TSP simmetrico con risoluzione Branch & Bound

Enrico Rizzello

Anno accademico 2021-2022

1 Il Travelling Salesman Problem

Il problema del commesso viaggiatore (o TSP) è un caso di studio cardine dell'informatica teorica nell'ambito della complessità computazionale. Il problema da risolvere è definito da un insieme di città con note le distanze tra ciascuna coppia di esse. Definendo una città iniziale occorrerà trovare il tragitto di minima percorrenza che un commesso viaggiatore deve seguire per visitare tutte le città una ed una sola volta e ritornare alla città di partenza. Si potrebbe associare al TSP un grafo non orientato $G = (V, E)$ con costi c_{ij} associati agli archi. E' dunque richiesto di determinare un insieme di archi $C^* \subset E$ con costo di percorrenza minimo possibile. Questo insieme C^* deve quindi determinare un circuito hamiltoniano, che è un ciclo che passa una ed una sola volta per ogni nodo del grafo in esame. In questa trattazione la versione simmetrica del TSP ossia $c_{ij} = c_{ji} \ \forall i, j \in V$ sarà quella analizzata ed implementata, di conseguenza ogni arco è quindi percorribile nelle due direzioni con lo stesso costo.
Si noti che il problema del TSP è della tipologia NP-hard, nella precisione NP-completo. Ne consegue che gli algoritmi che lo risolvono all'ottimo richiedono purtroppo una una complessità in tempo più che polinomiale ($P \neq NP$).
In questa trattazione sarà applicato una soluzione al problema di tipo Branch & Bound: Di base quello che l'algoritmo farà sarà suddividere nella fase di "Branching"lo spazio delle soluzioni in vari sottospazi la cui unione definisce però la regione delle soluzioni iniziale. Nella fase successiva di "Bounding"si potrà trarre vantaggio da questa suddivisione in quanto avremo a che fare con

spazi ridotti dove sarà più semplice effettuare la relativa analisi e la valutazione delle soluzioni ammissibili. Inoltre sarà più semplice potare quei sottoalberi nel caso di inammissibilità o dove abbiamo la certezza che abbiamo già trovato una soluzione migliore di quella che potremo ottenere proseguendo l'analisi.
Nella sezione successiva partendo dalla formulazione matematica del TSP verrà dimostrato come mediante un rilassamento di tipo lagrangiano su certi specifici vincoli renderemo la risoluzione del problema decisamente molto più semplice e di complessità polinomiale.

2 Ciclo Hamiltoniano e Formulazione Matematica

Per risolvere il problema del TSP occorre quindi trovare un circuito hamiltoniano con costo minore di tutti gli altri possibili. Un circuito hamiltoniano è dunque quel circuito dove su ogni nodo incidono esattamente due archi e rimuovendo un nodo n qualsiasi e i suoi due archi incidenti, si ottiene un albero sui rimanenti $|V| - 1$ nodi. Da questa definizione è possibile effettuare una formalizzazione matematica del TSP:

$$
\begin{aligned}
\min z = & \sum_{(i,j)\in E} c_{ij} \cdot x_{ij} \\
A \quad & \sum_{j\in V, i\neq j} x_{ij} = 2 \quad \forall i \in V \\
B \quad & \sum_{(i,j)\in E(U)} x_{ij} \leq |U| - 1 \quad \forall\, U \subseteq V \setminus \{n\} : |U| \geq 3 \\
C \quad & \sum_{(i,j)\in E, i, j\neq n} x_{ij} = |V| - 2 \\
D \quad & x_{ij} \in \{0, 1\} \quad \forall (i,j) \in E
\end{aligned}
$$

Il vincolo "A"impone che ogni nodo $i \in V$ vi siano esettamente due archi. I vincoli successivi "B"e "C"invece garantiscono che una volta scelto un nodo n, $V \setminus \{n\}$ risulti un albero privo di circuiti e che abbia $|V| - 2$ archi.
Il vincolo "B"nello specifico serve per eliminare gli eventuali circuiti: Definito un certo sottoinsieme di nodi $U \subseteq V$, definiamo $E(U)$ come l'insieme di archi $\{(i,j)\,|\,i,j \in U\}$. Si noti che ogni circuito sui nodi in U deve avere $|U|$ archi in $E(U)$, quindi per eliminare i circuiti consideriamo $|U| \geq 3$, che è definito appunto dal vincolo "B". L'ultimo vincolo "D"invece gestisce il dominio delle variabili decisionali: queste assumono il valore 1 se l'arco in analisi viene inserito nel circuito hamiltoniano e 0 in caso contrario.
Ora che è stata definita una riformulazione matematica del problema del TSP è quindi possibile un rilassamento o Lower Bound al problema.
Nella trattazione seguente è stato effettuato questo rilassamento Lagrangiano sul vincolo "A", questo trasforma il problema nel trovare il minimo 1-Tree. Questo è possibile eliminando tale vincolo per tutti i nodi tranne che per il nodo selezionato n e quindi impostando i moltiplicatori lagrangiani λ_i a 0.

$$\min z = \sum_{(i,j)\in E} c_{ij} \cdot x_{ij} + \sum_{k\in V} \lambda_k \, (2 \; - \sum_{j\in V, k\neq j} x_{kj})$$

$$A' \quad \sum_{j\in V, j\neq n} x_{nj} = 2$$

$$B \quad \sum_{(i,j)\in E(U)} x_{ij} \leq |U| - 1 \quad \forall \, U \subseteq V \setminus \{n\} : |U| \geq 3$$

$$C \quad \sum_{(i,j)\in E, i,j\neq n} x_{ij} = |V| - 2$$

$$D \quad x_{ij} \in \{0,1\} \quad \forall (i,j) \in E$$

La precedente è la riformulazione dei vincoli effettuando un rilassamento su "A"e modificando la funzione obiettivo aggiungendo il vincolo e introducendo dei moltiplicatori lagrangiani λ_i per ogni nodo. Per comodità notazionale è stato aggiunto in funzione obiettivo anche il moltiplicatore λ_n per il nodo n ma verrà posto a 0.
Dopo semplici calcoli algebrici otteniamo:

$$l(\lambda) = \min \sum_{(i,j)\in E} c'_{ij} \cdot x_{ij} + 2 \cdot \sum_{k\in V} \lambda_k$$

Con costi associati agli archi:

$$c'_{ij} = c_{ij} - \lambda_i - \lambda_j$$

3 1-Tree e Formulazione Matematica

Definiamo ora un 1-tree in quanto sarà poi importante per la trattazione seguente:
Dato un grafo non orientato $G = (V, E)$ e un nodo n, definiamo 1-tree un sottografo $H = (V, E_H)$ di G con $E_H \subset E$ con le proprietà di avere esattamente due archi incidenti sul nodo n in E_H ed escludento da H n e i suoi archi incidenti, questo risulta un albero sull'insieme dei nodi $V \setminus \{n\}$, più intuitivamente si afferma che H contenga un circuito passante per il nodo n.
Da questa definizione segue che $|E_H| = |V|$.
In questo rilassamento, come accennato nel precedente paragrafo, i moltiplicatori lagrangiani sono stati impostati tutti a 0. Questo elimina del tutto il vincolo e ci permette proprio di ricondurre il problema alla ricerca dell'1-tree di costo minimo.
Si noti però che qualsiasi circuito hamiltoniano risulta un 1-tree, ma non è vero il contrario. Ne consegue quindi che se indichiamo con S' l'insieme di tutti gli 1-tree ottenuti da un certo grafo e con S la regione ammissibile del TSP, abbiamo che $S \subset S'$. Possiamo quindi definire la funzione obbiettivo come:

$$\min z = \sum_{(i,j)\in E} c_{ij} \cdot x_{ij}$$

E'evidente come sia quindi un rilassamento per il problema del TSP simmetrico in quanto la sua risoluzione restituisce un lower bound per il valore ottimo del problema del TSP.

$$
\begin{aligned}
\min z = & \sum_{(i,j)\in E} c_{ij} \cdot x_{ij} \\
& \sum_{j\in V, j\neq n} x_{nj} = 2 \\
& \sum_{(i,j)\in E(U)} x_{ij} \leq |U| - 1 \quad \forall\, U \subseteq V \setminus \{n\} : |U| \geq 3 \\
& \sum_{(i,j)\in E, i,j\neq n} x_{ij} = |V| - 2 \\
& x_{ij} \in \{0,1\} \quad \forall (i,j) \in E
\end{aligned}
$$

La precedente è la riformulazione dei vincoli finale e si può notare chiaramente che è equivalente alla riformulazione iniziale post rilassamento settando i moltiplicatori lagrangiani a 0.

4 Calcolo del Lower Bound

Un lower bound al problema del TSP simmetrico è quindi il calcolo di un 1-tree di costo minimo: Questa procedura si può calcolare molto semplicemente trovando l'MST sul grafo che viene ricavato eliminando il nodo selezionato n e i suoi relativi archi incidenti. In seguito denominando E_T l'insieme di archi della soluzione ottenuta, aggiungeremo i due archi (n,k) e (n,h) che hanno distanza minima tra quelli che incidono sul nodo n scelto. Infine verrà restituito l'1-tree $H = (V, E_H)$ con $E_H = E_T \cup \{(n,k),(n,h)\}$.
Considerando che per la selezione della coppia di archi di costo minore possiamo effetturare una semplice scansione degli archi di costo $O(m)$, è stato scelto l'algoritmo greedy di Kruscal di costo $O(m \cdot \log n)$ per il calcolo dell'MST, in quanto è evidente che trattasi del passo con tempo di risoluzione potenzialmente maggiore della procedura.
Una soluzione più raffinata al costo di un maggior tempo computazionale potrebbe essere il calcolo di ogni 1-tree minimi scegliendo n diversi tra loro tra i nodi del grafo e scegliendo come lower bound complessivo quello migliore ottenuto con questa procedura.

5 Calcolo dell'Upper Bound

La ricerca delle soluzioni ammisssibili è direttamente correlato al calcolo dell'upper bound: Questo rissulta evidente in quanto dopo il calcolo di un 1-tree se poi risulta essere un circuito hamiltoniano con ogni nodo con un grado pari a due, l'upper bound della nostra soluzione verrà aggiornato solo se quello appena calcolato avrà un costo minore di quello corrente. E' evidente che inizialmente per il calcolo dell'upper bound risulta fondamentale settarlo a infinito in modo che possa esssere immediatamente sovrascritto dal primo upper bound calcolato dall'algoritmo.

6 Procedura di Brancing

La procedura di brancing ci permetterà di gestire quei casi in cui la soluzione del rilassamento non sia un circuito hamiltoniano ma bensì un 1-tree con un sottocircuito al suo interno. Per ovviare appunto a questo problema verrà partizionata con questa procedura la regione ammissibile S in più sottoinsiemi, introducendo una regola di divisione che possa impedire la formazione nei nodi figli di questo sottocircuito.
Se denotiamo con: $\{(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)\}$ gli archi che creano tale sottocircuito, allora il primo nodo figlio sarà generato settando $x_{i_1 j_1} = 0$ e quindi con la condizione che l'arco (i_1, j_1) non sia parte della soluzione, mentre il secondo nodo figlio verrà generato imponendo che (i_2, j_2) non venga inserito (quindi $x_{i_2 j_2} = 0$), ma che sia presente l'arco (i_1, j_1). La procedura continua per tutti i nodi in modo analogo fino all' r-esimo figlio che non conterrà l'ultimo modo, (quindi $x_{i_r j_r} = 0$) ma avrà tutti i primi $r - 1$ archi del sottocircuito ($\forall k = 1, 2, \ldots, r - 1 \;\; x_{i_k j_k} = 1$).
L'algoritmo utilizzerà due insiemi: E_0 ed E_1 (con $S(E_0, E_1)$), dove E_0 conterrà gli archi che verranno esclusi mentre E_1 quelli che saranno presenti in soluzione.
Ogni figlio avrà un sottoproblema del tipo $S(E_0, E_1)$ che conterrà tutti i circuiti hamiltoniani formati dagli archi in E_1 escludendo gli archi in E_0. Per calcolare il lower bound del sottoproblema si usa una procedura analoga a quella del paragrafo 4 dove saranno esclusi gli archi in E_0 e saranno presenti quelli in E_1, sia nella selezione dei due archi per il nodo n che per il calcolo dell'MST. Verrà quindi inizializzato l'insieme E_T con gli archi in E_1 che non sono incidenti sul nodo n (e non l'insieme vuoto) e verrà risolto il problema dell'MST con l'algoritmo di Kruscal, escludendo gli archi E_0 durante l'esecuzione dell'algoritmo. Dopo aver calcolato l'albero di copertura T a questo verranno aggiunti gli archi incidenti in n a costo più basso e quindi migliori. Se in E_1 non sono presenti questi archi vengono selezionati i migliori non presenti in E_0.
Per quanto concerne il branching di un nodo interno al sottocircuito $\{(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)\}$, dovranno essere eliminati gli archi dall'insieme E_1 correlato al nodo stesso, i quali non possono essere presenti nei figli che saranno creati. Infine verranno estesi gli insieme E_0, E_1 del nodo padre per la creazione dei figli, ciò garantisce che questi nodi figli continueranno ad essere un sottoinsieme della regione ammissibile del nodo padre.

7 Chiusura dei nodi

Per quanto riguarda la chiusura di un branch (B_i) all'interno dell'albero si hanno tre criteri:

1. Vi è una chiusura per "candidato ottimo"se dal rilassamento è stato generato un 1-tree che risulta essere anche un circuito hamiltoniano e la soluzione migliore deve essere aggiornata in caso il costo di questa sia minore dell'attuale migliore soluzione trovata.

2. Quando il lower bound ottenuto è superiore al migliore circuito hamiltoniano per ora trovato ($\hat{z} \leq LB(PB_i)$) e quindi il nodo viene chiuso per "bound".

3. Vi è una chiusura per inammissibilità se non c'è una soluzione ammissibile per tale rilassamento ed è quindi impossibile calcolare un 1-tree per il nodo.

Viene spesso utilizzato un approccio "Best First"per la scelta del nodo su cui fare prima Branch in caso ve ne siano più aperti. Questo genere di approccio si basa sull'analisi del Lower Bound: Viene infatti selezionato il nodo col il Lower Bound minore e dunque quello che ci potrebbe permettere, in caso vi sia una concreta ottimalità, di terminare l'esecuzione chiudendo gli altri nodi aperti.

8 Strutture Implementative

A seguire verranno discusse le strutture implementative scelte per questa tesina.
La struttura dati principale è implementata come un grafo ed è' rilevante sottolineare l'utilizzo di un'HashMap per rappresentare le liste di adiacenze calcolate per ogni 1-tree trovato, in quanto essendo inteso un grafo come un insieme di nodi correlati ad un certo numero di archi, risulta più semplice ed efficiente l'utilizzo delle HashMap per realizzare questi insiemi.
E' stato scelto l'algoritmo di Kruskal (come anticipato in precedenza) per l'implementazione dell'algoritmo di Minimum Spanning Tree, in quanto è una delle procedure eseguite più frequentemente e la complessità di tale algoritmo è $O(m \cdot \log n)$. L'algoritmo inizialmente esegue un ordinamento decrescente degli archi e successivamente esegue Merge Find Set per verificare che ogni arco in esamina non formi un ciclo e quindi possa essere incluso nella soluzione.
Dato che le due operazioni hanno costo logaritmico ne indichiamo la complessità definitiva come $O(m \cdot \log n)$. La funziuone "Depth First Search"permette infine l'individuazione dell'eventuale sottocircuito dell'1-tree in esamina in caso non sia un circuito Hamiltoniano, questa usa un "father array"che tiene traccia dei nodi padri in modo che non vengano visitati più volte gli stessi nodi del grafo. In seguito a partire da n viene individuato il ciclo procedendo a ritroso con i vettori dei padri, e da questo si farà branch B_i.

9 Analisi del codice

La procedura centrale per la risoluzione del TSP è firmata solveProblem() e restituisce un'istanza della classe TSPResult, contenente il circuito hamiltoniano con costo minore e una serie di statistiche riguardo ai nodi analizzati nell'albero di Branch. Nel caso in cui non vi fosse un circuito hamiltoniano nel grafo di partenza, viene restituito il grafo originale. Questo metodo richiama gli algoritmi sopra descritti e molte altre procedure di supporto al fine di gestire una PriorityQueue di SubProblem e terminando l'esecuzione non appena quest'ultima viene completamente svuotata in seguito alla chiusura di tutti i nodi per una delle ragioni precedentemente indicate.
É qui che risiede tutta la complessità della risoluzione del TSP simmetrico. Questo metodo infatti richiama sottoprocedure polinomiali in tempo, ma il numero di SubProblem che deve gestire può essere esponenziale.
La gestione dell'insieme dei SubProblem ancora aperti è stata realizzata con una coda di priorità **min-heap**, ossia con una struttura dati che presenta complessità $O(1)$ per leggere l'elemento con lower bound minore e $O(\log\ n)$ per estrarre ed aggiungere un elemento.

Terminiamo questo paragrafo con l'analisi delle prestazioni dell'implementazione. Su grafi banali come quello che verrà mostrato nell'esempio conclusivo della tesina il codice termina in pochi millisecondi generando una manciata di nodi nell'albero di branch.
Come abbiamo potuto toccare con mano durante i nostri test, il tempo necessario per l'esecuzione dipende fortemente dal numero di archi che popolano il grafo. Esecuzioni su grafi sparsi di 100 nodi richiedono decine di secondi, mentre esecuzioni su grafi completi anche solo di 20 nodi richiedono potenzialmente diversi minuti. Per ottenere un'analisi significativa delle prestazioni, abbiamo creato una classe Java BasicCompleteGraphGenerator che, come si evince dal nome stesso, ci permette di generare grafi completi una volta specificato il numero di nodi e il range entro il quale devono risiedere i costi degli archi. Ricordiamo che un grafo completo G con n nodi avrà $\frac{n\cdot(n-1)}{2}$ archi non orientati.
Per dare maggiore rilevanza ai test effettuati abbiamo anche dato la possibilità di parallelizzare i calcoli su più thread. Una volta infatti che si è fatto il branch di un nodo e si ha generato tutti i figli di uno stesso livello questi possono tranquillamente essere computati simultaneamente per poi decidere se vadano chiusi per un qualche motivo o espansi con una ulteriore operazione di

branch.
Mostriamo ora i dati che abbiamo raccolto. L'esecuzione è avvenuta su un calcolatore con le seguenti caratteristiche:

- Processore AMD Ryzen 7 5800X (4.6GHz)
- 32GB di RAM DDR4 (3200MHz)

Thread	$V \in G$	$E \in G$	# sottoproblemi	Tempo medio d'esecuzione
1	5	10	9	<1ms
2	5	10	10	<1ms
4	5	10	10	<1ms
8	5	10	10	<1ms
1	10	45	3043	72ms
2	10	45	3043	66ms
4	10	45	3044	28ms
8	10	45	3044	16ms
1	13	105	130192	4.2s
2	13	105	130192	4.1s
4	13	105	130186	1.8s
8	13	105	130164	1s
1	15	105	811302	33.3s
2	15	105	811302	33.2s
4	15	105	811306	18.6s
8	15	105	811306	9.0s
1	20	190	>2.5M	...
2	20	190	>2.5M	...
4	20	190	>2.5M	...
8	20	190	>2.5M	...

Sia il tempo che il numero di sottoproblemi generati sono stati calcolati facendo una media dei risultati ottenuti su 500 esecuzioni per ogni istanza di grafo completo e per ogni numero di thread. Il campione usato è stato ridotto a 30 solamente nell'ultima serie di test a causa del tempo richiesto sempre maggiore. I puntini nella tabella rappresentano esecuzioni che non sono terminate a causa della mancanza di RAM per poter ospitare tutti i sottoproblemi ancora da valutare. Il Garbage Collector di Java si occupa di eliminare tutte le istanze di classi non più referenziate, ma il numero di SubProblem che sono ancora attivi nella frontiera dell'albero di branch può essere esponenzialmente grande. Basti pensare al fatto che non tutti i SubProblem rimangono aperti e generano figli, ma coloro che lo fanno possono avere un branching factor molto elevato.
Ovviamente tutto questo non ci sorprende più di tanto poichè rispecchia quella che è una procedura di risoluzione esponenziale per un problema NP-hard come il TSP.

10 Esempio

Risolveremo in quest'ultimo paragrafo della tesina un esercizio sul TSP simmetrico preso da una vecchia prova d'esame del corso di Ottimizzazione Combinatoria datata 10/06/2015.

L'istanza di grafo $G(V, E)$ è la seguente:

$$c_{ij}: \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} - & & & & \\ 5 & - & & & \\ 2 & 6 & - & & \\ 3 & 7 & 8 & - & \\ 8 & 10 & 2 & 1 & - \end{pmatrix} \end{array}$$

Come nodo candidato scegliamo arbitrariamente il nodo 1 e calcoliamo l'1-tree generando prima l'MST sui $V \setminus \{1\}$ vertici e poi aggiungendo i due archi ad esso incidenti di peso minimo. L'1-tree che otteniamo è il seguente:

Ci troviamo con un primo 1-tree che non è un circuito hamiltoniano e sul quale dobbiamo fare branch. Il sottocircuito che esso contiene è formato dai seguenti archi $\{(1,3),(3,5),(5,4),(4,1)\}$. Genereremo 4 figli così caratterizzati:

Nodo 2: $E_0 = \{(1,3)\}$ e $E_1 = \emptyset$;

Nodo 3: $E_0 = \{(3,5)\}$ e $E_1 = \{(1,3)\}$;

Nodo 4: $E_0 = \{(5,4)\}$ e $E_1 = \{(1,3),(3,5)\}$;

Nodo 5: $E_0 = \{(4,1)\}$ e $E_1 = \{(1,3),(3,5),(5,4)\}$.

Calcolando per ognuno dei nuovi nodi il corrispondente 1-tree abbiamo rispettivamente:

Il nodo 2 viene chiuso per ottimalità in quanto presenta un circuito hamiltoniano di costo 17, che diventa la nostra prima soluzione. Il nodo 3 viene chiuso per Bound in quanto presenta un lower bound pari a 19, maggiore della nostra soluzione. Discorso analogo viene fatto per il nodo 4 con lower bound 20. Il nodo 5 rimane invece aperto dato che presenta lower bound 16 e sul quale faremo branch.
L'albero di Branch aggiornato con queste nuove considerazioni risulta essere:

Il sottocircuito che il nodo 5 presenta è formato dagli archi $\{(1,2),(2,3),(3,1)\}$, scremato dagli archi nel suo E_1 otteniamo $\{(1,2),(2,3)\}$. I 2 figli che generiamo sono così caratterizzati:

Nodo 6: $E_0 = \{(4,1),(1,2)\}$ e $E_1 = \{(1,3),(3,5),(5,4)\}$;

Nodo 7: $E_0 = \{(4,1),(2,3)\}$ e $E_1 = \{(1,3),(3,5),(5,4),(1,2)\}$.

Calcolando per ognuno dei nuovi nodi il corrispondente 1-tree abbiamo rispettivamente:

Il nodo 6 viene chiuso per bound, dato che ci garantisce un lower bound pari a 19, contro il costo 17 del nostro ottimo attuale. Il nodo 7 viene invece chiuso per ottimalità dato che presenta un circuito hamiltoniano con costo 17, uguale al nostro ottimo attuale che quindi non viene aggiornato.

L'albero di Branch finale risulta:

Il TSP simmetrico presenta come ottimo due possibili circuiti hamiltoniani di ugual costo: $\{(1,2),(2,3),(3,5),(5,4),(4,1)\}$ e $\{(1,2),(2,4),(4,5),(5,3),(3,1)\}$.