Branch and Bound per TSP

Tesina di Ottimizzazione Combinatoria

Leonardo Magliolo
Gianmario Cuccuru

A.A. 2023/2024

Travelling Salesman Problem

- Dato un insieme di città e il relativo costo di transito tra ogni coppia di città, il problema del commesso viaggiatore (Traveling Salesman Problem), o in breve TSP, consiste nel trovare il percorso più economico che consenta di visitare tutte le città una ed una sola volta e tornare infine al punto di partenza.
- Il problema del commesso viaggiatore è però un problema NP-completo e, allo stato dell'arte, non sono noti algoritmi capaci di risolverlo in tempo polinomiale.

Travelling Salesman Problem

- Il Traveling Salesman Problem può essere rappresentato mediante un grafo non orientato $G = (V, E)$ con costi c_{ij} associati agli archi. L'obiettivo è trovare un insieme di archi $C^* \subseteq E$ con il costo minimo possibile.
- Questo insieme C^* deve formare un circuito Hamiltoniano, ovvero un ciclo che attraversi ogni nodo del grafo una ed una sola volta.
- In questo lavoro viene considerata la versione simmetrica del TSP, in cui per ogni coppia di nodi $i, j \in V$ vale $c_{ij} = c_{ji}$.

Formulazione matematica

Una possibile formulazione matematica per il TSP utilizza delle variabili per ogni arco:

$$\forall e \in E, x_e = \begin{cases} 1 & \text{se } e \text{ è nel ciclo} \\ 0 & \text{altrimenti} \end{cases}$$

$$\min \sum_{e \in E} c_e x_e$$

$$(1) \quad s.t. \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V$$

$$(2) \quad \sum_{e \in E} x_e \leq |S| - 1 \quad \forall S \subset V : 3 \leq |S| \leq |V| - 1$$

$$(3) \quad x_e \in \{0, 1\} \quad \forall e \in E$$

Il lavoro di Held and Karp

- Held and Karp (1970) hanno proposto un rilassamento che fornisce dei limiti duali forti. Ad esempio, nella pratica, questi limiti sono utilizzati in Concorde.
- Successivamente è stato proposto sempre da Held and Karp (1971) un algoritmo di branch-and-bound che utilizza questo rilassamento.
- Questo algoritmo sfrutta una struttura chiamata minimum 1-tree che può servire come rilassamento valido per il TSP e ottenere così dei limiti duali.

1-Tree

Sia G(V,E) un grafo completo con costi c_e associati ad ogni arco $e \in E$. Un minimum 1-tree è un minimum spanning tree di $G \setminus \{1\}$ a cui viene aggiunto il nodo 1 insieme alla coppia di archi di costo minimo che lo collegano all'albero.

$$\min \sum_{e \in E} c_e x_e$$

$$(1) \quad s.t. \sum_{e \in \delta(1)} x_e = 2$$

$$(2) \quad \sum_{e \in E} x_e = |V|$$

$$(3) \quad \sum_{\substack{i,j \in S \\ i<j}} x_{i,j} \leq |S| - 1 \quad \forall S \subset V \setminus \{1\} \wedge |S| \geq 3$$

$$(4) \quad x_e \in \{0,1\}$$

1-Tree

- Il problema consiste quindi nel trovare un algoritmo che possa risolvere efficientemente il problema dell'individuazione di un minimum spanning tree sul grafo G.
- Nella formulazione precedente, $\delta(v)$ denota gli archi contenenti il vertice $v \in V$, mentre $c_e \in \mathbb{R}$ rappresenta il costo di un arco $e \in E$.
- Infine, $x_e \in \{0, 1\}$ è la variabile di decisione che indica se l'arco e è incluso o meno nel 1-tree.

1-Tree

- È possibile notare che ogni tour in G è un 1-tree e se un minimum 1-tree è un tour, allora quest'ultimo è una soluzione ottimale per il TSP. Pertanto, ogni minimum 1-tree è un rilassamento valido per il TSP.
- Tuttavia, non è detto che una soluzione di questo programma intero sia un tour. Per farlo, è necessario applicare un nuovo set di vincoli:

$$\sum_{e \in \delta(v)} x_e = 2 \qquad \forall\, v \in V \setminus \{1\}$$

Rilassamento Lagrangiano

La formulazione matematica del minimum 1-tree con l'introduzione dei moltiplicatori Lagrangiani, ottenuta mediante rilassamento Lagrangiano del vincolo (1) sul problema del ciclo Hamiltoniano di costo minimo, è la seguente:

$$\min \sum_{e \in E} c_e x_e - \sum_{v \in V \setminus \{1\}} \theta_v \left(2 - \sum_{e \in \delta(v)} x_e \right)$$

$$(1) \quad \sum_{e \in \delta(1)} x_e = 2$$

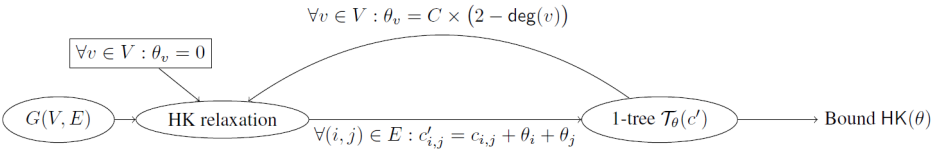
$$(2) \quad \sum_{e \in E} x_e = |V|$$

$$(3) \quad \sum_{\substack{i,j \in S \\ i<j}} x_{i,j} \leq |S| - 1 \quad \forall S \subset V \setminus \{1\} \wedge |S| \geq 3$$

$$(4) \quad x_e \in \{0, 1\}$$

Rilassamento Lagrangiano

- Un rilassamento ottimale per il 1-tree può essere individuato andando ad ottimizzare sulle variabili θ_v.
- A tal proposito, Held e Karp (1970, 1971), hanno proposto un approccio iterativo. L'idea è quella di aggiustare i moltiplicatori lagrangiani θ, passo dopo passo, per costruire una sequenza di 1-tree che fornisca dei limiti sempre migliori.



Rilassamento Lagrangiano

- Viene calcolato un primo minimum 1-tree cercando un minimum spanning tree su $G\backslash\{1\}$ e aggiungendo i due archi più economici incidenti sul nodo 1.
- Se il minimum 1-tree ottenuto è un tour, allora esso corrisponde alla soluzione ottimale del TSP.
- Altrimenti, vengono penalizzati alcuni nodi in quanto almeno un nodo ha un grado superiore a 2.

Rilassamento Lagrangiano

- L'idea principale di Held e Karp (1970, 1971) è quella di penalizzare tali nodi modificando il costo c_{ij} degli archi $(i, j) \in E$, in base ai valori di θ_i e θ_j. Siano $c'_{ij} \in \mathbb{R}$ i costi modificati.

$$c'_{ij} = c_{ij} + \theta_i + \theta_j \qquad \forall\, (i, j) \in E$$

- L'equazione sotto propone una scelta standard per calcolare i moltiplicatori, dove $C \in \mathbb{R}$ è una costante arbitraria e $deg(v)$ indica il grado del nodo $v \in V$ nel 1-tree corrente.

$$\theta_v = C \times \big(2 - deg(v)\big) \qquad \forall\, v \in V$$

Rilassamento Lagrangiano

- A questo punto, può essere calcolato un nuovo minimum 1-tree dal grafo con i costi aggiornati c'_{ij}.
- Questo 1-tree è indicato con $T_\theta(c')$, dove $c' = \{c_1, \ldots, c_{|E|}\}$ è l'insieme di tutti i costi modificati e $\theta = \theta_1, \ldots, \theta_{|V|}$ è l'insieme di tutti i moltiplicatori.
- Questo processo viene ripetuto e si ottiene un nuovo 1-tree $T_\theta(c')$ fino a quando non si ottiene più nessun miglioramento.

$$HK(\theta) = cost\big(T_\theta(c')\big) - 2\sum_{i=1}^{|V|} \theta_i \tag{1}$$

Branch and Bound

- Esistono diversi approcci per la determinazione esatta di un problema NP-completo come il TSP; tra questi, quelli di enumerazione implicita come il branch and bound sono i più diffusi.
- Questi algoritmi esplorano in modo sistematico lo spazio delle soluzioni alla ricerca di una soluzione ottima.
- I lower e upper bound sul valore ottimo della funzione obiettivo vengono sfruttati per ottenere delle informazioni sul problema, permettendo così di escludere dalla ricerca aree dello spazio delle soluzioni in cui non è presente la soluzione ottima.

Calcolo del lower bound

UNIVERSITÀ DI TORINO

- Per determinare un minimum 1-tree e ottenere quindi un lower bound per il problema del TSP, si calcola un minimum spanning tree su $G\backslash\{n\}$ e si aggiungono i due archi più economici incidenti sul nodo n. Il costo principale è $O(m \log n)$ con l'algoritmo di Kruscal.
- La selezione degli archi di costo minimo può essere invece effettuata con una semplice scansione degli archi in $O(m)$.
- Se il 1-tree calcolato risulta essere un circuito hamiltoniano, allora il lower bound può essere aggiornato se il suo costo è minore di quello attualmente noto. Il lower bound verrà inizializzato a $+\infty$ all'inizio della procedura.

Schema di branching

- È possibile rappresentare l'insieme ammissibile X attraverso un albero decisionale (o albero di branch), il quale associa a ciascuna soluzione ammissibile una sequenza di decisioni che la genera.
- Nel caso particolare del TSP, se la soluzione del rilassamento ottenuto risulta essere un circuito hamiltoniano, allora il nodo corrispondente viene chiuso per ammissibilità.
- Se invece il nodo non viene chiuso per ammissibilità, quello che vogliamo ottenere è impedire che nei nodi figli si vada a formare lo stesso circuito presente nel nodo padre.

Schema di branching

- Indichiamo con $\{(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)\}$ gli archi che compongono tale circuito.
- Il primo figlio verrà generato imponendo che l'arco (i_1, j_1) non faccia parte della soluzione, ossia impostando $x_{i_1 j_1} = 0$.
- il secondo figlio verrà generato imponendo invece che sia presente l'arco (i_1, j_1) ma assente l'arco (i_2, j_2), ovvero impostando $x_{i_1 j_1} = 1$ e $x_{i_2 j_2} = 0$.
- Ad ogni nodo dell'albero di branch così costruito sarà associato sia l'insieme E_0, il quale contiene tutti gli archi che NON fanno parte della soluzione, sia l'insieme E_1 che contiene tutti gli archi che invece fanno parte della soluzione.

Schema di branching

- Ciò consente di risolvere per ogni figlio un sottoproblema del tipo $S = (E_0, E_1)$, contenente tutti i circuiti hamiltoniani formati dagli archi presenti in E_1 ma non dagli archi presenti in E_0.
- Nella nostra implementazione, il calcolo dell'MST avviene utilizzando l'algoritmo di Kruscal. L'algoritmo non verrà inizializzato con l'insieme vuoto ma con gli archi presenti in E_1 che non sono incidenti sul nodo n.

Chiusura dei nodi

- Se $\hat{z} < LB(P_i)$, ossia quando il lower bound ottenuto è maggiore del migliore circuito hamiltoniano fin ora trovato, allora si può evitare di esplorare ulteriormente la parte dello spazio delle soluzioni rappresentata dal nodo. Il nodo viene chiuso per bound.
- Se non esiste alcuna soluzione ammissibile per il rilassamento considerato il nodo viene chiuso (o potato) per inammissibilità.
- Se il 1-tree generato dal rilassamento risulta essere un circuito hamiltoniano, allora viene chiuso perché è un possibile candidato ottimo.

Strategie di visita

- La scelta del prossimo nodo da visitare nell'albero decisionale può avvenire con strategie topologiche o basate sull'informazione.
- Le strategie topologiche includono la breadth-first (Q come coda) e la depth-first (Q come pila); la depth-first è utile negli algoritmi di enumerazione implicita per raggiungere rapidamente le foglie e generare soluzioni ammissibili.
- Le strategie topologiche sono semplici da implementare, con un basso costo di gestione di Q; la depth-first può mantenere un numero ridotto di nodi attivi.

Strategie di visita

- La strategia breadth-first richiede memoria esponenziale rispetto alla profondità dell'albero, mentre la depth-first è più efficiente in termini di memoria, memorizzando solo una sequenza di nodi.
- Le strategie basate sull'informazione, come la best-first, utilizzano una coda di priorità per selezionare il nodo più promettente, dirigendo la ricerca verso le zone più promettenti dell'albero decisionale.
- La strategia best-first è più complessa da implementare e richiede più memoria rispetto alle visite topologiche, rendendola meno pratica in contesti con risorse di memoria limitate.

Implementazione

- Strumenti utilizzati:
  - Java
  - TSPLib
- Abbiamo utilizzato le liste di adiacenza implementate con HashMap per rappresentare il grafo e il 1-tree, offrendo una gestione della memoria più efficiente ($O(n+m)$) rispetto a una matrice di adiacenza ($O(n^2)$). Questa struttura permette di scansionare gli adiacenti di un nodo con una complessità di $O(|A(u)|)$.

Implementazione

- Abbiamo implementato l'algoritmo di Kruskal per calcolare l'MST, il quale seleziona gli archi di peso minimo in ordine crescente, evitando cicli tramite operazioni di partizione (Find-Set e Union-Set) sui nodi. L'ordinamento degli archi richiede $O(m \log m)$ passi, mentre il costo complessivo dipende dalle operazioni Find-Set e Union-Set.
- Abbiamo implementato due versioni del branch and bound per il TSP:
  - una multi-thread con strategia Best-first search, che sfrutta la parallelizzazione per esplorare i sottoalberi più promettenti;
  - una single-thread con strategia Depth-first search, più adatta a situazioni con limitazioni di memoria.

Struttura dell'implementazione

Il codice sviluppato è strutturato nel seguente modo:

- Il package BranchAndBoundTSP contiene le classi necessarie per risolvere il problema del TSP utilizzando l'algoritmo di Branch and Bound. Include l'implementazione dell'algoritmo, la gestione dei problemi intermedi e le eccezioni per i casi non risolvibili.
- Il package Datastructures include le classi per la gestione delle strutture a grafo, l'implementazione dell'algoritmo di Kruskal per il calcolo dell'MST e la LIFO blocking queue.
- Il package TSPLib contiene il parser per leggere e interpretare i file della libreria TSPLib.

Test

Tutti i test sono stati svolti utilizzando un PC dotato di CPU AMD Ryzen 7 5800X e 32 GB di RAM. I risultati ottenuti sono i seguenti:

Burma 14 - BestFS		
NumThreads	Nodi	Tempo
1	1732736	64 s
2	1716906	54 s
8	1744534	31 s

Burma 14 - DFS		
NumThreads	Nodi	Tempo
1	1822381	70 s

Test

Tutti i test sono stati svolti utilizzando un PC dotato di CPU AMD Ryzen 7 5800X e 32 GB di RAM. I risultati ottenuti sono i seguenti:

Ulysses16 - BestFS		
NumThreads	Nodi	Tempo
1	3215759	12 m 32 s
2	3376546	10 m 54 s
8	3154971	7 m 8 s

Ulysses16 - DFS		
NumThreads	Nodi	Tempo
1	3837334	16 m 2 s

Test

Tutti i test sono stati svolti utilizzando un PC dotato di CPU AMD Ryzen 7 5800X e 32 GB di RAM. I risultati ottenuti sono i seguenti:

Gr17 - BestFS		
NumThreads	Nodi	Tempo
1	> 4.5 M	> 30 m, Out of memory
2	> 4.5 M	> 23 m, Out of memory
8	> 4.5 M	> 18 m, Out of memory

Gr17 - DFS		
NumThreads	Nodi	Tempo
1	5089038	33 m 34 s

Test

- I test sulle istanze TSP Burma14, Ulysses16 e Gr17 mostrano che la strategia BestFS con esecuzione multi-thread riduce significativamente i tempi di esecuzione, ma può incorrere in problemi di memoria per istanze più grandi, come Gr17.
- La strategia DFS, sebbene più lenta rispetto a BestFS, è più stabile e non incontra problemi di memoria anche per istanze più grandi.
- I risultati evidenziano l'efficacia di BestFS con multi-threading per la velocità, ma la gestione della memoria rimane un punto cruciale per istanze di grandi dimensioni.

UNIVERSITÀ DI TORINO

Grazie per l'attenzione