Algoritmi per la risoluzione del TSP

Michele Maione – 931468 – michele.maione@studenti.unimi.it. Ottimizzazione Combinatoria, A/A 2019-2020, Università degli Studi di Milano, via Celoria 18, Milano, Italia.

1 – Abstract

Per la risoluzione del TSP ho implementato gli algoritmi di approssimazione con rilassamento lagrangiano di Held–Karp[4] e di Volgenant–Jonker[5], l'algoritmo di Christofides[6] (1,5–approssimato), un algoritmo 2–approssimato[1] (Thomas Cormen) e quello di programmazione dinamica di Held–Karp[3] (per un confronto preciso su piccole topologie).

2 – Introduzione

Il problema del commesso viaggiatore (TSP, Traveling Salesman Problem) è trovare il percorso minimo passante per un insieme di città, tale che, si passi una ed una sola volta per la stessa città, e si ritorni alla città di partenza. Originariamente, il problema era trovare il tour più breve di tutte le capitali degli Stati Uniti.

Matematicamente può essere rappresentato come un grafo pesato. Se il grafo è diretto allora il TSP è definito asimmetrico. Lo scopo consiste nel trovare un ciclo hamiltoniano a costo minimo sul grafo, questo è un problema NP-completo, e viene quindi, principalmente, calcolato tramite approssimazione.

2.1 – Modello matematico

Ad ogni arco del grafo $G = (V, E)$ è associato un peso c, e una variabile booleana x che indica se l'arco appartiene al percorso. La formulazione matematica del problema, fornita da Dantzig–Fulkerson–Johnson, è:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij} \qquad (i \neq j)$$

$$S.T.$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad (i \neq j, \forall j \in V)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i \neq j, \forall i \in V)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad (i \neq j, \forall S \not\subseteq \{1, \ldots, n\}, |S| \geq 2)$$

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

3 – Algoritmo D.P. di Held–Karp

L'algoritmo Held–Karp, è un algoritmo di programmazione dinamica proposto nel 1962 per risolvere il TSP. L'algoritmo si basa su una proprietà del TSP: ogni sotto-percorso di un percorso di minima distanza è esso stesso di minima distanza; quindi calcola le soluzioni di tutti i sotto-problemi partendo dal più piccolo. Non potendo conoscere quali sotto-problemi risolvere, di fatto devono essere risolti tutti. L'algoritmo ha una complessità temporale $O(2^n \cdot n^2)$ e una complessità spaziale di $O(2^n \cdot \sqrt{n})$.

3.1 – Ottimizzazione

Ci sono due ottimizzazioni spaziali che possono essere fatte durante l'esecuzione:

1. Alla fine dell'elaborazione di una cardinalità s, si possono eliminare gli elementi della cardinalità $s - 2$, portando l'algoritmo ad una complessità spaziale da $O(2^n \cdot n)$ a $O(2^n \cdot \sqrt{n})$;
2. Prima dell'elaborazione di un set, si possono eliminare gli elementi appartenenti alla cardinalità $s - 1$, che hanno il primo elemento minore dal primo elemento del set attuale, riducendo del 15% la memoria utilizzata.

Nella seguente tabella di esempio di 5 nodi, ho evidenziato con colori diversi i set di nodi che dipendono tra di loro. Resta il fatto che la dipendenza è relativa solo alla cardinalità precedente.

Cardinalità	Set
1	{1}, {2}, {3}, {4}, {5}
2	{1,2}, {1,3}, {1,4}, {1,5}, {2,3}, {2,4}, {2,5}, {3,4}, {3,5}, {4,5}
3	{1,2,3}, {1,2,4}, {1,2,5}, {1,3,4}, {1,3,5}, {1,4,5}, {2,3,4}, {2,3,5}, {2,4,5}, {3,4,5}
4	{1,2,3,4}, {1,2,3,5}, {1,2,4,5}, {1,3,4,5}, {2,3,4,5}
5	{1,2,3,4,5}

4 – Algoritmo di Christofides

L'algoritmo Christofides[6] è un algoritmo del 1976 per trovare soluzioni approssimative al problema del TSP euclideo. È un algoritmo di approssimazione che garantisce un fattore 3/2 sulla soluzione ottima. L'algoritmo esegue i seguenti passi:

1. T = albero ricoprente minimo del grafo G;

2. O = insieme dei nodi con grado dispari in T;
3. S = sotto-grafo indotto di G usando i nodi in O;
4. M = accoppiamento perfetto di peso minimo su S;
5. H = multi-grafo connesso formato dagli archi di T e M;
6. C = circuito euleriano su H;
7. Z = circuito hamiltoniano da C, saltando i vertici ripetuti.

Z sarà la soluzione del TSP.

4.1 – Implementazione

Gli algoritmi interni utilizzati sono:

- MST: uso l'algoritmo di Prim *O(E · logV)*;
- OddVertices: creo un vettore di vertici di grado dispari *O(V)*;
- PerfectMatching: ho utilizzato l'algoritmo di Edmonds[7] per calcolare l'accoppiamento perfetto pesato, eseguito in $O(V^2 \cdot E)$;
- Hamiltonian: trovo un circuito euleriano e lo trasformo in un circuito hamiltoniano *O(E)*.

5 – Algoritmi con rilassamento lagrangiano

Gli algoritmi branch and bound con rilassamento lagrangiano di Held–Karp[4] e Volgenant–Jonker[5], risalgono rispettivamente al 1969 e al 1980. Si basano sul concetto che un ciclo hamiltoniano è un 1–albero in cui ogni vertice ha grado uguale a 2. Il costo di un ciclo hamiltoniano su un grafo connesso e non diretto è maggiore del costo minimo di 1–albero su quel grafo. Dato un tour ottimo H^* e un 1–albero T , si ha: $c(H^*) \geq min\{c(T)\}$. Il rilassamento lagrangiano tenta di migliorare il bound eliminando una parte dei vincoli dal problema originale, inserendoli nella funzione obiettivo. In questo caso si eliminano i vincoli di grado sui vertici $\{2,\ldots,n\}$, inserendoli nella funzione obiettivo come somma pesata secondo dei moltiplicatori lagrangiani λ_i . Nella tabella successiva è possibile notare che tutti i vincoli del problema corrispondono al modello matematico di un 1–albero. Le soluzioni del problema lagrangiano sono tutti gli 1–alberi del grafo.

$$
\begin{aligned}
& min\, L(\lambda) = min \sum_{e \in E} c_e \cdot x_e + \sum_{i \in V \setminus \{r\}} \lambda_i \cdot \Big(\sum_{e \in \delta(i)} x_e - 2 \Big) \\
& S.T. \\
& \sum_{e \in V} x_e = n \\
& \sum_{e \in E(S)} x_e \leq |S| - 1 \qquad (\forall\, S \subseteq \{2,\ldots,n\}, |S| \geq 2) \\
& x_e \in \{0,1\} \qquad (\forall\, e \in E)
\end{aligned}
$$

Il valore $L(\lambda)$ costituisce un lower bound. Quindi per ottenere un buon lower bound si può cercare di massimizzare $L(\lambda)$. Questa massimizzazione viene effettuata con il metodo del sub-gradiente.

5.1 – Metodo del sub-gradiente

Sia Held–Karp che Volgenant–Jonker propongono una serie di equazioni per il calcolo di una successione di $\lambda_i (\forall\, i \in V)$ secondo le seguente regole euristiche, tra loro differenti che cambiano leggermente i valori restituiti ma non l'algoritmo. Il parametro $\alpha = 2$ decresce durante l'esecuzione, M è il numero massimo di iterazioni.

Tabella 1: Held–Karp

$$\lambda_i^{k+1} = \lambda_i^k + t^k \cdot (d_i^k - 2)$$

$$t^k = \frac{\alpha^k [UB - L(\lambda^k)]}{\sum_{i \in V} (d_i(x^k) - 2)^2}$$

Tabella 2: Volgenant–Jonker

$$\lambda_i^{k+1} = \lambda_i^k + 0.6 \cdot t^k \cdot (d_i^k - 2) + 0.4 \cdot t^k \cdot (d_i^{k-1} - 2)$$

$$t^k = t^1 \, \frac{k^2 - 3(M-1)k + M(2M-3)}{2(M-1)(M-2)}$$

6 – Algoritmo 2–approssimato (Thomas Cormen)

L'altro algoritmo che ho codificato è un 2–approssimato che viene eseguito in $\Theta(V^2)$. L'algoritmo esegue i seguenti passi:

1. Seleziona un vertice r come radice;
2. Usa l'algoritmo di Prim per calcolare un albero di connessione minimo T;
3. Sia H una lista di vertice, ordinata in base a quando un vertice viene visitato per primo in un attraversamento anticipato di T;
4. Ritorna il ciclo hamiltoniano da H.

7 – Risultati ottenuti

Ho testato su varie istanze di grafi euclidei casuali, l'errore medio per l'algoritmo di Christofides è comunque sotto il fattore massimo 3/2.

	Errore medio		
Nodi	Christofides	Held–Karp	2–approssimato
4	1,00%	0,00%	1,00%
6	4,50%	0,00%	18,39%
10	1,67%	0,00%	15,74%
15	13,77%	0,00%	24,25%
20	4,84%	0,00%	21,88%

Nel seguente grafo da 6 nodi si può notare che l'algoritmo di Christofides calcolare un tour peggiore.

7.1.1 – Tempi d'esecuzione

L'algoritmo di Christofides ha complessità temporale $O(V^2 \cdot E)$ mentre il 2–approssimato $\Theta(V^2)$. Di seguito i risultati ottenuti:

	Tempo d'esecuzione		
Nodi	Christofides	Held–Karp	2–approssimato
25	00'00"01	00'00"01	00'00"01
100	00'00"02	01'41"89	00'00"01
500	00'00"21		00'00"08
1000	00'00"92		00'00"35

8 – Commenti conclusivi

8.1 – D.P. Held–Karp

Ho appurato che la soluzione di programmazione dinamica non è fattibile sui moderni computer personali. L'unico modo per risolvere questo problema su grandi topologie sono gli algoritmi di approssimazione.

8.2 – Christofides

L'algoritmo di Christofides ha un margine d'errore di 3/2, ed è il migliore (qualità/tempo), tra quelli implementati, per risolvere il problema.

8.3 – Rilassamento lagrangiano e branch and bound

Questi algoritmi sono stati quelli che hanno dato i risultati migliori, ma sono molto più lenti rispetto all'algoritmo di Christofides.

8.4 – 2–Approssimato

L'algoritmo 2–approssimato è molto veloce e leggero, e può essere usato per calcolare un discreto upper bound.

9 – Allegati

9.1 – Codice sorgente

Tutto il progetto è disponibile qui: http://github.com/mikymaione/Held-Karp-algorithm

10 – Riferimenti bibliografici

1. Cormen, T. and Leiserson, C. and Rivest, R. and Stein, C., 2010. Introduzione agli algoritmi e strutture dati. McGraw-Hill.
2. Vercellis, C., 2008. Ottimizzazione. Teoria, metodi, applicazioni. McGraw-Hill.
3. Held, M. and Karp, R., 1962. A dynamic programming approach to sequencing problems. Journal for the Society for Industrial and Applied Mathematics, 1:10.
4. Held, M. and Karp, R., 1970. The Traveling Salesman Problem and Minimum Spanning Trees. Operations Research, 18, 1138-1162.
5. Volgenant, T. and Jonker, R., 1982. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. European Journal of Operational Research, 9(1):83–89.
6. Christofides, N., 1976. Worst-case analysis of a new heuristic for the travelling salesman problem. Report 388, Graduate School of Industrial Administration, CMU.
7. Edmonds, J., 1965. Paths, trees, and flowers. Can. J. Math, 17: 449–467.
8. Lippman, S. and Lajoie, J. and Moo, B., 2012. C++ Primer. Addison-Wesley Professional.