UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI SCIENZE

Corso di Laurea Magistrale in Informatica

Decision Science

Il Viaggio Epico di Elon Musk: Il TSP come Guida tra i Pianeti

Professore
Massimo Di Francesco

Studenti
Faedda Michele 60/73/65285
Tkachenko Viacheslav 60/73/65286

ANNO ACCADEMICO 2023/2024

Indice

Capitolo 1

Introduzione

Il *Problema del Commesso Viaggiatore* (*Travelling Salesman Problem*, TSP) è uno dei classici problemi di ottimizzazione che richiede, nella sua versione originale, di determinare il percorso con costo minimo che un venditore deve seguire per visitare un insieme di città esattamente una volta, tornando infine alla città di partenza. Il TSP ha applicazioni in vari settori, come la logistica, la pianificazione dei tragitti e la progettazione di circuiti stampati.

Nell'ambito di questa tesina, applichiamo TSP in maniera giocosa al caso in cui Elon Musk, l'imprenditore visionario di SpaceX, desidera visitare tutti i pianeti del nostro sistema solare. Ogni percorso tra i pianeti ha un costo associato che può differire anche scambiando il punto di partenza e il punto di arrivo tra un nodo e l'altro. L'obiettivo è quello di trovare il percorso complessivo che consenta a Elon Musk di visitare tutti i pianeti riducendo il costo il più possibile.

Questa tesina si propone di esaminare il problema del TSP nel contesto dell'ottimizzazione lineare e di presentare due formulazioni del problema che comportano entrambe un numero esponenziale di vincoli. Inoltre, verrà introdotto un algoritmo di branching che permette di ridurre il numero dei vincoli, rendendo più gestibile la risoluzione del problema. Applicheremo queste formulazioni e l'algoritmo di branching al caso di studio di Elon Musk per determinare il percorso con costo minimo per la sua avventura spaziale nel sistema solare.

1.1 Sommario

In questa tesina partiremo introducendo le nozioni base in 2 per poi passare a descrivere le diverse formulazioni 3 del problema. Queste ci porteranno ad affrontare alcune sfide computazionali che cerchiamo di risolvere in 4 con due algoritmi: uno descritto nel [1] e l'altro che sfrutta le idee del precedente, ma con una modifica.

Capitolo 2

Fondamenti Teorici

Questa sezione è dedicata a ricapitolare in breve le nozioni teoriche dell'ottimizzazione lineare che verranno applicate al problema del TSP. Inoltre, saranno fatte delle prime considerazioni sul problema del TSP con alcuni concetti sulla modalità di rappresentazione tramite grafi. Le formulazioni usate per la risoluzione del problema verranno trattate nel capitolo successivo.

2.1 Ottimizzazione lineare

L'ottimizzazione lineare è una branca dell'ottimizzazione matematica che si occupa di risolvere problemi di massimizzazione o minimizzazione di una funzione lineare soggetta a un insieme di vincoli lineari. Un problema di ottimizzazione lineare può essere espresso nel seguente formato:

$$
\begin{aligned}
&\text{minimize} \quad c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
&\text{subject to} \\
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \geq b_1 \\
&a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \geq b_2 \\
&\quad \vdots \\
&a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \geq b_m \\
&x_1 \geq 0, \quad x_2 \geq 0, \quad \ldots, \quad x_n \geq 0
\end{aligned}
$$

dove $c_1, c_2, ..., c_n$ sono i coefficienti dei costi per le variabili decisionali $x_1, x_2, ..., x_n$. $a_{1_1}, a_{1_2}, ..., a_{m_n}$ sono i coefficienti dei vincoli tecnologici, mentre $b_1, b_2, ..., b_m$ sono i termini noti.

Durante il corso abbiamo visto che certi problemi di PL richiedono che le variabili decisionali assumano valori interi. In tal caso, lo spazio ammissibile è l'insieme dei punti a coordinate intere. Nel caso del Problema del commesso viaggiatore si

ha una situazione simile in quanto nelle formulazioni che tratteremo le variabili decisionali assumono valori booleani.

2.2 Introduzione ai Grafi

Un grafo è una struttura dati che rappresenta un insieme di oggetti, chiamati nodi o vertici, collegati tra loro da archi. I grafi sono usati per modellare e analizzare relazioni tra elementi di un insieme e sono ampiamente utilizzati in svariati campi. Spesso vengono rappresentati come *G = (V, E)* dove V è l'insieme dei nodi ed E è una famiglia di coppie di elementi di V che rappresentano i collegamenti tra i nodi.

2.2.1 Tipologie

Ci sono diverse tipologie di grafi, tra cui grafi non orientati, grafi orientati e grafi pesati [2].

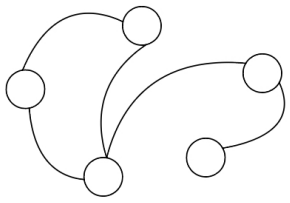

Figura 2.1: *Esempio grafo non orientato*

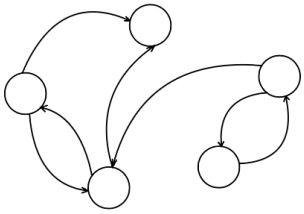

Figura 2.2: *Esempio grafo orientato*

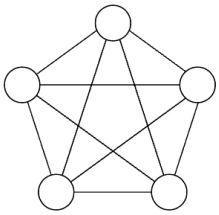

Figura 2.3: *Esempio grafo completo non orientato*

- nei **Grafi non Orientati** gli archi non hanno una direzione specifica; pertanto, una connessione tra due nodi è rappresentata da un arco che collega i due nodi senza una distinzione tra nodo di partenza e nodo di arrivo.
- nei **Grafi Orientati** gli archi hanno una direzione specifica; quindi, una connessione tra due nodi è rappresentata da un arco diretto che ha origine da un nodo di partenza e finisce in un nodo di arrivo. Ad esempio, un grafo orientato può rappresentare una rete di collegamenti ipertestuali tra le pagine web, in cui i nodi sono le pagine e gli archi indicano i collegamenti da una pagina all'altra.
- nei **Grafi Completi** si ha un grafo non orientato in cui vi è un arco tra ogni coppia di nodi. In altre parole, ogni nodo è direttamente collegato a tutti gli altri nodi del grafo.

- nei **Grafi Pesati** a ciascun arco viene assegnato un valore o un peso che rappresenta qualche tipo di attributo o costo associato all'arco. Ad esempio, un grafo pesato può rappresentare una rete stradale in cui i nodi sono intersezioni, gli archi sono le strade e i pesi associati indicano il tempo di percorrenza di una determinata strada.

2.2.2 Cammini

Ci sono varie nozioni riguardanti i grafi. Per la nostra tesina e la successiva analisi del TSP, riteniamo utile introdurre quelli relativi ai cammini.

Un *cammino* di un grafo *G = (V, E)* è una sequenza di archi consecutivi i quali permettono di collegare due vertici di V visitando una sequenza di vertici intermedi.

Il vertice appartenente ad un grafo si dice *connesso* ad un altro vertice dello stesso grafo semplicemente se esiste un cammino che collega i due vertici.

Se nel cammino il vertice di arrivo è uguale a quello di partenza si ha un *ciclo*.

Un ciclo semplice si dice *Hamiltoniano* se visita una ed una sola volta tutti i vertici del grafo.

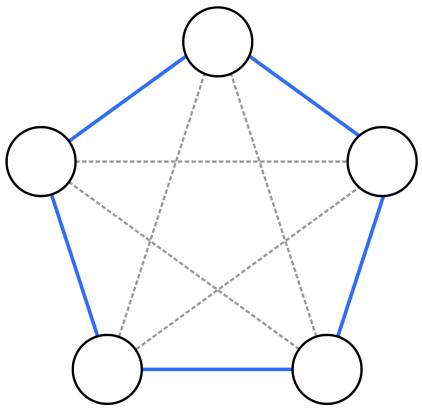

Figura 2.4: *Esempio Cammino Hamiltoniano*

Possiamo ora riformulare il problema del TSP come: dato un grafo *G = (V, E)* con dei pesi associati ad ogni arco, si cerca di individuare un ciclo hamiltoniano di costo minimo.

Possiamo fare una prima considerazione per il *Problema del Commesso Viaggiatore* in relazione al numero dei cicli che si possono avere.

2.2.3 Numero di cicli

In un grafo completo con n nodi, ogni permutazione dei nodi forma un ciclo hamiltoniano. Poiché ci sono n nodi, ci sono n! permutazioni possibili. Tuttavia, poiché un ciclo hamiltoniano può essere percorso in entrambe le direzioni, ogni ciclo hamiltoniano viene contato due volte (una volta in senso orario e una volta in senso antiorario). Siccome abbiamo un numero fattoriale di potenziali soluzioni da esplorare, enumerare tutte le potenziali soluzioni sembra proprio una cattiva idea.

Capitolo 3

Formulazioni

In questa sezione, presenteremo due formulazioni del problema del TSP con un numero esponenziale di vincoli: la formulazione di Cut Set e Subtour Elimination. Esamineremo quali problemi portano tali formulazioni, mentre nel capitolo successivo si esaminerà un metodo e una sua variazione per gestire queste problematiche.

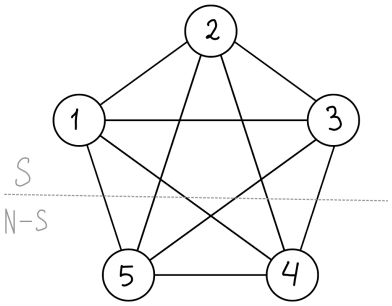

Figura 3.1: *Esempio di taglio*

Le successive due formulazioni si basano sull'idea di utilizzare un insieme di taglio per rappresentare alcuni vincoli del TSP. Un taglio è un insieme di archi che separa il grafo *G = (N, E)* in due parti disgiunte $S \subset N$ ed *N - S*.

Nelle formulazioni successive useremmo $\delta(S)$ per indicare un insieme di archi che hanno un nodo $i \in S$ e un nodo $j \in N - S$

$$\delta(S) = \{(i, j) \in E : i \in S, j \notin S\}$$

ed $E(S)$ per indicare un insieme di archi che hanno entrambi i nodi $i, j \in S$

$$E(S) = \{(i, j) \in E : i \in S, j \in S\}$$

In figura 3.1 $\delta(S) = \{(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)\}$ ed $E(S) = \{(1, 2), (1, 3), (2, 3)\}$.

3.1 Cut Set

Consideriamo un grafo completo non orientato in cui i nodi rappresentano le città e gli archi rappresentano i percorsi tra le città. Definiamo le variabili decisionali binarie x_{i_j} per ogni arco *(i, j)*, dove $x_{i_j} = 1$ se l'arco fa parte del percorso e $x_{i_j} = 0$ altrimenti.

$$x_{i_j} = \begin{cases} 1 & \text{sel'arco } (i,j) \in E \text{ fa parte del percorso ottimo} \\ 0 & \text{altrimenti} \end{cases}$$

Inizialmente, visto che vogliamo visitare ogni nodo una volta sola, imponiamo che ognuno di questi venga "toccato" esattamente da due archi del percorso ottimo.

$$\sum_{a \in \delta(i)} x_a = 2, \quad \forall i \in N$$

La corrente formulazione però può portare alla individuazione di un percorso ottimo con più di un ciclo (Figura 3.2).

La formulazione di Cut Set impone un vincolo per ogni possibile insieme di taglio che impedisce la formazione di cicli subtour. La formulazione completa del problema diventa quindi la seguente.

$$\text{minimize} \quad \sum_{a \in E} c_a x_a$$

$$\sum_{a \in \delta(i)} x_a = 2, \quad \forall i \in N$$

$$\sum_{a \in \delta(S)} x_a \geq 2, \quad \forall S \subset N, S \neq \emptyset, S \neq N$$

$$x_a \in \{0, 1\}, \quad \forall a \in E$$

Con i nuovi vincoli imponiamo che per qualunque partizione di N in S ed N - S, il ciclo deve avere almeno 2 archi tra i nodi di S e nodi di N - S.

3.2 Subtour Elimination

Partendo dalle stesse idee della formulazione Cut Set possiamo formulare il problema del TSP anche nel seguente modo.

Si hanno gli stessi vincoli per imporre che un nodo venga visitato solo una volta, mentre cambiano i vincoli per l'eliminazione dei cicli multipli. Riportiamo

quindi solo questi vincoli in quanto la restante parte della formulazione è identica a quella di Cut Set.

$$\sum_{a \in E(S)} x_a \leq |S| - 1, \quad \forall S \subset N, S \neq \emptyset, S \neq N$$

Imponiamo quindi che per ogni sottoinsieme di nodi S diverso dall'insieme N e dall'insieme vuoto, si selezionino al più $|S|$-*1* archi di *E(S)*.

Modificando gli ultimi vincoli delle due formulazioni in $0 \leq x_a \leq 1$ possiamo definire il rilassamento continuo delle due formulazini con gli spazi ammissibili P_{cutset} e P_{sub}. Si può dimostrare che $P_{cutset} = P_{sub}$, cioè che le due formulazioni siano ugualmente forti.

3.3 Elon Musk dimentica il teletrasporto

Risolvendo il TSP con le due formulazioni descritte nelle precedenti, si individua un ciclo hamiltoniano di costo minimo. La cattiva notizia sta nel numero di vincoli aggiunti per arrivare ad una soluzione connessa (quindi con un solo ciclo).

In particolare, mentre abbiamo un numero polinomiale di vincoli che impongono che un nodo venga visitato solo una volta, i vincoli per ottenere una soluzione connessa sono $O(2^{|N|})$.

Elon quindi si imbatte in un problema, le due formulazioni non gli permettono di ottenere la soluzione in modo efficiente. Vorrebbe rimuovere i vincoli di connessione, ma avendo dimenticato in laboratorio il teletrasporto deve cercare una soluzione alternativa.

Nel capitolo successivo spieghiamo l'algoritmo che permette di gestire questo problema riducendolo al Problema di Assegnamento ed aggiungendo i vincoli man mano.

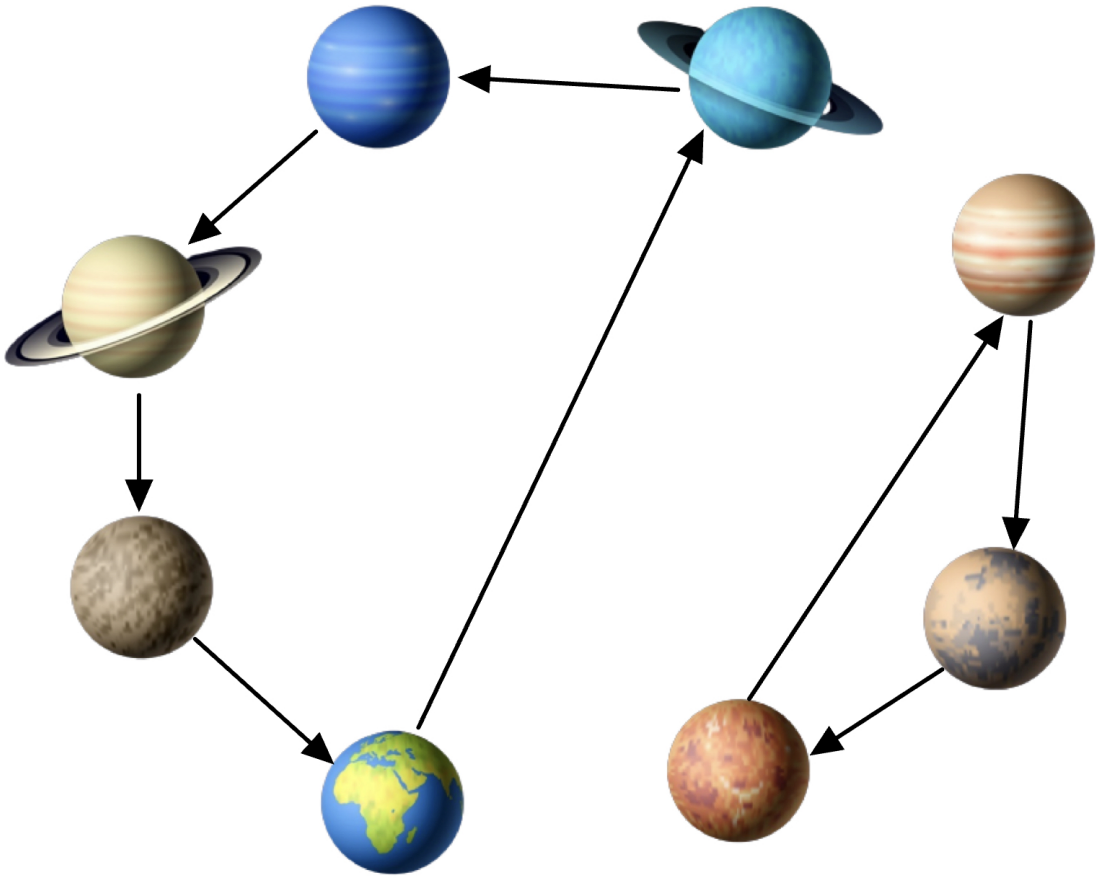

Figura 3.2: *Esempio soluzione disconnessa*

Capitolo 4

Branch & Bound

Come anticipato nel capitolo precedente possiamo usare due formulazioni (Cut Set o Subtour elimination) per risolvere il TSP. Il nostro problema però, sta nel avere un numero esponenziale di vincoli per garantire la connettività della soluzione. In questo capitolo descriviamo un modo per risolvere questo problema che viene anche descritto nel libro [1]. Forniamo inoltre una nostra variazione dell'algoritmo derivante dall'analisi dell'efficienza di quello descritto nel libro.

4.1 Branch & Bound totale

In questa sezione, presenteremo un algoritmo di branching che permette di ridurre il numero dei vincoli nel problema del TSP. L'algoritmo di branching è una tecnica di risoluzione che suddivide il problema originale in sottoproblemi più piccoli, riducendo in molti casi il costo della risoluzione. Nella nostra ambientazione del problema usiamo un grafo completo orientato quindi di seguito presentiamo brevemente la formulazione completa.

Con 4.a e 4.b ci assicuriamo che per ogni nodo ci sia esattamente un arco entrante e un arco uscente. I vincoli 4.c ci garantiscono la connettività della soluzione in quanto impongono almeno un possibile cammino da un nodo ad ogni altro nodo e il loro numero è $O(2^{|N|})$.

$$x_{i_j} = \begin{cases} 1 & \text{sel'arco } (i,j) \in E \text{ fa parte del percorso ottimo} \\ 0 & \text{altrimenti} \end{cases}$$

$$\text{minimize} \quad \sum_{i=1}^{|N|} \sum_{j=1}^{|N|} c_{i_j} x_{i_j}$$

$$\sum_{i=1}^{|N|} x_{i_j} = 1, \quad i = 1, \ldots, |N| \tag{4.a}$$

$$\sum_{j=1}^{|N|} x_{i_j} = 1, \quad j = 1, \ldots, |N| \tag{4.b}$$

$$\sum_{(i,j)\in\delta^+(S)} x_{i_j} \geq 1, \quad \forall S \subset N, S \neq \emptyset, S \neq N \tag{4.c}$$

$$dove \quad \delta^+(S) = \{(i,j) \in E : i \in S, j \notin S\}$$

$$x_a \in \{0,1\}, \quad \forall a \in E \tag{4.d}$$

4.1.1 Problema di assegnamento

In 3.3 (Elon Musk dimentica il teletrasporto) abbiamo già visto a cosa ci porta la rimozione dei vincoli di connessione. In particolare ci può portare ad una soluzione disconnessa e il problema restante prende il nome di Problema di Assegnamento (PA). La sua matrice dei vincoli è unimodulare e, quindi, la soluzione del rilassamento continuo è intera.

Possiamo procedere dicendo che la soluzione del PA sarà sicuramente un limite inferiore a quella del problema originale in quanto lo spazio ammissibile del TSP è compreso in quello del PA. Se la soluzione del PA produce un solo ciclo, allora sarà la soluzione ottima per il TSP, altrimenti si avrà un insieme di cicli disgiunti in cui ogni nodo è visitato esattamente una volta.

Possiamo quindi pensare di risolvere il TSP usando il Branch and Bound, in cui i limiti inferiori sono calcolati risolvendo dei problemi di assegnamento.

Nel branching si fissano a zero le variabili che possono rendere inammissibile la soluzione per il TSP, quindi quelle con valore 1 della soluzione al PA.

Nella figura 3.2 notiamo che abbiamo due cicli nella soluzione per il PA. Il numero di nodi è 8, quindi creiamo 8 nodi figli e per ogni problema cerchiamo di aprire uno dei archi del percorso proposto dalla soluzione del PA.

Fortunatamente non si avrà la necessità di visitare, e quindi anche costruire, tutto l'albero, in quanto possiamo potare alcune sue parti a priori. In particolare un nodo non genera figli se:

- se la soluzione per il PA è ammissibile per il TSP
- se la soluzione del PA è inammissibile
- se la soluzione del PA è inammissibile per il TSP ed è peggiore della migliore soluzione ammissibile per il TSP finora trovata

Forniamo il pseudocodice dell'algoritmo che abbiamo sviluppato.

Algorithm 1 Branch and Bound for TSP

```
best_solution ← None
best_objective_value ← +∞
stack ← a list with only one model for PA
while stack is not empty do
    current_model ← stack.pop()
    solution ← current.solve()
    if solution is None then continue                ▷ Infeasible solution, backtrack
    end if
    objective_value ← solution.get_objective_value()
    if objective_value ≥ best_objective_value then continue      ▷ Solution is
worse than the best found so far, backtrack
    end if
    if the number of cicles is 1 then      ▷ Found a feasible solution with lower
objective value
        best_solution ← solution
        best_objective_value ← objective_value
    end if
    for (i, j) ∈ (1 . . . N) × (1 . . . N) do
        if i ≠ j and x_ij is an arc in the current solution then
            Create a new model cloning the previous one
            Add constrain x_ij = 0 to the new model
            Push the new model to the stack
        end if
    end for
end while
if best_solution is None then
    raise error "Infeasible"
end if
return best_solution
```

4.2 Branch & Bound con branching parziale

Dopo aver implementato l'algoritmo descritto nella sezione precedente abbiamo eseguito dei test. Pur ottenendo dei buoni risultati per alcuni casi di studio abbiamo notato un degrado considerevole delle prestazioni per alcuni di essi. Dopo aver investigato sulla natura del problema abbiamo concluso che la principale causa era la forte frammentazione dell'albero indotta dal generare un numero |N| di figli per ogni iterazione nella quale veniva eseguito il branching.

Proponiamo quindi un modo alternativo, che sfrutta le stesse idee descritte prima, ma che esegue il branching solo su due variabili decisionali che nella soluzione corrente, non ammissibile del problema viaggiatore, rappresentano due degli archi scelti.

Forniamo inoltre, un'analisi delle prestazioni dei due algoritmi su diverse istanze del problema. Abbiamo scelto di utilizzare 15 nodi e nella tabella 4.1 rappresentiamo il tempo impiegato e il numero dei nodi esplorati. Il numero dei nodi aggiunti può essere visto anche come numero dei vincoli usati per garantire la connessione.

ID	B&B Time (s)	# added nodes	Binary B&B Time (s)	# added nodes
1	13.14	300	0.28	6
2	3.26	75	0.19	4
3	2.064	45	0.65	14
4	2.78	60	0.11	2
5	686.72	11370	5.64	118
6	1.73	30	0.32	4
7	37.71	420	2.47	42
8	12.06	240	1.52	22
9	35.92	585	2.09	38
10	8.58	195	6.31	146

Tabella 4.1: *I due Branch and Bound a confronto*

4.3 Applicazione al caso di studio di Elon Musk

Nella sezione precedente, abbiamo presentato come possiamo gestire il numero esponenziale di vincoli per garantire una soluzione connessa nel TSP tramite un algoritmo di branching. In questa sezione esaminiamo passo passo l'esecuzione dell'algoritmo di branching binario sulla nostra ambientazione del problema.

4.3.1 Definizione del problema

Consideriamo un grafo completo in cui i nodi rappresentano i pianeti del nostro sistema solare e gli archi rappresentano i percorsi tra i pianeti. Ogni arco ha associato un costo. L'obiettivo è determinare il percorso chiuso di costo minimo che permetta a Elon Musk di visitare tutti i pianeti una sola volta, tornando infine al pianeta di partenza.

Il grafo, nel nostro caso è un grafo orientato in quanto il costo può differire scambiando il pianeta di partenza e quello di arrivo. In particolare il costo del viaggio dipende dai seguenti fattori:

- distanza
- numero degli asteroidi sul percorso
- orientazione della navicella rispetto alla direzione della luce (la navicella di Elon vuole sprecare meno energia elettrica possibile acquisendone il più possibile dai raggi solari)

$$costo = w1 * dist\_norm + w2 * num\_ast\_norm + w3 * angolo\_trasformato$$

$$dist\_norm = (distanza - distanza\_min)/(distanza\_max - distanza\_min)$$

$$num\_ast\_norm = (num\_ast - num\_ast_min)/(num\_ast\_max - num\_ast\_min)$$

$$angolo\_trasformato = angolo/180°$$

dove w1, w2 e w3 rappresentano i pesi che diamo a ciascun parametro che influenza il costo finale.

4.3.2 Esecuzione dell'algoritmo

Andiamo a vedere le decisioni che prende l'algoritmo su ciascun nodo dell'albero. Rappresentiamo l'albero in Figura 4.2, nel quale i nodi sono numerati nello stesso ordine nel quale vengono visitati.

- Al nodo 1 come si può vedere nell'immagine 4.1 si ha una soluzione ammissibile per il PA, ma non per il TSP, quindi 117 è un limite inferiore alla soluzione del problema originario e si hanno due nuovi nodi: 2 e 3. Nelle fasi successive andiamo a migliorare sempre di più i limiti inferiori per poi raggiungere la soluzione ottima.
- Lo stesso avviene per i nodi 2 e 3. Hanno una soluzione ammissibile per il problema di PA, ma non per il TSP, quindi generano entrambi due figli.

- Il nodo 4 individua una soluzione ammissibile (rappresentata in figura 4.3) per il problema di TSP, per ora è la soluzione migliore, quindi viene aggiornata. Siccome si ha una soluzione ammissibile per il problema di TSP, il nodo 4 non genera figli.
- Negli altri nodi (5, 6, 7) la soluzione è peggiore di quella che è stata già trovata al nodo 4, quindi non generano figli.

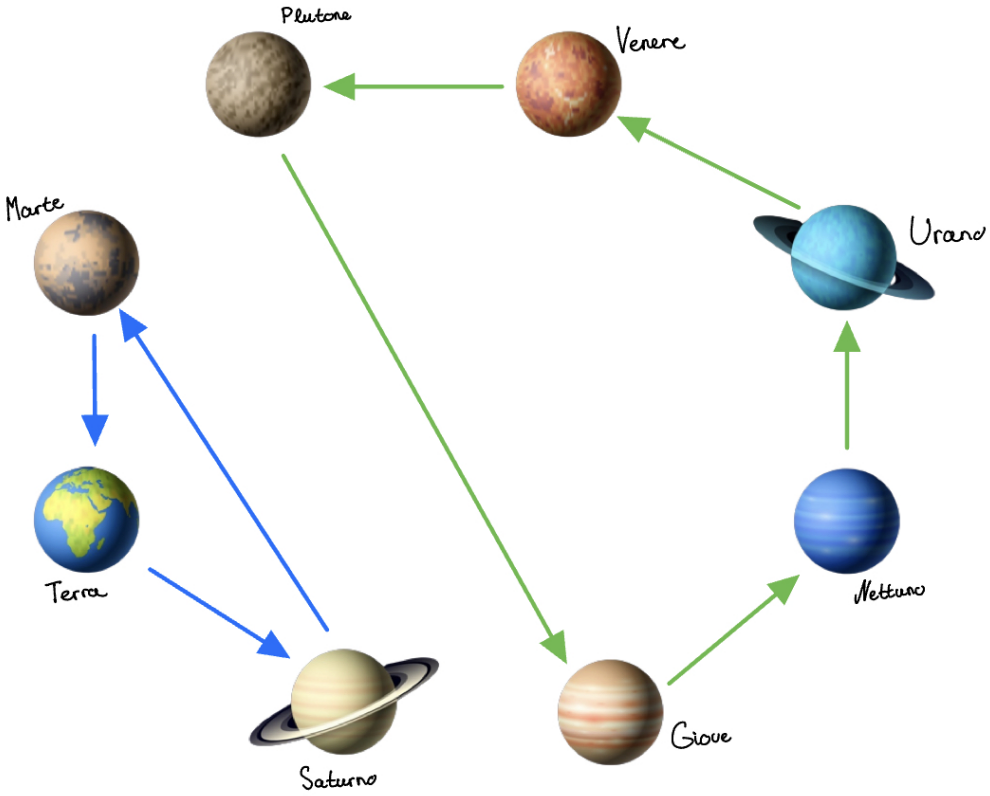

Figura 4.1: *Soluzione al problema di Assegnamento al Nodo 1*

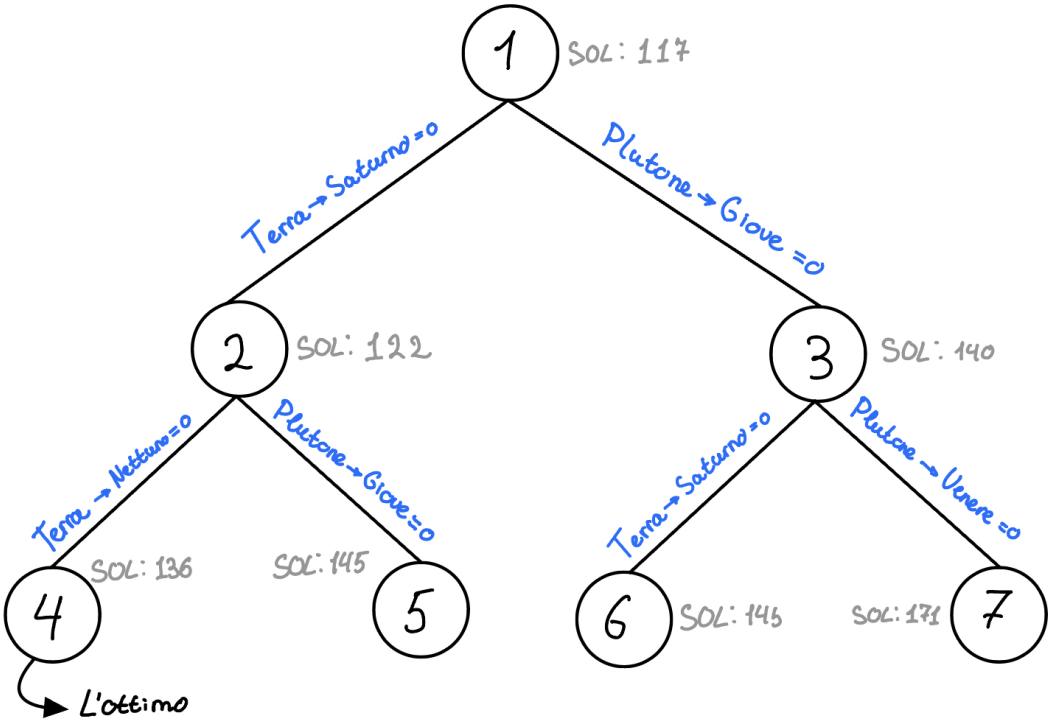

Figura 4.2: *Albero creato dall'algoritmo di branching*

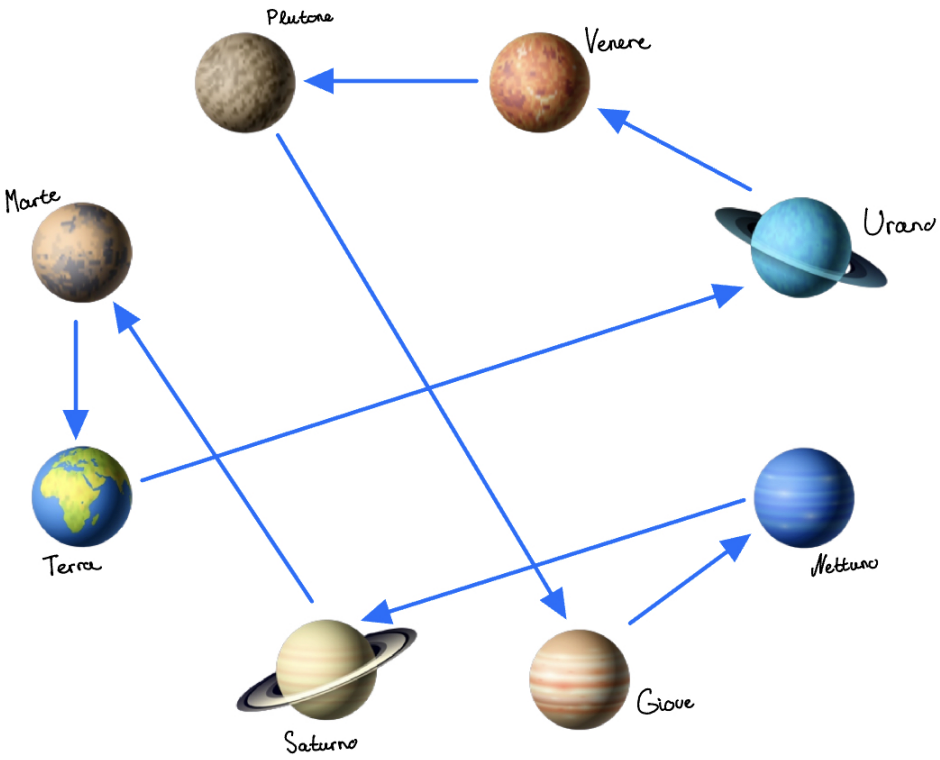

Figura 4.3: *Soluzione finale al problema del TSP*

Capitolo 5

Conclusioni

In questa tesina è stato analizzato il *Problema del Commesso Viaggiatore* (*Travelling Salesman Problem*, TSP), un problema di ottimizzazione che richiede di determinare il percorso di costo minimo per visitare un insieme di nodi esattamente una volta, tornando infine al nodo di partenza.

Sono state analizzate due formulazioni del problema che portano a un numero esponenziale di vincoli per garantire la connettività della soluzione. In seguito è stato implementato l'algoritmo di branching in due versioni che sono state confrontate. Arriviamo alla conclusione che il branching binario nel nostro caso sia più efficiente in quanto statisticamente porta ad un albero più piccolo.

Infine l'algoritmo, nella sua versione binaria, è stato applicato, facendo vedere i diversi passaggi, al caso di studio di Elon Musk per determinare il percorso di costo minimo per la sua avventura spaziale nel sistema solare.

Forniamo di seguito il link alla nostra repository GitHub nella quale può essere trovata la nostra implementazione insieme anche alla matrice che abbiamo usato per simulare l'esecuzione.

Bibliografia

[1] D. Bertsimas and J.N. Tsitsiklis. *Introduction to linear optimization.* Athena Scientific, 1997.

[2] M. Fischetti. *Lezioni Di Ricerca Operativa.* Independently Published, 2018.