte altre formulazioni per superare questo problema (Koc et al. [29], Bruglieri et al. [6]). Infine presentiamo una formulazione set partitioning, adatta per approcci di soluzioni di tipo Branch-and-Price (come verrà visto nel Capitolo 5).

2.1 Notazione

Secondo quanto stabilito da Miller-Hooks [18], il GVRP è definito su un grafo completo non orientato $G = (V, A)$ dove l'insieme dei vertici $V = I \cup \{v_0\} \cup F$ è formato dall'unione dei vertici rappresentanti i clienti $I = \{v_1, v_2, ..., v_n\}$ con il deposito v_0 e con l'insieme $F = \{v_{n+1}, v_{n+2}, ..., v_{n+s}\}$ delle stazioni di ricarica.
L'insieme $A = \{(v_i, v_j) : v_i, v_j \in V, i < j\}$ contiene gli archi che connettono i vertici di V, e rappresentano le interconnessioni stradali. Ad ogni arco $(v_i, v_j) \in A$ associamo un valore t_{ij} che indica il tempo di viaggio, un costo c_{ij} di carburante e una distanza d_{ij}.
Assumiamo, per compatibilità con alcuni dei lavori passati in rassegna, che la velocità impiegata da un veicolo a percorrere un arco sia sempre costante e abbia valore sp.
Un numero m di veicoli è disponibile al deposito, ponendo quindi un limite superiore

al numero di tour che sarà possibile effettuare contemporaneamente: ciascun veicolo ha capacità massima Q di carburante, con cui indicheremo sia i carburanti alternativi che i livelli delle batterie per le auto elettriche, con un tasso di consumo di questo di valore r. Assumiamo che ciascuno cliente possa essere servito da un veicolo che parta dal deposito e ritorni ad esso all'interno del tempo T_{max}.

Il problema consiste nel determinare una serie di percorsi per i veicoli così da minimizzare il costo totale e che rispettino le seguenti indicazioni:

- Ciascun veicolo può compiere al massimo un percorso;
- Ciascun percorso inizia e termina al deposito;
- Ciascun cliente è servito esattamente una volta;
- Il livello di carburante del veicolo non può essere inferiore alla quantità richiesta per compiere la distanza tra due nodi;
- La durata totale dei percorsi assegnati non può superare un tempo T_{max}.

Definiamo ora la variabile decisionale $x_{ij} \in \{0, 1\}$ come una variabile avente valore 1 se il veicolo viaggia dal nodo i al nodo j, 0 altrimenti.
Utilizzeremo f_i per misurare il carburante rimanente nel veicolo quando questo arriva al nodo i con la condizione che tale valore viene resettato a Q quando visita una stazione di servizio o il deposito.
Introduciamo infine la variabile temporale τ_i per indicare il tempo di arrivo presso il nodo i. Il suo valore è fissato a 0 alla partenza dal deposito.
Una delle differenze tra il GVRP e il VRP è la presenza nel primo delle stazioni di ricarica. Queste rappresentano un insieme a parte di vertici visitabili in quanto, a differenza dei vertici cliente, possono essere visitati più volte da più veicoli diversi. Di seguito andremo a presentare come le diverse formulazioni modellano questa caratteristica.

2.2 Formulazione di Erdogan e Miller-Hooks

La prima formulazione data del GVRP è quella di Erdogan e Miller-Hooks [18], per il cui sviluppo si sono basati sull'opera di Bard et al. [2]. In essa viene formulato un VRP con satelliti, ovvero un MILP con restrizioni sulla durata dei percorsi e sulla capacità dei veicoli: i veicoli considerati hanno la possibilità di fermarsi presso delle stazioni rifornimento per ricaricarsi e soddisfare le richieste dei clienti.

Consideriamo ora il grafo $G = (V, A)$ con le notazioni descritte in precedenza. Per risolvere il problema del riutilizzo dei vertici stazioni di ricarica, viene introdotto un insieme di vertici fittizi come copie di questi.
Consideriamo quindi il grafo $G' = (V', A')$, dove $V' = V \cup \Phi$ e $\Phi := \{v_{n+s+1}, ..., v_{n+s+s'}\}$ è un insieme di s' vertici fittizi in cui ciascun vertice v_{n+s+i}, con $i \in [1, s']$, corrisponde a una potenziale visita ad una AFS o al deposito.
Ad ogni vertice $v_f \in F$ è associato n_f che indica il numero di volte che la stazione di ricarica può essere visitata. Per limitare la complessità del problema, n_f deve essere fissato come il più piccolo possibile, ma abbastanza grande da non limitare i benefici che comporta al problema.

Introduciamo ora altre notazioni per il GVRP:

- I_0 Insieme dei clienti e del deposito $I_0 = \{v_0\} \cup I$;
- F_0 Insieme delle AFS e del deposito $F_0 = \{v_0\} \cup F'$, dove $F' = F \cup \Phi$;
- p_i Tempo trascorso presso il vertice i (se $i \in I$ allora corrisponde al tempo trascorso presso il cliente, se $i \in F$ corrisponde invece al tempo di ricarica presso la AFS).

La formulazione matematica del GVRP risulta quindi essere la seguente:

$$\min \sum_{\substack{i,j \in V' \\ i \neq j}} d_{ij} x_{ij} \tag{2.2.1}$$

s.t.

$$\sum_{\substack{j \in V' \\ j \neq i}} x_{ij} = 1, \qquad \forall i \in I \tag{2.2.2}$$

$$\sum_{\substack{j \in V' \\ j \neq i}} x_{ij} \leq 1, \qquad \forall i \in F_0 \tag{2.2.3}$$

$$\sum_{\substack{i \in V' \\ j \neq i}} x_{ij} - \sum_{\substack{i \in V' \\ j \neq i}} x_{ji} = 0, \qquad \forall j \in V' \tag{2.2.4}$$

$$\sum_{j \in V' \setminus \{0\}} x_{0j} \leq m \tag{2.2.5}$$

$$\sum_{j \in V' \setminus \{0\}} x_{j0} \leq m \tag{2.2.6}$$

$$\tau_j \geq \tau_i + (t_{ij} + p_j) x_{ij} - T_{max}(1 - x_{ij}), \qquad \forall i \in V', \forall j \in V' \setminus \{0\}, i \neq j \tag{2.2.7}$$

$$0 \leq \tau_0 \leq T_{max} \tag{2.2.8}$$

$$t_{0j} \leq \tau_j \leq T_{max} - (t_{j0} + p_j), \qquad \forall j \in V' \setminus \{0\} \tag{2.2.9}$$

$$f_j \leq f_i - r \cdot d_{ij} x_{ij} + Q(1 - x_{ij}), \qquad \forall j \in I, i \neq j \tag{2.2.10}$$

$$f_j = Q, \qquad \forall j \in F_0 \tag{2.2.11}$$

$$f_j \geq min\{r \cdot d_{j0}, r \cdot (d_{jl} + d_{l0})\} \qquad \forall j \in I, \forall l \in F' \tag{2.2.12}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \tag{2.2.13}$$

La funzione obiettivo (2.2.1) minimizza la distanza totale percorsa dai veicoli, mentre i vincoli (2.2.2) - (2.2.4) sono conosciuti come *vincoli di flusso* e hanno il compito di assicurare la continuità del percorso.
Più nello specifico, (2.2.2) assicura che ogni nodo visitato abbia esattamente un successore, mentre (2.2.3) che ogni AFS abbia al massimo un vertice successivo, che sia un cliente, un'altra AFS o il nodo di partenza. (2.2.4) si assicura invece che il flusso sia conservato, controllando che il numero di partenze da un vertice sia uguale a quello degli arrivi.
I vincoli (2.2.5) e (2.2.6) si occupano invece di controllare che il numero di veicoli che partono dal deposito sia uguale al numero di veicoli che tornano, e che tale numero non superi la quantità m di veicoli a disposizione.
I vincoli (2.2.7)-(2.2.9), detti *vincoli temporali*, si assicurano che ogni veicoli sia tornato al deposito entro T_{max}: nello specifico (2.2.7) traccia il tempo di arrivo di ogni veicolo nei vari vertici, (2.2.8) controlla il momento di partenza dal deposito e ne fissa un limite superiore. (2.2.9) invece fissa i limiti inferiori e superiori dei momenti di arrivo presso i vertici clienti e presso le stazioni di ricarica, oltre ad assicurarsi che ogni percorso sia completato entro il tempo massimo T_{max}.
I vincoli (2.2.10)-(2.2.12), detti *vincoli di consumo* , servono appunto a tracciare il consumo di carburante dei veicoli lungo il percorso.
Il vincolo (2.2.10) controlla il carburante tenendo conto della sequenza e del tipo di vertici che visita: infatti se il vertice $j \in V'$ viene visitato subito dopo un vertice $i \in V'$ (quindi $x_{ij} = 1$) e il vertice i è un vertice cliente, allora il livello di carburante all'arrivo presso il nodo j è ridotto in base alla distanza d_{ij} e al tasso di consumo del carburante.
Il vincolo (2.2.11) resetta il livello del carburante a Q quando si visita una stazione di servizio, mentre (2.2.12) controlla che ci sia abbastanza carburante per tornare al deposito direttamente o passando attraverso stazioni di ricarica.
Infine, (2.2.13) garantisce l'integrità binaria della variabile x_{ij}.

2.3 Formulazione senza nodi fittizi

L'aggiunta di nodi fittizi nella formulazione appena descritta comporta il rischio di aumentare in maniera molto pesante il numero di nodi totali del problema: nel peggior caso, si può arrivare ad avere una copia di ogni AFS (in numero di $|F|$) per ogni cliente (in numero di $|V|$) che aumenta il numero totale di nodi da $|F| + |V| + 1$ a $|F||V| + |V| + 1$. Per questo motivo, Koc et al. [29] hanno sviluppato una formulazione che con i giusti accorgimenti permette di evitare e limitare questo problema. La loro proposta corrisponde all'aggiungere ulteriori variabili binarie che suddividono il problema in percorsi di tre nodi che iniziano da un cliente (o dal deposito), raggiungono una stazione di servizio e terminano da un cliente (o nel deposito).
Aggiungiamo pertanto la variabile y_{ikj} descritta come sopra, avente valore 1 se, per $i, j \in I_0$ e $k \in F$, il veicolo viaggia dal punto i alla stazione k per poi proseguire al

punto j, altrimenti 0.
Definiamo anche la distanza e il tempo impiegati per percorrere tale tratto:

$$\widetilde{d}_{ikj} = d_{ik} + d_{kj} \qquad \forall i, j \in I; \quad \forall k \in F_0, \quad i \neq j \tag{2.3.1}$$

$$\widetilde{t}_{ikj} = p_i + p_k + (d_{ikj}/sp) \qquad \forall i, j \in I; \quad \forall k \in F, \quad i \neq j \tag{2.3.2}$$

Fissiamo $y_{0k0} = 0 \quad \forall k \in F$ e $y_{00j} = y_{j00} = 0 \quad \forall j \in V$.

Basandoci su queste definizioni e su quelle illustrate nella Sezione 2.1, il GVRP presentato da [29] è quindi il seguente:

$$\min \sum_{i,j \in I_0} d_{ij} x_{ij} + \sum_{i,j \in I_0} \sum_{k \in (\{0\} \cup F)} \widetilde{d}_{ikj} y_{ikj} \tag{2.3.3}$$

s.t.

$$\sum_{i \in I_0} (x_{ij} + \sum_{k \in (\{0\} \cup F)} y_{ikj}) = 1 \qquad \forall j \in I \tag{2.3.4}$$

$$\sum_{i \in I_0} (x_{ij} - x_{ji}) + \sum_{k \in (\{0\} \cup F)} (y_{ikj} - y_{jki}) = 0 \qquad \forall i \in I_0 \tag{2.3.5}$$

$$\tau_i - \tau_j + (M^1_{ikj} - \widetilde{t}_{ikj}) x_{ij} + (M^1_{ikj} - \widetilde{t}_{ikj} - t_{ij} - t_{ji}) x_{ij} +$$

$$(M^1_{ikj} - t_{ij}) y_{ikj} + (M^1_{ikj} - t_{ij} - \widetilde{t}_{ikj} - \widetilde{t}_{jki}) y_{jki} \leq$$

$$M^1_{ikj} - t_{ij} - \widetilde{t}_{ikj} \qquad \forall i, j \in I; \qquad \forall k \in (\{0\} \cup F) \tag{2.3.6}$$

$$\tau_j \geq t_{0j} x_{0j} + \sum_{k \in F} (\widetilde{t}_{0kj} y_{0kj}) \qquad \forall j \in I \tag{2.3.7}$$

$$\tau_j \leq T_{max} - (T_{max} - t_{0j}) x_{0j} - \sum_{k \in F} (T_{max} - \widetilde{t}_{0kj}) y_{0kj} \qquad \forall j \in I \tag{2.3.8}$$

$$\tau_i \leq T_{max} - t_{i0} x_{i0} - \sum_{k \in F} (\widetilde{t}_{ik0} y_{ik0}) \qquad \forall i \in I \tag{2.3.9}$$

$$f_j - f_i + M^2_{ij} x_{ij} + (M^2_{ij} - r \cdot d_{ij} - r \cdot d_{ji}) x_{ji} \leq M^2_{ij} - r \cdot d_{ij} \qquad \forall i, j \in I; i \neq j \tag{2.3.10}$$

$$f_i \leq Q - r \cdot d_{0j} x_{0j} - \sum_{i \in I_0} \sum_{k \in F} (r \cdot d_{kj} y_{ikj}) \qquad \forall j \in I \tag{2.3.11}$$

$$f \geq r \cdot d_{i0} x_{i0} + \sum_{j \in I_0} \sum_{k \in F} (r \cdot d_{ik} y_{ikj}) \qquad \forall i \in I \tag{2.3.12}$$

$$\tau_i, f_i \geq 0, \qquad \forall i \in I \tag{2.3.13}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in I_0 \tag{2.3.14}$$

$$y_{ikj} \in \{0,1\} \qquad \forall i,j \in I_0; \quad k \in F \tag{2.3.15}$$

Il modello fa uso di costanti M^1 e M^2, definite come segue $\forall i,j \in I$:

$$M^1_{ijk} = T_{max} + t_{ij} + t_{ijk} - t_{i0} - t_{0j} \qquad \forall k \in F \tag{2.3.16}$$

$$M^2_{ij} = Q + r \cdot d_{ij} - \min_{k \in F}\{r \cdot d_{kj}\} - \min_{k \in F} r \cdot d_i k \tag{2.3.17}$$

La funzione obiettivo (2.3.3) minimizza la distanza percorsa dai veicoli. I vincoli (2.3.4) e (2.3.5) sono conosciuti come *vincoli di grado* e si assicurano rispettivamente che ogni cliente sia visitato esattamente una volta e che il numero di archi uscenti ed entranti in un nodo combacino.
Il tempo di arrivo presso un nodo è controllato da (2.3.6) che, insieme a (2.3.7), (2.3.8) e (2.3.9) garantisce che ogni veicoli torni al deposito entro un tempo T_{max}.
Nello specifico, i vincoli (2.3.7) e (2.3.8) limitano la variabile τ_i e specificano il tempo di arrivo di un veicolo presso il primo cliente del percorso. (2.3.9) invece assicura che il tempo di arrivo presso l'ultimo cliente permetta di tornare al deposito prima del tempo massimo T_{max}.
Il vincolo (2.3.10) determina il livello di carburante presso ogni cliente in base alla distanza e il consumo di carburante del veicolo. La variabile (2.3.11) specifica il valore di carburante presso il primo nodo visitato o subito dopo la stazione di servizio, mentre (2.3.12) garantisce che il livello di carburante di un veicolo sia abbastanza per raggiungere una stazione di servizio oppure il deposito. Infine (2.3.13)-(2.3.15) sono vincoli di non negatività e integralità.

2.4 Formulazione senza nodi fittizi con più ricariche consecutive

La formulazione presentata da Koc et al. [29] implica però che si possa visitare solamente una stazione di ricarica nel tragitto tra due clienti: questa è una limitazione abbastanza restrittiva siccome possono esserci casi in cui è impossibile raggiungere due clienti con una sola fermata intermedia presso una stazione di ricarica.
Bruglieri et al. [6] propongono quindi un'altra formulazione, che consenta due visite consecutive presso le stazioni di ricarica.
Partendo dalla formulazione di Koc, applicano alcune modifiche:

- La variabile y_{ikj} viene sostituita dalla variabile binaria $z_{ijs_1s_2}$, uguale a 1 se le stazioni $s_1, s_2 \in F$ vengono visitate consecutivamente nel cammino tra i e j, 0 altrimenti.
- Definiamo le variabili relative a $y_{ijs_1s_2}$

$$\widetilde{t}_{ijs_1s_2} = t_{is_1} + t_{s_1s_2} + t_{s_2j} - t_{ij}$$

$$\widetilde{d}_{ijs_1s_2} = d_{is_1} + d_{s_1s_2} + d_{s_2j} - d_{ij}$$

- Consideriamo l'insieme
  $L_{ij} = \{(s_1 s_2) : s_1, s_2 \in F, d_{is_1} \leq Q/r \text{ e } d_{s_2 j} \leq Q/r\}$
  come l'insieme di tutte le stazioni di ricarica che è possibile visitare lungo il cammino da i a j.
  Per limitare il numero, eliminiamo le coppie $(s_1, s_2) \in L_{ij}$ tali che $d_{s_1 j} < d_{s_2 j}$ o se $d_{is_2} < d_{is_1}$. Si ha che $(s_3, s_4) \in L_{ij}$ se esiste una coppia $(s_1, s_2) \in L_{ij}$ tale che $d_{is_1} < d_{is_3}, \quad d_{s_2 j} < d_{s_4 j}, \quad d_{ijs_3 s_4} < d_{ijs_1 s_2}$

La nuova formulazione diventa quindi:

$$\min \sum_{(i,j)\in A} d_{ij} x_{ij} + \sum_{(i,j)\in A} \sum_{(s_1,s_2)\in L_{ij}} \widetilde{d}_{ijs_1 s_2} z_{ijs_1 s_2} \tag{2.4.1}$$

$$s.t. \sum_{(s_1,s_2)\in L_{ij}} z_{ijs_1 s_2} \leq x_{ij} \quad \forall (i,j) \in A \tag{2.4.2}$$

$$\sum_{\substack{j\in V \\ j\neq i}} x_{ij} = 1 \qquad \forall i \in I \tag{2.4.3}$$

$$\sum_{\substack{i\in V \\ i\neq j}} x_{ij} = \sum_{\substack{i\in V \\ i\neq j}} x_{ji} \qquad \forall j \in V \tag{2.4.4}$$

$$\sum_{\substack{j\in V \\ j\neq 0}} x_{0j} \leq m \tag{2.4.5}$$

$$\sum_{\substack{j\in V \\ j\neq 0}} x_{j0} \leq m \tag{2.4.6}$$

$$\tau_j \geq \tau_i + (t_{ij} + p_i) x_{ij} + \sum_{(s_1,s_2)\in L_{ij}} (\widetilde{t}_{ijs_1 s_2} + p_{s_1} + p_{s_2}) z_{ijs_1 s_2} - T_{max}(1 - x_{ij}) \quad \forall i \in V, j \in I, i \neq j \tag{2.4.7}$$

$$\tau_j \leq T_{max} - (t_{j0} + p_j) - \sum_{(s_1,s_2)\in L_{j0}} (\widetilde{t}_{j0s_1 s_2} + p_{s_1} + p_{s_2}) z_{j0s_1 s_2} \qquad \forall j \in I \tag{2.4.8}$$

$$f_i \leq \sum_{(s_1,s_2)\in L_{ij}} (Q - r \cdot d_{s_2 j}) z_{ijs_1 s_2} + Q(1 - \sum_{(s_1,s_2)\in L_{ij}} z_{ijs_1 s_2}) \qquad \forall j \in I, i \in I, i \neq j \tag{2.4.9}$$

$$f_i \leq y_i - r \cdot d_{ij} x_{ij} + 2Q(1 - x_{ij} + \sum_{(s_1,s_2)\in L_{ij}} z_{ijs_1 s_2}) \quad \forall j \in I, i \in V, i \neq j \tag{2.4.10}$$

$$f_i \geq r \cdot d_{i0}(x_{i0} - \sum_{(s_1,s_2)\in L_{ij}} z_{i0s_1 s_2}) \qquad \forall i \in I \tag{2.4.11}$$

$$f_i \geq \sum_{(s_1,s_2)\in Lij} (r \cdot d_{is_1} z_{ijs_1 s_2}) \qquad \forall (i,j) \in A \tag{2.4.12}$$

$$\sum_{(s_1,s_2)\in Lij} r \cdot d_{s_2 0} z_{i0s_1 s_2} \leq Q \qquad \forall i \in I \tag{2.4.13}$$

$$f_0 = Q \tag{2.4.14}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{2.4.15}$$

$$f_i \geq 0 \quad \forall i \in V \tag{2.4.16}$$

$$\tau_i \geq 0 \qquad \forall i \in V \tag{2.4.17}$$

$$z_{ijs_1s_2} \in \{0,1\} \quad \forall i \in I, j \in I, (s_1,s_2) \in L_{ij} \tag{2.4.18}$$

La funzione obiettivo (2.4.1) minimizza la distanza percorsa. I vincoli (2.4.2) e (2.4.3) assicurano che tutti i clienti vengano visitati una e una sola volta, mentre (2.4.4) assicura la continuità del percorso. I vincoli (2.4.5)-(2.4.6) limitano il numero di veicoli. L'arrivo ad ogni nodo viene regolato da (2.4.7), e la durata massima del percorso è limitata da (2.4.8). I livelli di carburante ad ogni nodo sono controllati da (2.4.9)-(2.4.10). I vincoli (2.4.11)-(2.4.13) controllano che sia rimasto abbastanza carburante per tornare al deposito dopo la visita all'ultimo nodo, che sia un cliente o una stazione di servizio. (2.4.14) impone che i veicoli lascino il deposito con il serbatoio pieno.
Infine (2.4.15)-(2.4.18) specificano le variabili.

2.5 Formulazione di tipo set partitioning

La formulazione di tipo *Set Partitioning* (SP) è stata presentata per la prima volta in ambito VRP da Balinski et al. [1]: tale modello può essere considerato come derivante da una decomposizione di Dantzig-Wolfe [11] comune alle formulazioni viste in precedenza. Il fulcro della decomposizione di Dantzig-Wolfe consiste nel concentrarsi a risolvere un sotto-problema più piccolo rispetto al problema originale, permettendo pertanto di sorvolare sopra la complessità di quest'ultimo.
Per definire il set partitioning per il GVRP, faremo riferimento ai lavori di Bezzi et al. [5] e alle definizioni trovate in [17]. Come prima cosa, è necessario definire un insieme contenente tutti i cammini ammissibili: per fare ciò, riporteremo parte della formulazione di Bruglieri et al. [6], in cui venivano considerati esclusivamente i cammini tra stazioni di ricarica, tra cui era possibile visitare un certo numero di clienti nel tragitto.
Sia pertanto k un tale cammino, consideriamo con s_k il suo nodo di partenza, con a_k quello di arrivo, con $\widetilde{d}_k$ la distanza percorsa e $\widetilde{t}_k$ il tempo impiegato. Indichiamo infine con I_k l'insieme dei clienti serviti lungo il percorso k.
Di tutti i possibili percorsi così definiti, noi stiamo cercando un sottoinsieme di questi che sia anche ammissibile, ovvero che rispetti le condizioni dettate dal tempo e dall'autonomia dei veicoli. Bruglieri et al. [6] introducono pertanto una regola che permetterà di delimitare meglio tale insieme.
Regola di ammissibilità: dato un percorso π_k con lunghezza d_k, nodo di partenza s_k e di arrivo a_k, con un insieme $I_k \subseteq I$ di clienti, durata totale $\widetilde{t}_k$ ($\widetilde{t}_k = \frac{\widetilde{d}_k}{sp} + \sum_{i \in I_k} p_i$). Sia D_{max} la distanza massima che è possibile percorrere con il carburante pieno.
Allora π_k è ammissibile se e solo se valgono le seguenti condizioni:

1. $\widetilde{d}_k \leq D_{max}$;

2. $\frac{\widetilde{d}_{0s_k}}{sp} + \widetilde{t}_k + p_{s_k} + p_{a_k} + \frac{\widetilde{d}_{a_k 0}}{sp} \leq T_{max} - p^{start}$.

Le condizioni assicurano la fattibilità di portare a termine il percorso in termini di distanza e in termini di tempo.
Sia pertanto $\Omega = \{p : p$ è un cammino ammissibile$\}$ l'insieme di tutti i cammini ammissibili, e sia $\lambda_i^p \in \{0, 1\}$ una variabile binaria che vale 1 se il cliente i viene servito lungo il cammino p, 0 altrimenti.
Consideriamo x_p come una variabile binaria che vale 1 se e solo se il cammino p è nella soluzione $\forall p \in \Omega$, mentre c_p come il costo di tale cammino, che nel nostro caso corrisponde alla distanza tra s_p e a_p.

Basandoci quindi su quanto detto e su lavori come quello presentato da Bezzi et al. [5], possiamo definire il GVRP come:

$$min \sum_{p \in \Omega} c_p x_p \tag{2.5.1}$$

$$s.t. \sum_{p \in \Omega} \lambda_{ip} x_p = 1, \qquad \forall i \in I \tag{2.5.2}$$

$$\sum_{p \in \Omega} x_p \leq m \tag{2.5.3}$$

$$x_p \in \{0, 1\} \qquad \forall p \in \Omega \tag{2.5.4}$$

La funzione obiettivo (2.5.1) minimizza il costo del tragitto. (2.5.2) assicura che ciascun cliente sia visitato esattamente una volta. (2.5.3) restringe il massimo numero di veicoli utilizzati. (2.5.4) assicura che la variabile x_p sia binaria.

Capitolo 3

Metodi per il GVRP basati sui piani di taglio

Il metodo dei piani di taglio *(Cutting Plane)* è un metodo di risoluzione per i problemi di programmazione lineare intera (e mista) ideata da Ralph Gomory in [24] e si basa sul concetto di sviluppare una serie di disuguaglianze lineari, dette *tagli*, per ridurre la parte di regione ammissibile e condurre, attraverso la soluzione iterata di problemi di programmazione lineare, alla soluzione che soddisfa anche i vincoli di interezza.
Basandoci sui lavori di Bruglieri et al. [34] e sulle dispense [17] e [12], consideriamo una formulazione MILP di un problema (che chiameremo *Problema Originale*-PO) e otteniamo un rilassamento di questo eliminando i vincoli di interezza: questo permetterà di risolvere il problema con dei classici risolutori computazionali per programmazione lineare (ad esempio con il metodo del simplesso). Supponiamo inoltre di disporre di un insieme (anche implicito) di vincoli che, se aggiunti al problema originale, garantiscono che la soluzione produca soluzioni intere anche senza imporre i vincoli di interezza. Chiamiamo questi vincoli *tagli*. Dopo aver risolto il problema rilassato si valuta se la soluzione x^* trovata sia una soluzione ottimale anche per il problema originale, ossia se soddisfa anche i vincoli di interezza: se ciò avviene abbiamo terminato, altrimenti individuiamo uno dei tagli non rispettati da x^* e aggiungiamolo al rilassamento, continuando ad iterare fintanto che non si giunge a una soluzione ottimale intera.
Si tratta quindi di aggiungere ai vincoli già presenti altri vincoli col compito di "tagliare fuori" le soluzioni frazionarie che, in quanto tali, non possono essere una soluzione del problema originario. Questo procedimento viene iterato fintanto che non si raggiunge una soluzione intera, che corrisponderà alla soluzione ottimale.
Definiamo ora l'algoritmo per il metodo dei piani di taglio.
Per semplicità, consideriamo di avere un problema lineare intero (nel caso di variabili miste basterà limitarsi a considerare solamente quelle intere)
Sia $\Omega \subseteq \mathbb{Z}^n$ la regione ammissibile del problema e sia $a^T x \leq \beta$ un suo taglio.

1. Risolviamo il rilassamento lineare e troviamo una soluzione $\overline{x}$.

2. Distinguiamo ora due casi: se $\overline{x}$ è intera allora abbiamo concluso. Altrimenti cerchiamo un taglio $a^T x \leq \beta$ che sia valido per Ω ma violato da $\overline{x}$. Aggiungiamo quindi il taglio al rilassamento lineare e ripetiamo il punto 1.

La ricerca dei tagli da aggiungere risulta pertanto essere di grande importanza, siccome con l'aggiunta di un numero eccessivo di tagli si rischia di avere un problema nuovamente troppo complesso. D'altro canto, un numero troppo ridotto di tagli rischia di portare ad avere un rilassamento insufficiente. Risulta pertanto conveniente eliminare qualche taglio quando questi cominciano a non essere efficaci.

3.1 Tagli di Gomory

Uno dei primi, e tra i più importanti, metodi di taglio è il metodo dei *tagli di Gomory*, presentato da Gomory in [26].
Questi tagli sono applicabili a qualsiasi problema MILP e hanno la caratteristica di garantire che un metodo di piani di taglio che li usi come tagli converge alla soluzioni ottima intera, sotto assunzioni normalmente ragionevoli per problemi pratici (ad esempio, la razionalità dei coefficienti).
In questa sezione illustreremo la versione per i problemi MILP ideata dallo stesso in [25] e studiata poi da Marchand et al. in [35, pag. 401].
Consideriamo un generico programma lineare $\min\{c^T x : x \in X\}$ dove $X = \{x \in \mathbb{Z}^n : Ax = b\}$. Sia x^* una soluzione ottimale del rilassamento lineare $\min\{c^T x : x \in P\}$, $P \subseteq \mathbb{R}^{n+}$ e sia $B \subseteq \{1, ..., n\}$ una base di A con $x^*_B = A_B^{-1} b - A_B^{-1} A_N x_N$ e $x^*_N = 0$, dove $N = \{1, ..., n\} \setminus B$.
Se x^* è intera, abbiamo trovato la soluzione ottimale per $\min\{c^T x : x \in X\}$.
Altrimenti, uno dei valori di x^*_B deve essere frazionario.
Sia $i \in B$ tale che $x^*_i \notin \mathbb{Z}$. Siccome ogni soluzione intera accettabile $x \in X$ soddisfa $x_B = A_B^{-1} b - A_B^{-1} A_N x_N$

$$A_i^{-1} b - \sum_{j \in N} A_i^{-1} A_j x_j \in \mathbb{Z} \tag{3.1.1}$$

Osservazione: sia $(a^k)^T x \leq \alpha^k$ una disuguaglianza valida per un poliedro $P^k \subseteq \mathbb{R}^{n+}$ per $k = 1, 2$.
Allora

$$\sum_{i=1}^{n} \min(a_i^1, a_i^2) x_i \leq \max(\alpha^1, \alpha^2) \tag{3.1.2}$$

è valida per $P^1 \cup P^2$ e $conv(P^1 \cup P^2)$.
Definiamo ora la funzione f tale che $f(\alpha) = \alpha - |\alpha|$ per $\alpha \in \mathbb{R}$. Indichiamo

$\overline{a}_j = A_i^{-1} A_j$ $\quad$ $\overline{b} = A_i^{-1} b$
$f_i = f(\overline{a}_j)$ $\quad$ $f_0 = f(\overline{b})$
$N^+ = \{j \in N : \overline{a}_j \geq 0\}$ $\quad$ $N^- = N \setminus N^+$

Allora (3.1.1) diventa $\sum_{j\in N} \overline{a}_j x_j = f_0 + k$ per qualche $k \in \mathbb{Z}$. Consideriamo i due casi $\sum_{j\in N} \overline{a}_j x_j \geq 0$ e $\sum_{j\in N} \overline{a}_j x_j \leq 0$.
Nel primo caso abbiamo che $\sum_{j\in N^+} \overline{a}_j x_j \geq f_0$ deve essere valida. Nel secondo abbiamo invece $\sum_{j\in N^-} \overline{a}_j x_j \leq f_0 - 1$ che è equivalente a

$$-\frac{f_0}{1-f_0} \sum_{j\in N^-} \overline{a}_j x_j \geq f_0$$

Applicando l'osservazione precedente, abbiamo che $P^1 = P \cap \{x : \sum_{j\in N} \overline{a}_j x_j \geq 0\}$ e $P^2 = P \cap \{x : \sum_{j\in N} \overline{a}_j x_j \leq 0\}$, ottenendo la seguente disuguaglianza valida

$$\sum_{j\in N^+} \overline{a}_j x_j - \frac{f_0}{1-f_0} \sum_{j\in N^-} \overline{a}_j x_j \geq f_0 \tag{3.1.3}$$

Questa disequazione può essere rafforzata nel seguente modo: osserviamo che la sua derivata rimane inalterata all'aggiunta di multipli interi a variabili intere. Facendo in questo modo mettiamo ciascuna variabile in N^+ o in N^-, definiti come sopra. Se una variabile è in N^+, il coefficiente finale di (3.1.3) è $\overline{a}_j$ e il miglior coefficiente che si può ottenere dopo l'aggiunta di multipli interi è $f_i = f(\overline{a}_j)$.
In N^- invece si ha che il coefficiente finale risulta $(\frac{f_0}{1-f_0})\overline{a}_j$, e il miglior coefficiente possibile è $\frac{f_0(1-f_j)}{1-f_0}$. Nel complesso otteniamo il miglior coefficiente usando $min\{f_i, f_0(1-f_i)/(1-f_0)\}$. In questo modo si ottiene il miglior taglio intero misto di Gomory.

$$\sum_{\substack{j:f_j\leq f_0 \\ j\in\mathbb{Z}}} f_j x_j + \sum_{\substack{j:f_j> f_0 \\ j\in\mathbb{Z}}} \frac{f_0(1-f_j)}{1-f_0} x_j$$

$$+ \sum_{\substack{j\in N^+ \\ j\notin\mathbb{Z}}} \overline{a}_j x_j - \sum_{\substack{j\in N^- \\ j\notin\mathbb{Z}}} \frac{f_0}{1-f_0} \overline{a}_j x_j \geq f_0 \tag{3.1.4}$$

Si dimostra che che un algoritmo basato sull'aggiunta iterativa di queste disequazioni risolve $min\{c^T x : x \in X\}$ con $X = \{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : Ax = b\}$ in un numero finito di passaggi, se si conosce $c^T x \in \mathbb{Z} \quad \forall x \in X$.

3.2 Esempi di Tagli specifici per problemi VRP

Proponiamo ora alcuni esempi di tagli specifici per formulazioni di problemi della famiglia VRP, suddividendoli in varie categorie.

Eliminazione dei sotto-cammini (Laporte et al. [31])
Vengono utilizzati per eliminare le soluzioni che contengono percorsi chiusi che non visitano il deposito.
Per ogni sottoinsieme S dei nodi (escluso il deposito) si ha che

$$\sum_{i,j \in S} (x_{ij} + \sum_{k \in F_{ijk}}) \leq |S| - r_{GVRP}(S)$$

Dove r_{GVRP} corrisponde al numero minimo di veicoli necessari.
Proponiamo anche una versione elaborata da Koc et al. [29] per la formulazione mostrata nella Sezione 2.3:

$$\sum_{j \in V'} (x_{0j} + \sum_{k \in F} y_{0jk}) \geq r_{GVRP}$$

Tagli di capacità (Clarke et al. [7])
Assicurano che la capacità di un veicolo non venga superata. Vengono utilizzati per problemi come il *Capacitated Vehicle Routing Problem* (CVRP) o la sua variante green, il CGVRP, dove è necessario monitorare anche la capacità dei veicoli che trasportano la merce.
Una formulazione è data da:

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 2 \left\lceil \frac{p_S}{Cap} \right\rceil$$

Dove p_S indica la domanda totale dei clienti in un sottoinsieme $S \subset V$, e Cap è la capacità del veicolo.
Tagli Combinatori
Derivano dalle proprietà combinatorie specifiche del GVRP. Di seguito presentiamo alcuni di questi elaborati da Bruglieri et al. [6] per la formulazione presentata nella Sezione 2.4, ma è possibile adattarle alla formulazione presentata nella Sezione 2.3 facilmente considerando il caso in cui $s_1 = s_2$ con $s_1, s_2 \in L_{ij}$.
Sia $L_{ij}, \forall i, j \in I$ l'insieme che comprende tutte le stazioni di ricarica che possibile visitare andando dal punto i al punto j, già introdotto in precedenza nella Sezione 2.4 $L_{ij} := \{(s_1, s_2) : s_1, s_2 \in I, d_{is_1} \leq Q/r$ e $d_{s_2 j} \leq Q/r\}$ e la variabile binaria $z_{ijs_1s_2}$ che ricordiamo avere valore 1 se le stazioni s_1 e s_2 sono visitate consecutivamente, e 0 altrimenti.
La prima disequazione che viene presentata (3.2.1) rafforza la necessità di ricarica di un veicolo che viaggia dal punto i al punto j $(i, j \in I)$ se raggiunto quest'ultimo non è più possibile raggiungere una stazione di servizio con il carburante completamente ricaricato.

$$\sum_{(s_1,s_2) \in L_{ij}} z_{ijs_1s_2} \geq x_{ij} \qquad \forall (i,j) \in A : d_{ij} > Q/r \tag{3.2.1}$$

Il vincolo (3.2.2) impedisce invece che si visiti una stazione di servizio una volta visitato l'ultimo cliente se non si ha abbastanza carburante per tornare al deposito. Infatti considerando ϵ come una costante positiva e abbastanza piccola e notando che $f_i - rd_{i0} > 0$ otteniamo che $z_{i0s_1s_2}$ debba essere un numero intero strettamente minore di 1, ovvero 0.

$$\sum_{(s_1,s_2)\in L_{i0}} z_{i0s_1s_2} \leq 1 - \epsilon \cdot (f_i - rd_{i0}) \forall i \in V \tag{3.2.2}$$

La disequazione (3.2.3) impedisce invece di servire una terna di clienti i, j, k senza fermate di servizio intermedie se la distanza necessaria per ciò sommata alla distanza minima tra la stazione di servizio più vicina (o il deposito) supera la distanza massima consentita al veicolo a carburante pieno.
Indichiamo con $D = \max_{\substack{i,j\in F\cup I_0 \\ i\neq j}} d_{ij}$ la distanza massima percorribile.
Possiamo notare come (3.2.3) sia vera in quanto se la distanza non rispetta quanto richiesto sopra, allora il termine diventa strettamente minore di due: infatti le due sommatorie diventano 0, e il termine $(d_{ij} + d_{jk} + \min_{s\in F\cup 0} d_{ks} - f_i/r)/3D$ risulta essere un numero positivo minore di 1. Pertanto perché al disequazione sia rispettata si deve avere che x_{ij} o x_{jk} si annullino.

$$x_{ij} + x_{jk} \leq 2 + \sum_{(s_1,s_2)\in L_{ij}} z_{ijs_1s_2} + \sum_{(s_1,s_2)\in L_{jk}} z_{jks_1s_2}$$

$$-\frac{d_{ij} + d_{jk} + \min_{s\in F\cup 0} d_{ks} - y_i/r}{3D} \qquad \forall (i,j) \in A, \forall (j,k) \in A \tag{3.2.3}$$

Il vincolo (3.2.4) risulta simile al vincolo (3.2.3), con la distanza tra i tre punti sommata alla distanza di ogni stazione vicina (o dal deposito) supera la distanza che può essere percorsa nel tempo massimo rimanente.

$$x_{ij} + x_{jk} \leq 2 + \sum_{(s_1,s_2)\in L_{ij}} z_{ijs_1s_2} + \sum_{(s_1,s_2)\in L_{jk}} z_{jks_1s_2}$$

$$-\frac{d_{ij} + d_{jk} + \min_{s\in F\cup 0} d_{ks} - (T_{max} - \tau_i)\cdot v}{3D} \qquad \forall (i,j) \in A, \forall (j,k) \in A \tag{3.2.4}$$

K-tagli (Desaulniers et al. [14])
Si tratta di vincoli che vengono utilizzati per assicurarsi che non esistano percorsi che visitano meno di k nodi senza passare per il deposito. Dato un insieme I di clienti, la corrispondente disequazione è

$$\sum_{\substack{j\in I \\ j\neq i}} x_{ij} \geq k \qquad \forall i \in I$$

Capitolo 4

Metodi per il GVRP basati su Branch and Cut

All'interno delle soluzioni esatte proposte per il GVRP, il Branch and Cut ($B\&C$) risulta essere tra quelli usati maggiormente: viene sviluppato per la prima volta da Padberg et al. in [38], come tecnica di ottimizzazione combinatoria che unisce il metodo del Branch-and-Bound con le tecniche di taglio.
Gadegaard et al. [36] utilizzano l'algoritmo per trovare la soluzione del *Mixed Fleet Green Vehicle Routing Problem* (MFGVRP), ovvero il sottogruppo del GVRP che studia il caso in cui i veicoli a disposizioni siano di diverso tipo, il che comporta l'avere diverse stazioni di ricarica e diverse proprietà di autonomia. La loro versione inoltre risulta essere anche un sottogruppo del CVRP, aggiungendo pertanto la necessita di monitorare anche la capienza dei veicoli. Il $B\&C$ da loro utilizzato si rivela essere molto efficace, riuscendo a superare i risultati ottenuti da un modello MILP risolto tramite CPLEX, un solver allo stato dell'arte per programmazione matematica.
Un utilizzo di questo algoritmo si può trovare nel lavori di Fazeli et al. [19], dove viene studiato il sottogruppo del GVRP riguardante il *min/max electric vehicle routing problem* (MEVRP), dove viene chiesto di minimizzare la distanza massima di un veicolo elettrico, considerando anche la presenza di AFS. Nel loro lavoro, confrontano l'algoritmo $B\&C$ con degli algoritmi euristici, provando come sebbene non sia in grado di competere in termini di tempistiche e di quantità si soluzioni trovate, è comunque in grado di avvicinarsi con alla soluzione ottimale.
L'algoritmo $B\&C$ viene però molto spesso integrato assieme ad altri algoritmi: è questo il caso del lavoro di Koc et. al [29], dove viene implemento con un algoritmo di *simulated annealing* per la soluzione del GVRP, introducendo anche la formulazione presentata nella Sezione 2.3. Tale lavoro mostra che l'algoritmo $B\&C$ (unito al simulated annealing) è in grado di risolvere ottimamente 22 su 40 istanze in un tempo ragionevole.

4.1 Branch and Bound

Il metodo Branch-and-Bound è un metodo iterativo per la risoluzione di problemi di programmazione lineare intera (o mista) sviluppato a partire dal partizionamento della

regione accettabile di un problema e ne valuta successivamente la funzione obiettivo permettendo in alcuni casi di non considerare parte dei sottogruppi che si vengono a creare. Viene sviluppato per la prima volta da Doig et al. [30] per la risoluzione del *Traveling Salesman Problem*, e da allora è uno dei metodi alla base di molte soluzioni per la programmazione lineare intera.
Cominciamo con alcune considerazioni.

Lemma 4.1.1. *[17] Il valore ottimale di un rilassamento ottimale è un limite superiore del valore ottimale del problema lineare intero.*

Lemma 4.1.2. *[17] Se la regione accettabile Ω di un problema di ottimizzazione è diviso in due sottoinsiemi (che chiameremo $\Omega = \Omega_1 \cup \Omega_2$), il valore ottimale del problema iniziale è il valore migliore dei due sotto-problemi:*

$$\max\{c^T x : x \in \Omega\} = \max\{max\{c^T : x \in \Omega_1\}, \max\{c^T x : x \in \Omega_2\}\}$$

Basandoci su [17] e [12], consideriamo quindi un problema di programmazione lineare mista, indicando con x_i le sue variabili intere e con y_i quelle continue.
Sia P_0 il suo rilassamento lineare e $(\overline{x}, \overline{y})$ una sua soluzione ottenuta con un qualsiasi metodo risolutivo per problemi lineari (ad esempio con il metodo del simplesso). Se il rilassamento lineare non ha soluzioni accettabili, allora abbiamo concluso in quanto non ne ha neanche il problema originale
Studiamo $\overline{x}$: se è intera, allora abbiamo concluso e $(\overline{x}, \overline{y})$ è soluzione anche del problema originale.
Altrimenti, individuiamo un indice i tale che $\overline{x}_i$ non sia intera: consideriamo i due sotto-problemi che si formano aggiungendo i vincoli $x_i \geq [\overline{x_i}]$ e $x_i \leq [\overline{x_i}]$ (questa azione prende il nome di *branching*).
Siccome le regioni accettabili dei sotto-problemi formano insieme la regione accettabile del problema originario (ogni soluzione di PO è contenuta in uno dei due sotto-problemi) possiamo usare il Lemma (4.1.2).
A questo punto si comincia a iterare: definendo un sotto-problema come *attivo* se può contenere la soluzione intera del problema originario, si procede risolvendo i sotto-problemi e a studiarne le rispettive soluzioni.
C'è da osservare che i possibili esiti del processo di branching sono due: un successo se la soluzione trovata risulta essere realizzabile, oppure un fallimento se la soluzione non è realizzabile o se è peggiore della migliore soluzione attuale.
Siccome i due sotto-problemi venutisi a formare continuano a soddisfare i vincoli del problema originale, per il Lemma (4.1.2) sappiamo che le soluzioni ottenute da questi due problemi comprendono la soluzione del problema originale.
Per poter confrontare le varie soluzioni realizzabili ottenute, introduciamo la *soluzione titolare* x^*: se la soluzione realizzabile che è stata trovata è la prima dell'algoritmo, allora fissiamo tale valore come titolare; altrimenti compariamola con l'attuale soluzione titolare.
Se la nuova soluzione è migliore della precedente, diventa la nuova soluzione titolare: in tal caso viene eliminato il sotto-problema di entrambe le soluzioni

Per evitare di dover controllare un numero eccessivo di possibili soluzioni e quindi minimizzare il tempo di esecuzione, vengono adottati una serie di criteri di esplorazione, cioè criteri che consentano di eliminare parte dei sotto-problemi.
Questa fase prende il nome di *potatura* (*bounding*) e può essere eseguita secondo tre diversi criteri:

Potatura per inammissibilità: avviene per ragioni di realizzabilità, dovute allo stato irrealizzabile di un sotto-problema P, che porterebbe pertanto a soluzioni irrealizzabili;

Potatura per bound: Il valore ottimale del rilassamento del problema (P_k) è peggiore di quello del valore titolare in atto;

Potatura per ottimalità: avviene quando si trova la soluzione ottimale del problema (P_k), in quanto non è possibile trovare soluzioni migliori.

Secondo quanto mostrato nelle dispense [17] e [12], considerando (P_0) come il problema originale ed I l'insieme degli indici delle variabili intere, formalmente l'algoritmo risulta essere il seguente.
Sia dato il problema MILP

$$z_I = \max c^T x$$

$$Ax \leq b$$

$$x \geq 0$$

$$x_i \in \mathbb{Z} \qquad i \in I$$

Indichiamo con LB (*Lower Bound*) il limite inferiore delle soluzioni z_I.
L'algoritmo per il Branch-and-Bound è quindi:

1. Se esiste un nodo T attivo, scegliere il nodo (P_k). Altrimenti ritorno la soluzione ottimale x^*.

2. Risolvere il rilassamento lineare di (P_k) o ottenendo una soluzione ottima x^k di valore z_L^k oppure concludendo che il rilassamento lineare di (P_k) non è ammissibile.

   (a) Se il rilassamento non è ammissibile, eliminare P_k da T.

   (b) Se $z_L^k \leq LB$ allora (P_k) non può avere soluzioni migliori di quella corrente x^*, e si elimina (P_k) da T.

   (c) Se $x^k \in Z \quad \forall i \in I$ allora x^k è soluzione ottima per (P_k) ed ammissibile per (P_0), dunque:

   - Se $c^T x^k > LB$ si pone $x* = x^k$ e $LB = c^T x^k$.
   - Potare (P_k) da T.

3. Se nessuno dei casi (a)(b)(c) si verifica, allora scegliere un indice $h \in I$ tale che $x_h^k \notin Z$ non sia intera e ramificare sulla variabile x_h, ponendo come figli di (P_k) in T i problemi:

$$(P_{l+1}) := (P_l) \cap \{x_h < [x_h^k]\}$$
$$(P_{l+2}) := (P_l) \cap \{x_h \geq [x_h^k]\}$$

Porre infine l:= l+2 e tornare al punto 1.

□

4.2 Branch and cut

Il Branch-and-cut ($B\&C$) viene sviluppato per la prima volta da Padbeg et al. [38] e consiste in un algoritmo esatto basato sulla combinazione dell'algoritmo Branch-and-Bound con i piani di taglio: questi ultimi permettono di migliorare il rilassamento del problema in modo da migliorare i bound e facilitare l'occorrenza di soluzioni intere, mentre l'algoritmo Branch-and-Bound procede con un approccio "dividi e conquista", come appena illustrato. Definiamo un generico MILP

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0 \\ & x_i \quad intero, i = 1, ..., p \end{array}$$

Dove x e c sono n-vettori, b è un m-vettore e A matrice $m \times n$. Le prime p variabili sono intere, le restanti frazionarie.
Secondo Mitchell [37], l'algoritmo del Branch-and-cut risulta essere :

1. *Inizializzazione*: definiamo il problema iniziale come P^0, e fissiamo i problemi attivi $L = \{P^0\}$.
   Fissiamo il limite superiore come $\overline{z}_l = +\infty$ e $z_l^* = -\infty$ per $l \in L$

2. *Condizioni di arresto*: Se $L = \emptyset$ allora la soluzione x^* che ha come valore obiettivo $\overline{z}$ è ottimale.
   Se non esiste tale x^*, allora il problema è inammissibile.

3. *Selezione dei problemi*: Selezioniamo ed estraiamo un problema P^1 da L.

4. *Rilassamento*: Risolviamo il rilassamento lineare di P^1. Se ciò è inammissibile, allora $z_l^* = +\infty$ e andiamo al punto 6.
   Altrimenti, sia z_l^* il valore obiettivo ottimale ottenuto dalla rilassamento: se tale valore è finito poniamo allora x_l come la soluzione ottimale, altrimenti fissiamo $z_l^* = -\infty$.

5. *Aggiunta tagli*: Cerchiamo alcuni tagli violati da x_l da aggiungere in caso al rilassamento. Andare al punto 4.

6. *Esplorazione e Potatura*:

   - Se $z_l^* \geq \overline{z}$ andare al punto 2.
   - Se $z_l^* < \overline{z}$ e x_l, poniamo $\overline{z} = z_l$, eliminiamo tutti i problemi con $z_l^* \geq \overline{z}$ da L e andiamo al punto 2.

7. *Partizionamento*: Sia $\{S^{lj}\}_{j=1}^{j=k}$ una partizione dell'insieme di vincoli S^l del problema MILP. Aggiungere i problemi $\{P^{lj}\}_{j=1}^{j=k}$ a L, dove P^{lj} è P^l con la regione accettabile limitata a S^{lj} e z_{lj}^* (j=1,...,k) è l'insieme dei valori di z_l^* per il problema genitore di l. Andare al punto 2.

4.3 Aspetti implementativi

L'algoritmo $B\&C$ richiede una pesante quantità di memoria per problemi come il GVRP, pertanto sono stati sviluppati diversi metodi per semplificare ed implementare il suo algoritmo.Riportiamo ora alcuni di questi metodi dal lavoro di Mitchell [37].

Inizializzazione: si tratta di una tecnica basata sull'usare delle soluzioni iniziali vicine alla soluzione ottimale attesa per diminuire il divario tra il bound ottenuto dai rilassamenti continui e la soluzione titolare.

Generazione di Colonne: questo metodo consiste nell'eliminare inizialmente alcune delle variabili del sistema per aggiungerle dinamicamente all'occorrenza. Si tratta del metodo Branch-and-Price-and-Cut che andremo a vedere più nel dettaglio nella Sezione 5.4.

Soluzioni euristiche: notando come il trovare una buona soluzione ci permetta di potare parte del problema, possiamo usare un buon algoritmo per passare da una soluzione frazionaria di un problema a una buona soluzione intera, così da eliminare altri sotto-problemi. Alcuni esempi sono le combinazioni con metodi come *local search* e altre *metaeuristiche*.

Lifting di tagli: non tutti i tagli che aggiungiamo possono essere validi per tutti i sotto-problemi. Affinché un taglio diventi ammissibile per ogni sotto-problema, è necessario sottoporlo a *lifting*, ovvero modificarlo affinché sia applicabile per tutti i sotto-problemi. Prendiamo ad esempio il caso di un problema binario, e consideriamo la disequazione:

$$\sum_{j \in J} a_j x_j \leq h \text{ per qualche sottoinsieme } J \subseteq \{1, ..., n\}$$

Se essa risulta essere valida in un nodo dove la variabile x_i è stata fissata a zero, allora può essere modificata come

$$\sum_{j \in J} a_j x_j + a_i x_i \leq h$$

dove a_i corrisponde a uno scalare il cui valore viene scelto in modo tale da rendere la disuguaglianza il più forte possibile.

Capitolo 5

Metodi per il GVRP basati su Branch and Price

Il Branch-and-Price ($B\&P$) è una tecnica risolutiva che si basa sugli algoritmi del Branch-and-Bound, già visto in precedenza, e della generazione di colonne, che vedremo in questa sezione.
In questo algoritmo infatti durante l'esplorazione dei nodi dell'albero i vari problemi vengono risolti usando il metodo della generazione di colonne: questa tecnica viene introdotta per la prima volta da Toth et al. in [43] nei primi anni '90 per la risoluzione di un problema di VRP, gettando le basi per numerose applicazioni successive del Branch-and-Price in vari problemi di ottimizzazione combinatoria, dimostrando la potenza e la versatilità di questo approccio (Behenke et al. [4], Kazu et al. [44], Bezzi et al. [5], per citarne alcuni).
In Bezzi et al. [5] si considera una variante dell' Electric Vehicle Routing Problem, in particolare nel caso di un insieme di diversi veicoli con capacità limitata che devono visitare un insieme di clienti per soddisfarne le richieste. Per cercare la soluzione ottimale, viene utilizzato un algoritmo di $B\&P$ implementato con l'aggiunta di alcune accortezze per ridurre il numero di iterazioni necessarie, come l'aggiunta di condizioni per evitare duplicati nella generazione di percorsi. Si passa poi a ramificare, prima sul numero di veicoli e successivamente sui singoli archi.
Hartl et al. [27] considerano invece l'Electric Fleet Size and Mix Vehicle Routing Problem with Time Windows and Recharging Station (E-FSMFFTW), ovvero nel caso in cui si aggiunga la richiesta di avere un intervallo di tempo in cui un cliente può essere servito, portando al dover considerare anche gli orari del percorso e i luoghi di ricarica dei veicoli. I veicoli disponibili inoltre si differenziano tra loro per quanto riguardo il tipo, dimensioni della batteria e costo di acquisto. L'algoritmo proposto per la soluzione è simile al $B\&P$ proposta per il VRPTW [14] , ma differisce per l'aggiunta delle stazioni di ricarica, tipiche dei GVRP.

5.1 Metodo di Generazione di Colonne

Il metodo di Generazione di Colonne (CG) viene utilizzato per i VRP per la prima volta da Desrochers et al. [15] nel 1992, applicandolo al *Vehicle Routing Problem With Time Windows* con un numero illimitato di veicoli disponibili.
Nel loro lavoro, risolvono con la generazione di colonne il rilassamento lineare del set partitioning del VRPTW, aggiungendo colonne al bisogno per risolvere un sotto-problema tramite un metodo di programmazione dinamica.
Il metodo di generazione di colonne (vedi [17]) permette infatti di non considerare alcune variabili del problema, sviluppando un sotto-problema tramite il rilassamento lineare, che prende il nome di *Problema Principale* (Master Problem-MP).
Una volta ottenuta una soluzione per il MP, si considerano alcune variabili non considerate precedentemente per migliorare la soluzione trovata.
Per quanto riguarda il GVRP, basandoci sul lavoro di Bezzi et al. [5] e sulle dispense [17], definiamo come problema principale il rilassamento lineare della formulazione set partitioning presentata nella Sezione 2.5.

$$\min \sum_{p \in \Omega} c_p x_p \tag{5.1.1}$$

$$s.t. \sum_{p \in \Omega} x_p \leq m \tag{5.1.2}$$

$$\sum_{p \in \Omega} \lambda_{ip} x_p = 1 \qquad \forall i \in I \tag{5.1.3}$$

$$x_p \in \{0, 1\} \qquad \forall p \in \Omega \tag{5.1.4}$$

Dove le colonne realizzate dalle variabili corrispondono ai percorsi accettabili.
A causa del numero di cammini elevato, risulta molto difficile enumerarli tutti e potrebbe non essere possibile: consideriamo per questo un insieme $S \subseteq \Omega$, comprendente solo alcuni dei percorsi accettabili.
Il problema che ne deriva prende il nome di *Problema principale ristretto* (Restricted Master Problem-RMP).

$$\min \qquad \sum_{p \in S} c_p x_p \tag{5.1.5}$$

$$s.t. \sum_{p \in S} x_p \leq m \tag{5.1.6}$$

$$\sum_{p \in S} \lambda_{ip} x_p = 1 \qquad \forall i \in I \tag{5.1.7}$$

$$x_p \geq 0 \qquad \forall p \in S \tag{5.1.8}$$

Definiamo ora le variabili duali $\pi_0 \in \mathbb{R}$ e $\pi_i \in \mathbb{R} \quad \forall i \in I$ facenti riferimento rispettivamente ai vincoli (5.1.6) e (5.1.7). Ora non resta che trovare le colonne da inserire all'interno

del MP e per fare ciò è necessario risolvere il seguente sotto-problema:

$$c_p^* = \min_{p \in \Omega} \{c_p - \sum_{i \in I} \lambda_{ip} \pi_i - \pi_0\} \tag{5.1.9}$$

che prende il nome di *Pricing Problem*.

Il valore c_p^* indica il costo ridotto della soluzione ottimale $p^* \in \Omega$. Nel caso del GVRP, tale costo corrisponde alla distanza percorsa.
Per la risoluzione del pricing problem possono essere usati algoritmi euristici, come il *Tabu Search* [14], oppure esatti, come *l'algoritmo di etichettatura bidirezionale* che vedremo in seguito nella Sezione 5.2.
Ogni volta che viene trovato un cammino con costo c_P^* negativo, allora questo viene inserito nell'insieme S per il RMP.
Siccome è possibile trovare diversi cammini con costo c_P^* negativo, si troveranno nuovi cammini ad ogni nuova iterazione del problema.
Il metodo termina nel caso non sia più possibile generare cammini che soddisfino tale condizione: raramente la soluzione trovata risulta intera, per questo prende il via la parte di algoritmo relativa al Branch-and-Bound.

5.2 Algoritmo di etichettatura bidirezionale

Per risolvere il Pricing Problem viene utilizzato principalmente un algoritmo esatto, denominato di etichettatura bidirezionale (*bi-directional label setting algorithm*), variante del più classico labeling setting tratto da Irnich et al. [28].
L'algoritmo che ora andremo a presentare è stato formulato accorpando quello elaborato in diversi lavori, più precisamente nei lavori di Dabia et al [8], Hartl et al. [27] e Yu et al. [46]. Questo agisce eseguendo una ricerca simultanea partendo sia dal nodo iniziale e proseguendo in avanti (forward), sia partendo da quello finale e proseguendo all'indietro (backward). Le due ricerche poi verranno combinate per trovare il percorso ottimale.
Cominciamo definendo l'algoritmo in avanti: sia p un cammino parziale che parte dal deposito d e definiamo con L_p^f l'etichetta associata a p, corrispondente a una tupla $(p, v, R_t, R_f, R_{\overline{c}}, S)$ dove v rappresenta l'ultimo vertice visitato, R_t il tempo di arrivo in suddetto vertice, $R_{\overline{c}}$ il costo del percorso, R_f la quantità di carburante rimasta e l'insieme $S \subseteq \Omega$ consiste nell'insieme dei vertici che non è possibile visitare.
Poniamo lo stato iniziale $L_0^f = (\{0\}, d, 0, Q, 0, \emptyset)$, corrispondente al cammino che parte dal deposito e ha pertanto quantità di carburante massima Q. Considerando un arco $(i, j) \in A$. È possibile definire un'estensione di L_p^f con tale arco se valgono le seguente condizioni:
$i = v(L^f)$
$j \notin S$
$R_t + t_{ij} \leq T_{max}$
$d_{ij} \cdot r \leq R_f(L)$
In tal caso, chiamando la nuova etichetta L'^f, otteniamo i nuovi valori:

$$p(L'^f) = p(L^f) \cup \{j\} \qquad R_{\overline{c}}(L'^f) = R_{\overline{c}}(L^f) + \overline{c}_{ij}$$
$$R_t(L'^f) = R_t(L^f) + t_{ij} \qquad R_f(L'^f) = \begin{cases} R_f(L^f) - d_{ij} \cdot r & se \quad i \in I \\ Q \quad \text{altrimenti} \end{cases}$$
$$v(L'^f) = j \qquad S(L'^f) = S(L^f) \cup \{c | t(L'^f) + t_{jc} > T_{max}\}$$

Per ridurre il numero di etichette, applichiamo un metodo di eliminazione basato su quello proposto da Feillet et al. [21].
Siano L_1 ed L_2 etichette in avanti(o indietro). L_1 domina L_2 se:

$$v(L_1) = v(L_2) \qquad R_t(L_1) \leq R_t(L_2)$$
$$R_{\overline{c}}(L_1) \leq R_{\overline{c}}(L_2) \qquad R_f(L_1) \geq R_f(L_2)$$
$$S(L_1) \subseteq S(L_2)$$

Vediamo ora come si definisce e sviluppa la versione indietro.
Definiamo con L_p^b l'etichetta associata al percorso p, corrispondente a una tupla $(p, v, R_t, R_f, R_{\overline{c}}, S)$ come prima.
Lo stato iniziale sarà $L_0^b = \{\{0\}, d, 0, 0, 0, \emptyset\}$. Una estensione di L^b con un arco $(i, j) \in A$ è possibile se valgono le seguenti condizioni:
$j = v(L^b)$
$i \notin S$
$d_{ij} \cdot r + R_f(L^b) \leq Q$
$t_{ij} + R_t \leq T_{max}$
In tal caso, chiamando la nuova etichetta L'^b, otteniamo:

$$p(L'^b) = p(L^b) \cup \{i\} \qquad R_{\overline{c}}(L'^b) = R_{\overline{c}}(L^b) + \overline{c}_{ij}$$
$$R_t(L'^b) = R_t(L^b) + t_{ij} \qquad R_f(L'^b) = \begin{cases} R_f(L^b) + d_{ij} \cdot r & se \quad i \in I \\ 0 \quad \text{altrimenti} \end{cases}$$
$$v(L'^b) = i \qquad S(L'^b) = S(L^b) \cup \{c | t(L'^b) + t_{jc} > T_{max}\}$$

Come per il metodo "in avanti", definiamo un criterio di dominanza che ci permetta di eliminare alcune delle liste (tratto sempre dal lavoro di Feillet et al. [21]).
Siano L_1 e L_2 etichette indietro. L_1 domina su L_2 se:

$$v(L_1) = v(L_2) \qquad R_t(L_1) \leq R_t(L_2)$$
$$R_{\overline{c}}(L_1) \leq R_{\overline{c}}(L_2) \qquad R_f(L_1) \leq R_f(L_2)$$
$$S(L_1) \subseteq S(L_2)$$

Dopo l'estensione, si applica una regola di combinazione per concatenare le due liste (avanti e indietro), se le seguenti condizioni sono soddisfatte:

$v(L^f) = v(L^b)$ $\quad$ $R_t(L^f) + R_t(L^b) \leq T_{max}$
$S(L^f) \cap S(L^b) = \emptyset$ $\quad$ $R_f(L^f) \geq R_f(L^b)$

Una volta che le due etichette sono state fuse insieme, i nuovi valori sono:

$p(L) = p(L^f) \cup p(L^b)$ $\quad$ $v(L)L = d$
$R_t(L) = R_t(L^f) + R_t(L^b)$ $\quad$ $R_f(L) = R_f(L^f) - R_f(L^b)$
$S(L) = S(L^f) \cup S(L^b)$ $\quad$ $R_{\bar{c}}(L) = R_{\bar{c}}(L^f) + R_{\bar{c}}(L^b)$

5.3 Algoritmo Branch and Price

La struttura base dell'algoritmo Branch-and-Price corrisponde l'algoritmo Branch-and-Bound, che si occupa di suddividere il problema originale in sotto-problemi più semplici e usare le soluzioni così trovate per restringere la ricerca della soluzione ottimale. Basandoci sugli studi effettuati da Feillet [20] e da Barnhart et al [3] possiamo infatti studiare come durante l'esplorazione dei vari nodi, nei sotto-problemi venga utilizzata la generazione di colonne: tale tecnica consiste nell'aggiungere alcune variabili (colonne) al problema solo quando questo risulta necessario. Tali colonne vengono trovate risolvendo un sotto-problema, che individua quali variabili possono migliorare la soluzione corrente.
La chiave di questo approccio risiede nella conoscenza che per problemi di programmazione matematica con un grande numero di variabili, la maggior parte delle colonne non fanno parte di una base e per tanto avranno la variabile corrispondente nulla in ogni soluzione.
Iniziamo definendo il problema iniziale con una formulazione set partitioning come mostrato nella Sezione 2.5:

1. Riformuliamo il problema iniziale con una riformulazione, solitamente la decomposizione di Dantzig–Wolfe [9], per ottenere il problema principale (MP);

2. Effettuiamo una restrizione del problema (RMP) che considera solo un sottoinsieme $S \subseteq \Omega$ di colonne;

3. Risolviamo il Pricing Problem per trovare quali colonne possono rientrare nella base e ridurre la funzione obiettivo: bisogna cioè trovare una colonna con un costo ridotto negativo. Tale sotto problema deve essere risolto solamente per provare che una soluzione ottimale di RMP è anche una soluzione ottimale di MP;

   - Se esiste una colonna con costo negativo, la aggiungo all'insieme S di colonne di RMP e torno al punto 2;
   - Se tale cammino non esiste, proseguire.

4. Studio la soluzione trovata: se è intera allora abbiamo terminato. Altrimenti proseguire;

5. Applico l'algoritmo Branch-and-Bound (descritto nella Sezione 4.1) su tale soluzione e torno al punto 3.

5.4 Algoritmo Branch-and-Price-and-Cut

Descriviamo ora uno dei metodi risolutivi più completi per il GVRP, l'algoritmo Branch-and-Price-and-Cut ($B\&P\&C$).
Si tratta di una tecnica che si ottiene aggiungendo una serie di tagli in varie fasi dell'algoritmo Branch-and-Price, e che può migliorarne in maniera significativa le prestazioni.
Nel lavoro di Desaulniers et al. [13] del 2016, questo algoritmo viene presentato per risolvere l'*Electric Vehicle Routing Problem with Time Windows* (EVRPTW) e sue quattro varianti dovute alle possibili limitazioni sulla ricarica dei veicoli. In tale lavoro vengono presentate diverse varianti per l'algoritmo di etichettatura, utilizzato per risolvere il pricing problem, in base alla possibilità o meno di ricarica parziale. La ramificazione attuata invece è unica per tutte le varianti, e aggiunge due possibili variabili rispetto ai soliti numeri di veicoli e archi: si tratta del numero totale di ricariche, in generale e rispetto a una precisa stazione di ricarica.
Come è possibile intuire dal nome, questo algoritmo si può dividere in tre fasi in base a che algoritmo agisce in quel momento: generazione di colonne, piani di taglio o Branch-and-Bound.
Basandoci sullo studio del lavoro citato e di altri lavori (Desrosiers et al. [16], Chu et al. [45]), e su quello delle sezioni precedenti, possiamo elaborare l'algoritmo Branch-Price-and-Cut. Si parte da una formulazione set partitioning, come avveniva per il Branch-and-Price. Vengono quindi definiti il problema principale e il problema principale ristretto, cercando la soluzione di quest'ultimo tramite un algoritmo di Generazione di Colonne per ottenere quindi una soluzione primale e una duale.
Il Pricing Problem ottenuto dalla soluzione duale permette ora di identificare alcune colonne con costo ridotto negativo (tramite algoritmi come quello di etichettatura bidirezionale) che non appartengano ancora a RMP, aggiungendole e iterando.
A questo punto si inseriscono all'interno del problema principale alcuni tagli per ridurre numero di nodi necessari per l'albero nell'algoritmo Branch-and-Bound.
Infine si applica proprio tale algoritmo, ramificando sulle soluzioni frazionarie trovate fintanto che non si trova una soluzione intera ammissibile per il problema principale.
Formalmente l'algoritmo diventa:

1. *Inizializzazione:* Formuliamo il rilassamento lineare del GVRP con un sottoinsieme iniziale di S;

2. *Generazione di Colonne*: Si risolve il pricing problem (attraverso un algoritmo label) aggiungendo nuove colonne al RMP fintanto che non si trovano cammini migliori.

3. *Piani di taglio*: viene risolto il rilassamento delle cammini trovati. Identificare e aggiungere ulteriori tagli per migliorare la formulazione. Continuare fintanto che non si trovano tagli.

4. *Ramificazione*: se la soluzione non è intera, selezionare una variabile e ramificare, creando nuovi problemi.

5. *Iterazioni*: continuare il processo di ramificazione fintanto che non si esauriscono i nodi attivi o non vengono soddisfatti i criteri di arresto.

Capitolo 6

Valutazione delle performance

In questo capitolo riportiamo le performance dei diversi metodi per la soluzione di modelli per GVRP, come presentati in letteratura.

6.1 Risultati di algoritmi $B\&C$

Uno dei primi utilizzi del metodo di $B\&C$ si può trovare nel lavoro di Koc et al. [29]: qui viene studiano l'andamento e l'efficacia dell'algoritmo $B\&C$ unito a un algoritmo euristico, il simulated annealing. Come insieme di istanze benchmark, utilizzano quello proposto da Miller-Hooks et al. [18].
Ogni istanza comprende 20 clienti ed è possibile raggrupparle in quattro scenari, ciascuno da 10 problemi:

- S1: i clienti vengono disposti uniformemente con 3 AFS;
- S2: i clienti vengono disposti in gruppi con 3 AFS;
- S3: metà dei problemi sono selezionati da S1, metà da S2. Il numero di AFS diventa 6;
- S4: viene selezionato un problema da S1 e uno da S2, per ciascuno si aumenta di 2 il numero di AFS per ogni problema fino ad arrivare a 10.

L'algoritmo proposto in [29] si rivela essere efficace per la maggior parte dei casi, riuscendo a risolvere 22 istanze su 40 entro il tempo limite di un'ora. Osservando la distribuzione delle istanze risolte, si nota come i problemi dove i clienti sono disposti uniformemente (S1) sono più difficili da risolvere rispetto a quelli dove i clienti sono raggruppati (S2).
Studiando l'andamento dell'algoritmo nei problemi di (S3) e di (S4) invece possiamo osservare come l'aumentare il numero delle AFS faciliti la risoluzione del problema.
Consideriamo ora il lavoro presentato da Gadegaard et al. in [36]: in questo caso, l'articolo tratta dello studio di un *Mixed Fleet Green Vehicle Routing Problem*, analizzando però anche il caso in cui la flotta sia omogenea. Sono questi i dati che noi andremo a studiare ora.

L'insieme di problemi utilizzato questa volta corrisponde a quello presentato da Schneider et al. [40] e ispirato al lavoro di Solomon et al. [41]: consiste di 56 istanze di grandi dimensioni (ciascuna contenente 100 clienti e 21 stazioni di ricarica) raggruppate in classi a differenza di come vengono distribuiti geograficamente i clienti: raggruppati, casuali, o misti tra i due. Da queste istanze di grandi dimensioni vengono poi derivate una serie di istanze più piccoli (da 50, 25, 15, 10 e 5 clienti), ottenendo un totale di 204 istanze.
Si comincia poi ad osservare prima l'andamento nei casi di numeri inferiori di clienti (da 5 a 25 clienti) confrontando i risultati ottenuti con quelli prodotti da una formulazione MILP risolta direttamente da un risolutore computazionale CPLEX. Si nota come l'algoritmo $B\&C$ riesca a risolvere circa il 96% di tutti i problemi, a differenza di CPLEX che ne risolve solamente il 69%
Si riesce pertanto a constatare come l'algoritmo $B\&C$ specializzato sia più efficace del modello MILP risolto con solver standard.
Per quanto riguarda i problemi con un numero più elevato di clienti, i risultati ottenuti mostrano come l'algoritmo $B\&C$ continui a risolvere la maggior parte delle istanze entro il tempo limite. All'interno del loro algoritmo di $B\&C$ i tagli utilizzati vengono suddivisi in tre tipi e viene poi analizzato quanto spesso ciascuno di questi è necessario: i risultati mostrano come a differenza del numero di clienti del problema vari l'importanza del tipo di taglio.

6.2 Risultati di algoritmi $B\&P$ e $B\&P\&C$

Per quanto riguarda lo studio dell'algoritmo $B\&P$ invece consideriamo l'analisi proposta da Hart et al. [27], in cui si confronta l'algoritmo $B\&P$ con l'algoritmo euristico Adaptive Large Neighbourhood Search (ALNS) come soluzione di un *Vehicle Routing Problem with Time Windows and Recharging Station*. L'insieme di problemi considerato corrisponde all'insieme proposta da Schneider et al. [40] già illustrato. In particolare, ci limiteremo ad analizzare i casi in cui il numero di clienti siano un numero ridotto. I risultati ottenuti mostrano come l'algoritmo $B\&P$ riesca a trovare soluzioni ottimali in pochi secondi per ogni istanza, eccetto una in cui riesce comunque a trovare una soluzione ottimale entro i venti minuti. L'algoritmo ALNS riesce anch'esso a trovare soluzioni ottimali o molto vicine all'ottimo per ogni caso, ma impiegandoci un tempo maggiore in quasi tutte le istanze, anche se resta sempre sotto il minuto.
Per quanto riguarda l'algoritmo Branch-and-price-and-cut studiamo invece i risultati in Desaulniers et al. (2016) [13], dove vengono analizzata l'efficienza del $B\&P\&C$ per la soluzione di un EVRPTW, considerando i vari sotto problemi che si vengono a formare quando si studia la possibilità di fermarsi più volte o no presso le stazioni di servizio, e i casi in cui la ricarica può avvenire anche solo parzialmente. Per il nostro studio noi considereremo i casi in cui la ricarica deve essere totale ed è possibile fermarsi più volte. Per la soluzione del pricing problem propongono sia un algoritmo di etichettatura bidirezionale che uno mono-direzionale, studiandone i vari comportamenti.
I risultati ottenuti mostrano che, in entrambi i casi, l'algoritmo riesce a risolvere in maniera ottimale il problema per la maggior parte dei problemi, contando 63 istanze risolte per

il mono-direzionale su 87, e 68 per il bidirezionale. Si nota però che nonostante entrambi risolvano all'incirca lo stesso numero di istanze all'interno del tempo limite, il bidirezionale riesce a risolvere con una velocità nettamente maggiore, arrivando quasi a dimezzare le tempistiche. Le istanze che non sono state risolte inoltre sono quasi tutti facenti parte del gruppo di 100 clienti.

Capitolo 7

Conclusioni

In questa tesi abbiamo presentato un'analisi di alcune soluzioni esatte per il Green Vehicle Routing Problem formulato da Miller-Hooks et al. [18].
Nel Capitolo 2 proponiamo un modello matematico per il problema, proponendo diversi formulazioni e soluzioni per come gestire la presenza delle stazioni di ricarica e la possibilità di visitarle più volte da diversi veicoli all'interno della stessa soluzione, formulazione proposte da Koc et al. [29] e Bruglieri et al. [6].
In seguito esaminiamo gli algoritmi esatti più comunemente usati come soluzione del GVRP, evidenziandone le componenti e presentando alcuni aspetti implementativi.
Cominciando con il metodo dei piani di taglio, ne esplicitiamo il funzionamento, elencando anche alcuni esempi di tagli che è possibile utilizzare. In seguito studiamo l'algoritmo del Branch-and-cut, partendo dal suo algoritmo base, il branch-and-bound, ed elencando alcune idee per l'implementazione di questo. Vengono inoltre presentati alcuni lavori in cui questo algoritmo è utilizzato per risolvere varianti del GVRP.
Successivamente presentiamo l'algoritmo del Branch-and-Price, enunciando anche un algoritmo per risolvere il problema di pricing , ovvero una variante dell'algoritmo di etichettatura di Irnich et al. [28], uno degli algoritmi più utilizzati per risolvere suddetto problema.
Infine passiamo in rassegna alcuni risultati computazionali sull'utilizzo di questi metodi ricavati dai lavori sul GVRP e da alcune sue varianti. Possiamo concludere che gli algoritmi esatti rappresentano un approccio praticabile per i problemi di GVRP, fintanto che questi hanno dimensioni ridotte. In questi casi permettono di trovare soluzioni precise velocemente in maniera più efficiente di altri algoritmi euristici, ma questa efficienza comincia a venir meno rapidamente quando il numero di clienti comincia ad aumentare, arrivando in molti casi e non riuscire a produrre una soluzione all'interno del limite di tempo.

Bibliografia

[1] Michel Balinski e Richard E. Quandt. On an Integer Program for a Delivery Problem. *Operations Research* 12 (1964), pp. 300–304. URL: https://api.semanticscholar.org/CorpusID:123265430.

[2] Jonathan Bard, Liu Huang, Moshe Dror e Patrick Jaillet. A branch and cut algorithm for the VRP with satellite facilities. *Institute of Industrial and Systems Engineers Transactions* 30 (1998), pp. 821–834. DOI: 10.1023/A:1007500200749.

[3] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh e Pamela H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research* 46.3 (1998), pp. 316–329. DOI: 10.1287/opre.46.3.316.

[4] Martin Behnke, Thomas Kirschstein e Christian Bierwirth. A column generation approach for an emission-oriented vehicle routing problem on a multigraph. *European Journal of Operational Research* 288.3 (2021), pp. 794–809. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2020.06.035. URL: https://www.sciencedirect.com/science/article/pii/S0377221720305786.

[5] Dario Bezzi, Alberto Ceselli e Giovanni Righini. A route-based algorithm for the electric vehicle routing problem with multiple technologies. *Transportation Research Part C: Emerging Technologies* 157 (2023). ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2023.104374. URL: https://www.sciencedirect.com/science/article/pii/S0968090X23003649.

[6] M. Bruglieri, S. Mancini e O. Pisacane. More efficient formulations and valid inequalities for the Green Vehicle Routing Problem. *Transportation Research Part C: Emerging Technologies* 105 (2019), pp. 283–296. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2019.05.040. URL: https://www.sciencedirect.com/science/article/pii/S0968090X1930779X.

[7] G. Clarke e J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research* 12.4 (1964), pp. 568–581. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/167703.

[8] Said Dabia, Emrah Demir e Tom Van Woensel. An Exact Approach for a Variant of the Pollution-Routing Problem. *Transportation Science* (2015). DOI: 10.1287/trsc.2015.0651.

[9] G. Dantzig, R. Fulkerson e S. Johnson. Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America* 2.4 (1954), pp. 393–410. ISSN: 00963984. URL: http://www.jstor.org/stable/166695.

[10] G. B. Dantzig e J. H. Ramser. The Truck Dispatching Problem. *Management Science* 6.1 (1959), pp. 80–91. ISSN: 00251909, 15265501. URL: http://www.jstor.org/stable/2627477.

[11] George B. Dantzig e Philip Wolfe. Decomposition Principle for Linear Programs. *Operations Research* 8.1 (1960), pp. 101–111. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/167547.

[12] Luigi De Giovanni, Marco Di Summa e Giacomo Zambelli. Metodi Risolutivi per la Programmazione Lineare Intera. Università di Padova (2015). URL: https://www.math.unipd.it/~luigi/courses/metmodoc1516/m06.pli.pdf.

[13] Guy Desaulniers, Fausto Errico, Stefan Irnich e Michael Schneider. Exact Algorithms for Electric Vehicle-Routing Problems with Time Windows. *Operations Research* 64 (2016). DOI: 10.1287/opre.2016.1535.

[14] Guy Desaulniers, Franois Lessard e Ahmed Hadjar. Tabu Search, Generalized k-Path Inequalities, and Partial Elementarity for the Vehicle Routing Problem with Time Windows. *Transportation Science* 42 (2008). DOI: 10.1287/trsc.1070.0223.

[15] Martin Desrochers, Jacques Desrosiers e Marius Solomon. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research* 40 (1992), pp. 342–354. DOI: 10.1287/opre.40.2.342.

[16] Jacques Desrosiers e Marco E. Lübbecke. Branch-Price-and-Cut Algorithms. *Wiley Encyclopedia of Operations Research and Management Science*. Wiley, John e Sons, Ltd, 2011. ISBN: 9780470400531. DOI: https://doi.org/10.1002/9780470400531.eorms0118. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470400531.eorms0118. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470400531.eorms0118.

[17] Marco Di Summa, Luigi De Giovanni e Francesco Rinaldi. Operations Research: Course Notes Master's Degree in Mathematics. Università di Padova (2022).

[18] Sevgi Erdoğan e Elise Miller-Hooks. A Green Vehicle Routing Problem. *Transportation Research Part E: Logistics and Transportation Review* 48.1 (2012). Selected Papers from the 19th International Symposium on Transportation and Traffic Theory, pp. 100–114. ISSN: 1366-5545. DOI: https://doi.org/10.1016/j.tre.2011.08.001. URL: https://www.sciencedirect.com/science/article/pii/S1366554511001062.

[19] Seyed Sajjad Fazeli, Saravanan Venkatachalam e Jonathon M. Smereka. Efficient algorithms for electric vehicles' min-max routing problem. *Sustainable Operations and Computers* 5 (2024), pp. 15–28. ISSN: 2666-4127. DOI: https://doi.org/10.1016/j.susoc.2023.07.002. URL: https://www.sciencedirect.com/science/article/pii/S2666412723000107.

[20] Dominique Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR* 8 (2010), pp. 407–424. DOI: 10.1007/s10288-010-0130-z.

[21] Dominique Feillet, Pierre Dejax, Michel Gendreau e Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks* 44 (2004), pp. 216–229. DOI: 10.1002/net.20033.

[22] Wenqi Gao, Zhixing Luo e Houcai Shen. A branch-and-price-and-cut algorithm for time-dependent pollution routing problem. *Transportation Research Part C: Emerging Technologies* 156 (2023), p. 104339. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2023.104339. URL: https://www.sciencedirect.com/science/article/pii/S0968090X23003297.

[23] Annisa Kesy Garside, Robiah Ahmad e Mohd Nabil Bin Muhtazaruddin. A recent review of solution approaches for green vehicle routing problem and its variants. *Operations Research Perspectives* 12 (2024), p. 100303. ISSN: 2214-7160. DOI: https://doi.org/10.1016/j.orp.2024.100303. URL: https://www.sciencedirect.com/science/article/pii/S2214716024000071.

[24] Paul C. Gilmore e Ralph E. Gomory. A Linear Programming Approach to the Cutting Stock Problem I. *Operation Research* 9 (1961). DOI: 10.1287/opre.9.6.849.

[25] Ralph E. Gomory. *An algorithm for the mixed integer problem.* RAND Corporation, 1960.

[26] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society* 64.5 (1958), pp. 275–278.

[27] Gerhard Hiermann, Jakob Puchinger, Stefan Ropke e Richard Hartl. The Electric Fleet Size and Mix Vehicle Routing Problem with Time Windows and Recharging Stations. *European Journal of Operational Research* Accepted (2016). DOI: 10.1016/j.ejor.2016.01.038.

[28] Stefan Irnich e Guy Desaulniers. Shortest Path Problems with Resource Constraints. 2006, pp. 33–65. ISBN: 0-387-25485-4. DOI: 10.1007/0-387-25486-2_2.

[29] Çağrı Koç e Ismail Karaoglan. The green vehicle routing problem: A heuristic based exact solution approach. *Applied Soft Computing* 39 (2016), pp. 154–164. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2015.10.064. URL: https://www.sciencedirect.com/science/article/pii/S1568494615007085.

[30] Ailsa Land e Alison Doig. An Automatic Method for Solving Discrete Programming Problems. *Econometrica* 28 (1960), pp. 105–132. DOI: 10.1007/978-3-540-68279-0_5.

[31] Gilbert Laporte, Yves Nobert e Paul Pelletier. Hamiltonian location problems. *European Journal of Operational Research* 12.1 (1983), pp. 82–89. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(83)90182-0. URL: https://www.sciencedirect.com/science/article/pii/0377221783901820.

[32] Yiming Liu, Baldacci Roberto, Jianwen Zhou, Yang Yu, Yu Zhang e Wei Sun. Efficient feasibility checks and an adaptive large neighborhood search algorithm for the time-dependent green vehicle routing problem with time windows. *European Journal of Operational Research* 310.1 (2023), pp. 133–155. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2023.02.028. URL: https://www.sciencedirect.com/science/article/pii/S0377221723001637.

[33] Hongyuan Luo, Mahjoub Dridi e Olivier Grunder. A branch-price-and-cut algorithm for a time-dependent green vehicle routing problem with the consideration of traffic congestion. *Computers & Industrial Engineering* 177 (2023), p. 109093. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2023.109093. URL: https://www.sciencedirect.com/science/article/pii/S0360835223001171.

[34] Simona Mancini e Ornella Pisacane. A more efficient cutting planes approach for the green vehicle routing problem with capacitated alternative fuel stations. *Optimization Letters* 15 (2021), pp. 1–17. DOI: 10.1007/s11590-021-01714-3.

[35] Hugues Marchand, Alexander Martin, Robert Weismantel e Laurence Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics* 123.1 (2002), pp. 397–446. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(01)00348-1. URL: https://www.sciencedirect.com/science/article/pii/S0166218X01003481.

[36] Jesper B. Mikkelsen, Sune L. Gadegaard e Jens Lysgaard. A Branch-and-Cut Algorithm for the Mixed Fleet Green Vehicle Routing Problem (2023). URL: https://pure.au.dk/ws/portalfiles/portal/370772459/A_Branch-and-Cut_Algorithm_for_the_Mixed_Fleet_Green_Vehicle_Routing_Problem.pdf.

[37] John E. Mitchell. Branch-and-Cut Algorithms for Combinatorial Optimization Problems (1988). URL: https://api.semanticscholar.org/CorpusID:14530622.

[38] Manfred Padberg e Giovanni Rinaldi. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Review* 33.1 (1991), pp. 60–100. ISSN: 00361445, 10957200. URL: http://www.jstor.org/stable/2030652.

[39] Neda Rezaei, Sadoullah Ebrahimnejad, Amirhossein Moosavi e Adel Nikfarjam. A green vehicle routing problem with time windows considering the heterogeneous fleet of vehicles: Two metaheuristic algorithms. *European Journal of Industrial Engineering* (2019). DOI: 10.1504/EJIE.2019.10022249.

[40] Michael Schneider, Andreas Stenger e Dominik Goeke. The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations. *Transportation Science* 48 (2014), pp. 500–520. DOI: 10.1287/trsc.2013.0490.

[41] Marius M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research* 35.2 (1987), pp. 254–265. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/170697.

[42] Timothy Sweda, Irina Dolinskaya e Diego Klabjan. Adaptive Routing and Recharging Policies for Electric Vehicles. *Transportation Science* 51 (2017). DOI: 10.1287/trsc.2016.0724.

[43] Paolo Toth e Daniele Vigo. Exact Solution of the Vehicle Routing Problem (1998). A cura di Thomas G. Crainic e Gilbert Laporte. DOI: 10.1007/978-1-4615-5755-2_1.

[44] Sihan Wang, Cheng Han, Yang Yu, Min Huang, Wei Sun e Ikou Kaku. Reducing Carbon Emissions for the Vehicle Routing Problem by Utilizing Multiple Depots. *Sustainability* 14 (2022), p. 1264. DOI: 10.3390/su14031264.

[45] Yunqiang Yin, Xiaochang Liu, Feng Chu e Dujuan Wang. An exact algorithm for the home health care routing and scheduling with electric vehicles and synergistic-transport mode. *Annals of Operations Research* (2023). DOI: 10.1007/s10479-023-05313-6.

[46] Yang Yu, Sihan Wang, Junwei Wang e Min Huang. A branch-and-price algorithm for the heterogeneous fleet green vehicle routing problem with time windows. *Transportation Research Part B: Methodological* 122 (2019), pp. 511–527. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2019.03.009. URL: https://www.sciencedirect.com/science/article/pii/S0191261518308944.