8.10 Algoritmi di approssimazione

Dimostrare che un problema è NP-completo significa rinunciare a progettare per esso un algoritmo polinomiale di risoluzione (a meno che non crediamo che P sia uguale a NP). A questo punto, però, ci chiediamo come dobbiamo comportarci: dopo tutto, il problema deve essere risolto. A questo interrogativo possiamo rispondere in diversi modi. Il primo e il più semplice, già trattato nel Paragrafo 8.7, consiste nell'ignorare la complessità temporale intrinseca del problema, sviluppare comunque un algoritmo di risoluzione e sperare che nella pratica, ovvero con istanze provenienti dal mondo reale, il tempo di risoluzione sia significativamente minore di quello previsto: dopo tutto, l'analisi nel caso pessimo, in quanto tale, non ci dice come si comporterà il nostro algoritmo nel caso di specifiche istanze.

Un secondo approccio, in linea con quello precedente, ma matematicamente più fondato, consiste nell'analizzare l'algoritmo da noi progettato nel caso medio rispetto a una specifica distribuzione di probabilità: questo è quanto abbiamo fatto nel caso dell'algoritmo di ordinamento per distribuzione (Paragrafo 3.4). Vi sono due tipi di problematiche che sorgono quando vogliamo perseguire tale approccio. La prima consiste nel fatto che un'analisi probabilistica del comportamento dell'algoritmo è quasi sempre difficile e richiede strumenti di calcolo delle probabilità talvolta molto sofisticati. La seconda e, probabilmente, più grave questione è che l'analisi probabilistica richiede la conoscenza della distribuzione di probabilità con cui le istanze si presentano nel mondo reale: purtroppo, quasi mai conosciamo tale distribuzione e, pertanto, siamo costretti a ipotizzare che essa sia una di quelle a noi più familiari, come la distribuzione uniforme.

Un terzo approccio si applica al caso di problemi di ottimizzazione, per i quali a ogni soluzione è associata una misura e il cui scopo consiste nel trovare una soluzione di misura ottimale: in tal caso, possiamo rinunciare alla ricerca di soluzioni ottime e accontentarci di progettare algoritmi efficienti che producano sì soluzioni peggiori, ma non troppo. In particolare, diremo che `A` è un **algoritmo di `r`-approssimazione** per il problema di ottimizzazione Π se, per ogni istanza `x` di Π, abbiamo che `A` con `x` in ingresso restituisce una soluzione di `x` la cui misura è al più `r` volte quella di una soluzione ottima (nel caso la misura sia un costo) oppure almeno $\frac{1}{r}$-esimo di quella di una soluzione ottima (nel caso la misura sia un profitto), dove `r` è una costante reale strettamente maggiore di 1.

Per chiarire meglio tale concetto, consideriamo il problema del minimo ricoprimento tramite vertici, che nella sua versione di ottimizzazione consiste nel trovare un sottoinsieme dei vertici di un grafo di cardinalità minima che copra tutti gli archi del grafo stesso. Il Codice 8.4 realizza un algoritmo di approssimazione per tale problema basato sul paradigma dell'algoritmo goloso. In particolare, dopo aver inizializzato la soluzione ponendola uguale all'insieme vuoto (righe 3 e 4), il codice esamina uno dopo l'altro tutti gli archi del grafo (righe 5-11): ogni qualvolta ne trova uno i cui due estremi non sono stati selezionati (riga 7), include entrambi gli estremi nella soluzione (righe 8 e 9).

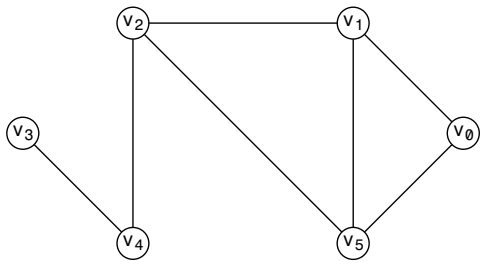
ALVIE **Codice 8.4** Algoritmo per il calcolo di un ricoprimento tramite vertici.

```
1   RicoprimentoVertici( A ):
2                          ⟨pre: A è la matrice di adiacenza di un grafo di n nodi⟩
3     FOR (i = 0; i < n; i = i + 1)
4       preso[i] = FALSE;
5     FOR (i = 0; i < n; i = i + 1)
6       FOR (j = 0; j < n; j = j + 1) {
7         IF (A[i][j] == 1 && !preso[i] && !preso[j]) {
8           preso[i] = TRUE;
9           preso[j] = TRUE;
10        }
11      }
12    RETURN preso;
```

ESEMPIO 8.10

Considerando il grafo mostrato nella Figura 8.3, la soluzione prodotta dall'algoritmo include tutti e sei i vertici; infatti prima vengono inseriti nella soluzione i vertici v_0 e v_1, in quanto (v_0, v_1) non è coperto, poi la coppia v_2 e v_4 e infine v_3 e v_4, poiché l'arco (v_3, v_5) risulta ancora scoperto.

Dovrebbe essere evidente che la soluzione costruita dal Codice 8.4 dipende dall'ordine in cui si elaborano gli archi. Infatti applichiamo lo stesso algoritmo al grafo seguente che è un isomorfismo di quello rappresentato nella Figura 8.3.



Nella soluzione entrano prima i nodi v_0 e v_1 e poi i nodi v_2 e v_4, in quanto l'arco (v_2, v_4) sarà il primo arco esaminato incidente in v_2 e non ancora coperto. In questo caso il ricoprimento finale ha dimensione 4 mentre quello di cardinalità minima ha solamente tre nodi.

Non possiamo garantire che tale soluzione sia di cardinalità minima, ma possiamo però mostrare che la soluzione calcolata dal Codice 8.4 include un numero di vertici che è sempre minore oppure uguale al doppio della cardinalità di una soluzione ottima.

Teorema 8.7 *Il Codice 8.4 è un algoritmo di* 2*-approssimazione per il problema del minimo ricoprimento tramite vertici.*

Dimostrazione Dato un grafo G, la soluzione S prodotta dall'algoritmo è un ricoprimento tramite vertici, poiché ogni arco viene coperto con due vertici se, al momento in cui viene esaminato, questi sono entrambi non inclusi nella soluzione e con almeno un vertice in caso contrario.

Il sottografo indotto dalla soluzione S calcolata dall'algoritmo con G in ingresso, è formato da $\frac{|S|}{2}$ archi a due a due disgiunti, ovvero senza estremi in comune. Chiaramente, un qualunque ricoprimento di tale sottografo (e quindi di G) deve includere almeno $\frac{|S|}{2}$ vertici: pertanto, |S| è minore oppure uguale al doppio della cardinalità di un qualunque ricoprimento tramite vertici di G e, quindi, della cardinalità minima. ❑

La complessità temporale del Codice 8.4 è $O(n^2)$, in quanto ogni iterazione dei due cicli annidati uno dentro l'altro richiede un numero costante di operazioni.

8.11 Opus libri: il problema del commesso viaggiatore

Concludiamo questo capitolo e il libro con un'ultima opera algoritmica, relativa a uno dei problemi di ottimizzazione più analizzati (in tutte le sue varianti) nel campo dell'informatica e della ricerca operativa, ovvero il **problema del commesso viaggiatore**.

Dato un insieme di città e specificato, per ogni coppia di città, la distanza chilometrica per andare dall'una all'altra o viceversa, un commesso viaggiatore si chiede quale sia il modo più breve per visitare tutte le città una e una sola volta, tornando al termine del giro alla città di partenza.

ESEMPIO 8.11

Consideriamo la seguente istanza del problema in cui 9 città olandesi sono analizzate e in cui le distanze chilometriche sono tratte da una nota guida turistica internazionale:

	A	B	D	E	H	L	M	R	U
Amsterdam	0	101	98	121	20	55	213	73	37
Breda		0	30	57	121	72	146	51	73
Dordrecht			0	92	94	45	181	24	61
Eindhoven				0	136	134	89	113	88
Haarlem					0	51	228	70	54
L'Aia						0	223	21	62
Maastricht							0	202	180
Rotterdam								0	57
Utrecht									0

Se il commesso viaggiatore decide di percorrere le città secondo il loro ordine alfabetico, allora percorre un numero di chilometri pari a

$$101 + 30 + 92 + 136 + 51 + 223 + 202 + 57 + 37 = 929$$

Supponiamo, invece, che decida di percorrerle nel seguente ordine: Amsterdam, Haarlem, L'Aia, Rotterdam, Dordrecht, Breda, Maastricht, Eindhoven e Utrecht. In tal caso, il commesso viaggiatore percorre un numero di chilometri pari a

$$20 + 51 + 21 + 24 + 30 + 146 + 89 + 88 + 37 = 506$$

Mediante una ricerca esaustiva di tutte le possibili $9! = 362880$ permutazioni delle nove città, possiamo verificare che quest'ultima è la soluzione migliore possibile.

Sfortunatamente, il commesso viaggiatore non ha altra scelta che applicare un algoritmo esaustivo per trovare la soluzione al suo problema, in quanto la sua versione decisionale è un problema NP-completo. Più precisamente, consideriamo il seguente problema di decisione: dati un grafo completo $G = (V, E)$, una funzione p che associa a ogni arco del grafo un numero intero non negativo e un numero intero $k \geq 0$, esiste un *tour* del commesso viaggiatore di peso non superiore a k, ovvero un ciclo hamiltoniano in G (Paragrafo 7.1.1) la somma dei cui archi è minore oppure uguale a k?

Per dimostrare che tale problema è NP-completo, consideriamo il problema del circuito hamiltoniano che consiste nel decidere se un grafo qualsiasi include un ciclo hamiltoniano. Utilizzando la tecnica della progettazione di componenti possiamo dimostrare che tale problema è NP-completo.

Teorema 8.8 *Ciclo hamiltoniano è trasformabile in tempo polinomiale nella versione decisionale del problema del commesso viaggiatore.*

Dimostrazione Dato un grafo $G = (V, E)$, definiamo un grafo completo $G' = (V, E')$ e una funzione p tale che, per ogni arco e in E', $p(e) = 1$ se $e \in E$, altrimenti $p(e) = 2$. Scegliendo $k = |V|$, abbiamo che se esiste un ciclo hamiltoniano in G, allora esiste un tour in G' il cui costo è uguale a k. Viceversa, se non esiste un ciclo hamiltoniano in G, allora ogni tour in G' deve includere almeno un arco il cui peso sia pari a 2, per cui ogni tour ha un costo almeno pari a $k + 1$. ❑

Conseguenza del risultato appena provato è che il problema del commesso viaggiatore (nella sua forma decisionale) è NP-completo.

Sfortunatamente, possiamo mostrare che il problema di ottimizzazione non ammette neanche un algoritmo efficiente di approssimazione. A tale scopo, consideriamo nuovamente il problema del circuito hamiltoniano e, facendo uso della tecnica detta del **gap**, dimostriamo che se il problema del commesso viaggiatore ammette un algoritmo efficiente di approssimazione, allora il problema del circuito hamiltoniano è risolvibile in tempo polinomiale.

Teorema 8.9 *Sia* r > 1 *una qualsiasi costante, non esiste alcun algoritmo di* r*-approssimazione di complessità polinomiale per il problema del commesso viaggiator a meno che NP coincida con P.*

Dimostrazione Per assurdo, sia r > 1 una costante e sia A un algoritmo polinomiale di r-approssimazione per il problema del commesso viaggiatore. Dato un grafo G = (V, E), definiamo un grafo completo G′ = (V, E′) e una funzione p tale che, per ogni arco e in E′, p(e) = 1 se e ∈ E, altrimenti p(e) = 1 + s|V| dove s > r – 1. Notiamo che G′ ammette un tour del commesso viaggiatore di costo pari a |V| se e solo se G include un circuito hamiltoniano: infatti, un tale tour deve necessariamente usare archi di peso pari a 1, ovvero archi contenuti in E.

Sia T il tour del commesso viaggiatore che viene restituito da A con G′ in ingresso. Dimostriamo che T può essere usato per decidere se G ammette un ciclo hamiltoniano, distinguendo i seguenti due casi.

1. Il costo di T è uguale a |V|, per cui T è un tour ottimo. Quindi, G ammette un ciclo hamiltoniano.
2. Il costo di T è maggiore di |V|, per cui il suo costo deve essere almeno pari a |V| – 1 + 1 + s|V| = (1 + s)|V| > r|V|. In questo caso, il tour ottimo non può avere costo pari a |V|, in quanto altrimenti il costo di T è maggiore di r volte il costo ottimo, contraddicendo il fatto che A è un algoritmo di r-approssimazione: quindi, non esiste un ciclo hamiltoniano in G.

In conclusione, applicando l'algoritmo A al grafo G′ e verificando se la soluzione restituita da A ha un costo pari oppure maggiore al numero dei vertici, possiamo decidere in tempo polinomiale se G ammette un circuito hamiltoniano, da cui si deduce che il problema del circuito hamiltoniano è in P e quindi P coincide con NP. ❑

8.11.1 Problema del commesso viaggiatore su istanze metriche

Sebbene il problema del commesso viaggiatore non sia, in generale, risolvibile in modo approssimato mediante un algoritmo polinomiale, possiamo mostrare che tale problema, ristretto al caso in cui la funzione che specifica la distanza tra due città soddisfi la disuguaglianza triangolare, ammette un algoritmo di 2-approssimazione.

Un'istanza del problema del commesso viaggiatore soddisfa la **disuguaglianza triangolare** se, per ogni tripla di vertici i, j e k, p(i, j) ≤ p(i, k) + p(k, j): intuitivamente, ciò vuol dire che andare in modo diretto da una città i a una città j non può essere più costoso che andare da i a j passando prima per un'altra città k. L'istanza delle città olandesi dell'Esempio 8.11 soddisfa la disuguaglianza triangolare, anche se tale disuguaglianza non sempre è soddisfatta quando si tratta di distanze stradali.

La disuguaglianza triangolare è invece sempre soddisfatta se i vertici del grafo rappresentano punti del piano euclideo e le distanze tra due vertici corrispondono alle loro distanze nel piano, in quanto in ogni triangolo la lunghezza di un lato è sempre minore della somma delle lunghezze degli altri due lati. Inoltre, la disuguaglianza è soddisfatta nel caso in cui il grafo completo `G` sia ottenuto nel modo seguente, a partire da un grafo `G'` connesso non necessariamente completo: `G` ha gli stessi vertici di `G'` e la distanza tra due suoi vertici è uguale alla lunghezza del cammino minimo tra i corrispondenti vertici di `G'`. Per questo motivo, la risoluzione del problema del commesso viaggiatore, ristretto al caso di istanze che soddisfano la disuguaglianza triangolare, è un problema di per sé interessante che sorge abbastanza naturalmente in diverse aree applicative.

La versione decisionale di tale problema è NP-completo, in quanto la trasformazione a partire dal problema del circuito hamiltoniano nella dimostrazione del Teorema 8.8 genera istanze che soddisfano la disuguaglianza triangolare: se i pesi degli archi sono solo 1 e 2, ovviamente tale disuguaglianza è sempre soddisfatta. Mostriamo ora un algoritmo polinomiale di `2`-approssimazione per il problema del commesso viaggiatore, ristretto al caso di istanze che soddisfano la disuguaglianza triangolare.

L'idea alla base dell'algoritmo è che la somma dei pesi degli archi di un minimo albero ricoprente `R` di un grafo completo `G` costituisce un limite inferiore al costo di un tour ottimo. Infatti, cancellando un arco di un qualsiasi tour `T`, otteniamo un cammino hamiltoniano e, quindi, un albero ricoprente: la somma dei pesi degli archi di questo cammino deve essere, per definizione, non inferiore a quella dei pesi degli archi di `R`. Quindi, il costo di `T` (che include anche il peso dell'arco cancellato) è certamente non inferiore alla somma dei pesi degli archi di `R`.

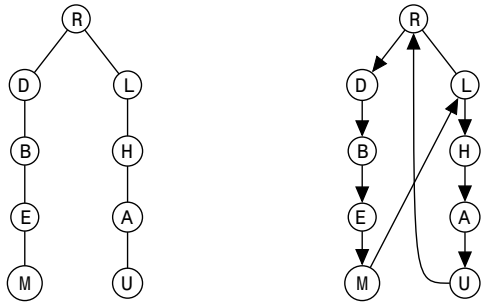
L'algoritmo (realizzato nel Codice 8.5) costruisce un minimo albero ricoprente di `G` (riga 3) e restituisce i nodi in un array `visitato` in cui questi compaiono secondo l'ordine di una visita DFS. Si ottiene un tour del grafo supponendo implicitamente che il vertice in ultima posizione (ovvero, posizione `n - 1`) deve essere connesso a quello in prima posizione (ovvero, posizione `0`).

ALVIE Codice 8.5 Algoritmo per il calcolo di un tour approssimato del commesso viaggiatore.

```
1  CommessoViaggiatore( P ):
2       ⟨pre: P è la matrice di adiacenza e dei pesi di un grafo G completo di n nodi⟩
3  mst = Jarník-Prim( P );
4  visitato = DFS( mst );
5  RETURN visitato;
```

ESEMPIO 8.12

Nella parte sinistra della seguente figura è rappresentato il minimo albero ricoprente del grafo dell'Esempio 8.11.



Se la visita dell'albero inizia dal nodo R, il Codice 8.5 restituisce la sequenza (R, D, B, E, M, L, H, A, U) che corrisponde al tour mostrato nella parte destra della figura in alto evidenziato con archi diretti.

Poiché il calcolo del minimo albero ricoprente e l'esecuzione della visita dell'albero possono essere realizzati in $O(n^2 \log n)$, abbiamo che l'algoritmo appena descritto è polinomiale. Inoltre vale il seguente risultato.

Teorema 8.10 *Il Codice 8.5 è un algoritmo di* 2*-approssimazione per il problema del commesso viaggiatore metrico.*

Dimostrazione Quello restituito è un tour in quanto ogni nodo compare una volta soltanto e nodi consecutivi sono collegati da un arco in quanto il grafo è completo.

Ora consideriamo un arco $e = (u, v)$ del tour T che non appartiene al minimo albero ricoprente R: questo è un arco trasversale oppure, nel caso in cui sia l'ultimo arco del tour, all'indietro (si veda il Paragrafo 7.2.2). Sia R_e l'insieme degli archi appartenenti al percorso tra u e v nel minimo albero di ricoprimento. A causa della disuguaglianza triangolare la somma dei pesi degli archi in R_e non può essere inferiore al peso dell'arco e. Inoltre se e' è un altro arco di T ma non in R e diverso da e, allora $R_e \cap R_{e'} = \phi$, in quanto, altrimenti, l'algoritmo DFS avrebbe visitato una componente dell'albero più di una volta. Riassumendo

$$\sum_{e \in T} p(e) \leq \sum_{e \in R} p(e) + \sum_{e \in T - R} p(e) \leq \sum_{e \in R} p(e) + \sum_{e \in R_e} p(e) \leq 2 \sum_{e \in R} p(e).$$

Dove l'ultima disuguaglianza segue perché gli R_e sono disgiunti e la penultima dalla disuguaglianza triangolare. Infine giungiamo al risultato cercato tenendo conto che il costo del minimo albero di ricoprimento non può essere maggiore del costo del tour ottimale. ❑

Per concludere, osserviamo che l'algoritmo di approssimazione realizzato dal Codice 8.5 non è il migliore possibile. In effetti, con un opportuno accorgimento nello scegliere gli archi del minimo albero ricoprente da duplicare, possiamo modificare tale algoritmo ottenendone uno di `1,5`-approssimazione. Inoltre, nel caso di istanze formate da punti sul piano euclideo, possiamo dimostrare che, per ogni `r > 1`, esiste un algoritmo polinomiale di `r`-approssimazione: in altre parole, il problema del commesso viaggiatore sul piano può essere approssimato tanto bene quanto vogliamo (ovviamente al prezzo di una complessità temporale che, pur mantenendosi polinomiale, cresce al diminuire di `r`).

8.11.2 Paradigma della ricerca locale

Un algoritmo per la risoluzione di un problema di ottimizzazione basato sul **paradigma della ricerca locale** opera nel modo seguente: a partire da una soluzione iniziale del problema, esplora un insieme di soluzioni "vicine" a quella corrente e si sposta in una soluzione che è migliore di quella corrente, fino a quando non giunge a una che non ha nessuna soluzione vicina migliore. Pertanto, il comportamento di un tale algoritmo dipende dalla nozione di vicinato di una soluzione (solitamente generato applicando operazioni di cambiamento locale alla soluzione corrente), dalla soluzione iniziale (che può essere calcolata mediante un altro algoritmo) e dalla strategia di selezione delle soluzioni (per esempio, scegliendo la prima soluzione vicina migliore di quella corrente oppure selezionando la migliore tra tutte quelle vicine alla corrente).

Non esistono regole generali per decidere quali siano le regole di comportamento migliori: per questo, ci limitiamo in questo paragrafo finale a descrivere due algoritmi basati sul paradigma della ricerca locale per la risoluzione (non ottima) del problema del commesso viaggiatore. Notiamo sin d'ora che non siamo praticamente in grado di formulare nessuna affermazione (non banale) relativamente alle prestazioni di questi algoritmi né in termini di complessità temporale né in termini di qualità della soluzione ottenuta. Tuttavia, questo tipo di strategie (dette anche **euristiche**) risultano nella pratica estremamente valide e, per questo, molto utilizzate.

Entrambi gli algoritmi che descriviamo fanno riferimento a operazioni locali di cambiamento: essi tuttavia differiscono tra di loro per quello che riguarda la lunghezza massima della sequenza di tali operazioni. In particolare, i due algoritmi modificano la soluzione corrente selezionando un numero fissato di archi e sostituendoli con un altro insieme di archi (della stessa cardinalità) in modo da ottenere un nuovo tour.

Il primo algoritmo, detto **2-opt**, opera nel modo seguente: dato un tour `T` del commesso viaggiatore, il suo vicinato è costituito da tutti i tour che possono essere ottenuti cancellando due archi `(x, y)` e `(u, v)` di `T` e sostituendoli con due nuovi archi `(x, u)` e `(y, v)` in modo da ottenere un tour differente `T'` (notiamo che