A proposito di Commesso viaggiatore: programmazione dinamica

Come definire dei sottoproblemi?

un ciclo di costo minimo a partire dal vertice 1, puo' essere definito come la concatenazione di un cammino minimo dal vertice 1 a un vertice i che attraversa una e una sola volta ogni vertice in un insieme X e un cammino minimo dal vertice i al vertice 1, che attraversa una e una sola volta i vertici nell'insieme $S = V - (X \cup \{i,1\})$

Chiamiamo g(i, S) il costo di un cammino di costo minimo dal vertice i al vertice 1 che attraversa una e una sola volta ogni vertice nell'insieme S.

$g(1, V - \{1\})$ definisce allora il costo di un giro ottimo.

Per il principio di ottimalità si può definire, per $i \notin S$:

$$g(i, S) = \min_{j \in S} \{d[i, j] + g(j, S - \{j\})\}, \quad S \neq \Phi$$

$$g(i, \Phi) = d[i, 1]$$

Da cui si ottiene:

$$g(1, V - \{1\}) = \min_{2 \leq k \leq n} \{d[1, k] + g(k, V - \{1, k\})\}$$

Commesso viaggiatore: programmazione dinamica

Esempio:

	1	2	3	4
1	0	20	5	8
2	20	0	6	2
3	5	6	0	4
4	8	2	4	0

Calcoliamo la funzione g per il grafo:

Usiamo $d_{i,j}$ come abbreviazione per d[i, j]

$g(2, \Phi) = d_{2,1} = 20$
$g(3, \Phi) = d_{3,1} = 5$
$g(4, \Phi) = d_{4,1} = 8$

$g(2, \{3\}) = d_{2,3} + g(3, \Phi) = 11$
$g(3, \{2\}) = d_{3,2} + g(2, \Phi) = 26$
$g(4, \{2\}) = d_{4,2} + g(2, \Phi) = 22$

$g(2, \{4\}) = d_{2,4} + g(4, \Phi) = 10$
$g(3, \{4\}) = d_{3,4} + g(4, \Phi) = 12$
$g(4, \{3\}) = d_{4,3} + g(3, \Phi) = 9$

$g(2, \{3,4\}) = \min \{ d_{2,3} + g(3, \{4\}, d_{2,4} + g(4, \{3\}) \} = 11$
$g(3, \{2,4\}) = \min \{ d_{3,2} + g(2, \{4\}, d_{3,4} + g(4, \{2\}) \} = 16$
$g(4, \{2,3\}) = \min \{ d_{4,2} + g(2, \{3\}, d_{4,3} + g(3, \{2\}) \} = 13$

$g(1, \{2,3,4\}) = \min \{d_{1,2}+ g(2, \{3,4\}, d_{1,3}+ g(3, \{2,4\}), d_{1,4}+ g(4, \{2,3\})\} = 21$

Commesso viaggiatore: programmazione dinamica

Soluzione ricorsiva con memorizzazione.
gtab(i,S): lunghezza minima di un cammino semplice da "i" a "1" in "S"

```
TSP_Memoized (parametri opportuni)
    for k ← 1 to n
        for ogni S ⊂ V
                gtab[k, S] ← -1    \\Gli elementi di gtab sono inizializzati a -1
    gtab[i, Φ) ← d[i, 1]     \\arco diretto se S = Φ
    return Lookup_TSP (1, V – {1})

Lookup_TSP (i, S)
        if (S = Φ) return d[i, 1]
        if (gtab[i, S] >= 0) return gtab[i, S]   \\gia'  calcolato
        costo ← ∞
        for ogni j ∈ S
                distviaj ← d[i, j] + Lookup_TSP (j, S - {j})
                if (distviaj < costo)  costo ← distviaj
        gtab[i, S] ← costo
        return costo
```

Commesso viaggiatore: programmazione dinamica

L'algoritmo di programmazione dinamica che si basa sul calcolo della funzione g richiede un gran numero di confronti.

- Calcoliamo il numero di g(i, S) con $|S| = k$ e $i \neq 1$:

  ci sono (n-1) scelte per i e il numero di insiemi S che non includono 1 e i è $\binom{n-2}{k}$, perciò il numero di g(i, S) per un dato valore di k è: $(n-1)\binom{n-2}{k}$
- k assume i valori da 1 a n-2
- per ogni S di cardinalità k vengono effettuati k-1 confronti
- (n-2) sono i confronti richiesti per calcolare g(1,V - {1})

Numero di confronti per calcolare g(1,V-{1})

Numero di scelte per i

$$\boxed{(n-2)} + \sum_{\boxed{k=1}}^{\boxed{n-2}} \boxed{(n-1)}\,\boxed{(k-1)}\,\boxed{\binom{n-2}{k}} \qquad \Theta(n^2\, 2^n)$$

In S ci sono tutti i vertici tranne "i"e "1"

Numero di confronti

Numero di insiemi di cardinalita' k

Ricordiamo che $\sum_{k=0}^{n} \binom{n}{k} = 2^n.$

Commesso viaggiatore: complessita`

Algoritmo di Backtracking:

La complessità riferita al caso peggiore è $\Theta(n!)$. Con 15 nodi il tempo di attesa per la soluzione è di circa 12 ore. Con 16 nodi ci vogliono circa 7 giorni.

Algoritmo Branch & Bound:

Migliorano i tempi di risposta nei casi medi; il calcolo della funzione “lower bound” è lineare in n, ma il numero di cammini da generare nel caso peggiore resta n!.

Algoritmo di programmazione dinamica

Un algoritmo che si basi sul calcolo della funzione g avrà complessità

$$\Theta(n^2\, 2^n)$$

Meglio di $\Theta(n!)$, ma sempre un tempo che aumenta in modo drammatico all’aumentare del numero di vertici.

Per l’algoritmo di programmazione dinamica è inoltre critica l’occupazione di spazio: $\mathbf{O}(n\, 2^n)$