Ricerca Operativa
A.A. 2007/2008

19. Cenni su euristiche e metaeuristiche per ottimizzazione combinatoria

1

Motivazioni

- L'applicazione di metodi esatti non è sempre possibile a causa della complessità del problema
  - Non sempre è possibile ottenere un modello che sia sufficientemente accurato e maneggevole
  - Non sempre è possibile determinare una procedura "polinomiale" che individui la soluzione ottima
- È comunque importante trovare una "buona" soluzione (anche nell'ambito di procedure esatte come il Branch and Bound)

Classificazione (una possibile)

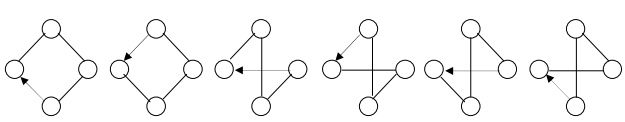
- Metodi euristici (dipendenti dal problema)
  - Euristiche costruttive
  - Metodi iterativi
  - …
- Algoritmi approssimati (metodi euristici con performance garantite in termini di qualità della soluzione trovata)
- Meta-euristiche (metodologie generali, schemi algoritmici)
  - Semplificazioni di schemi esatti
  - Algoritmi basati su ricerca locale
  - Algoritmi evolutivi
  - Meta-euristiche ibride
  - …
- Nota: nel seguito si assumeranno problemi di minimizzazione

Il problema del commesso viaggiatore (Traveling Salesman Problem - TSP)

Date *n* città, determinare il percorso che passi per le città una ed una sola volta (*ciclo hamiltoniano*) e che minimizzi i costi/tempi/distanze di viaggio.

Problema di ottimizzazione combinatoria

Es. 4 città ⇒ 6 possibili giri



In generale *n* città ⇒ (n-1)! sequenze (tra le quali ricercare l'ottimo)

TSP – Una formulazione PLI

- Modelliamo il problema su un grafo G=(N,A).
  - N: insieme delle *n* città;
  - A: insieme dei collegamenti con costo c_{ij} (= ∞ se il collegamento non esiste).
- Variabile x_{ij}: 1 se l'arco (i,j) fa parte del circuito hamiltoniano, 0 altrimenti

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

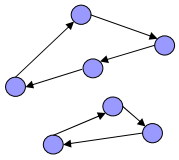
$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n$$

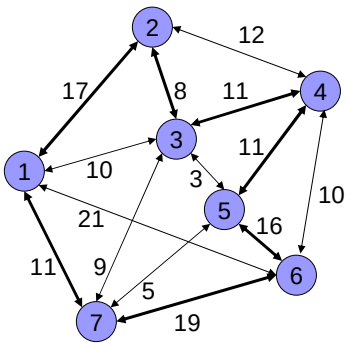
$$\sum_{i,j\in S} x_{ij} \le |S| - 1 \qquad \forall\, S \subseteq N : 2 \le |S| \le n-1$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i,j = 1, \ldots, n$$

Numero esponenziale di vincoli per eliminare soluzioni con sotto-cicli (sub-tour elimination)



TSP – Un'istanza



Note

- $c_{ij} = c_{ji}$ (TSP **simmetrico**)
- $c_{ij} = \infty$ se l'arco (i,j) non esiste
- si fissa 1 come città di partenza

Un algoritmo euristico (NN – nearest neighbor)

1. Si parta dalla città 1: i ← 1
2. Si visiti la città non visitata j che minimizza c_{ij}
3. Se tutte le città sono visitate, chiudere il ciclo con l'arco (j,1). STOP.
4. Porre i ← j e tornare al passo 2.

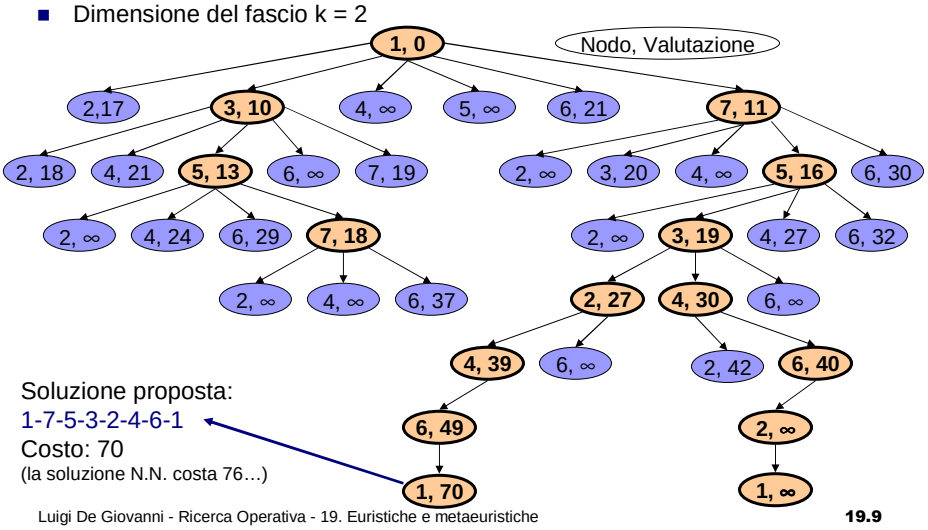
- Euristica costruttiva: ad ogni passo si aggiunge una città
- di tipo greedy (miope): aggiunge la città che garantisce "localmente" il minor aumento della funzione obiettivo
- Molto veloce – complessità $O(n^2)$ - ma scarsa qualità dei risultati: esistono molte altre euristiche migliori…

Un altro algoritmo euristico Beam search

- Semplifica il branch & bound: si parte da un albero di enumerazione ma si contiene l'esplosione combinatoria del numero di nodi ad ogni livello
- Fissa il numero di nodi da elaborare ad ogni livello (dimensione del fascio): ad ogni livello si sviluppano solo un numero limitato di nodi
- Complessità computazionale:
  - A: numero di livelli dell'albero
  - B: numero di figli del nodo generico
  - k: dimensione del fascio
  - V: complessità computazionale per la valutazione di un singolo nodo
  - Numero di nodi per livello: O(kB)
  - Numero totale di nodi valutati: O(AkB)
  - Complessità computazionale: O(AkBV) – polinomiale! (se V polinomiale)

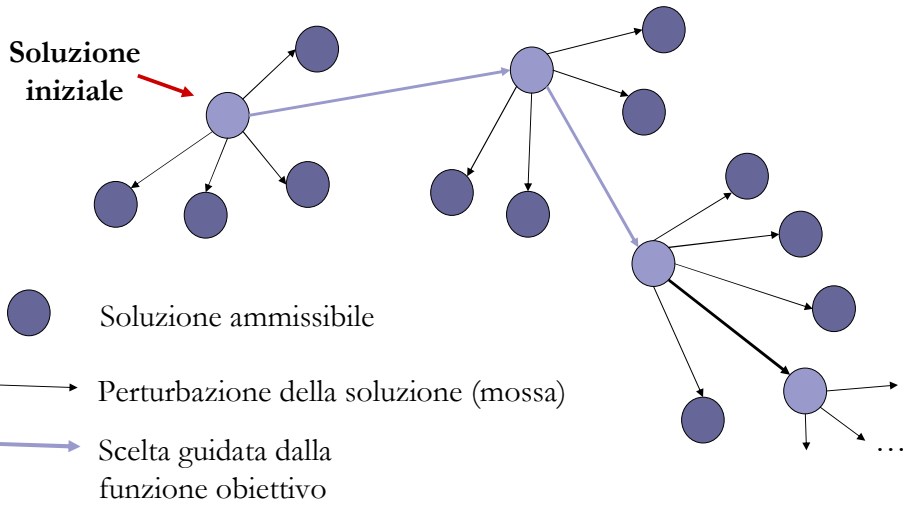
Esempio: Beam Search per TSP

- A = *n*: ad ogni livello si sceglie la prossima città da visitare
- Branching n-ario (B < *n*): un figlio per ogni città ancora da visitare.
- Valutazione del nodo: costo fino a qual momento
- Dimensione del fascio k = 2



Soluzione proposta:
1-7-5-3-2-4-6-1
Costo: 70
(la soluzione N.N. costa 76…)

Ricerca Locale



Soluzione ammissibile

Perturbazione della soluzione (mossa)

Scelta guidata dalla funzione obiettivo

Ricerca locale: schema

1. Si parte da una soluzione ammissibile (nota o ottenuta, ad es., con euristica)
2. Si definisce un vicinato della soluzione corrente: insieme di soluzioni ottenibili con una perturbazione (mossa)
3. Se esiste un vicino che migliora la f.o., si sceglie il primo (*first improvement-F.I.*) o il migliore (*steepest descent-S.D.*) come centro del prossimo vicinato e si va a 2.
4. Altrimenti STOP: la soluzione corrente è un minimo locale

Ricerca locale: elementi

Per lo specifico problema bisogna definire

1. Rappresentazione della soluzione
   - Importante: condiziona tutto il resto!
2. Soluzione iniziale
   - Efficienza, qualità, possibilità di migliorarla con la successiva ricerca
3. Struttura del vicinato (mosse)
   - Possibilità di miglioramento della soluzione corrente .vs. dimensioni ed efficienza computazionale
4. Strategia di discesa (F.I. oppure S.D.)

Ricerca Locale per TSP

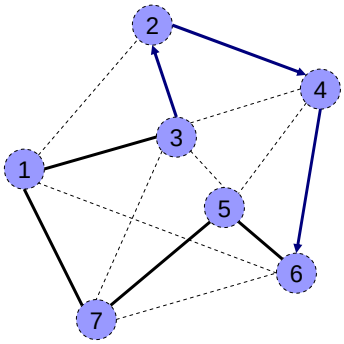
Una possibile scelta implementativa:

1. Rappresentazione della soluzione: sequenza di città
2. Soluzione iniziale: NN
3. **Vicinato 2-OPT**: scambio una coppia di città nella sequenza di partenza e percorro in senso inverso tra le due città (*sub-tour reversal*). Corrisponde a scambiare due archi (non consecutivi) nella soluzione con due non nella soluzione. *$O(n^2)$ vicini, ciascuno con tempo di valutazione costante.*
4. Strategia di discesa S.D.

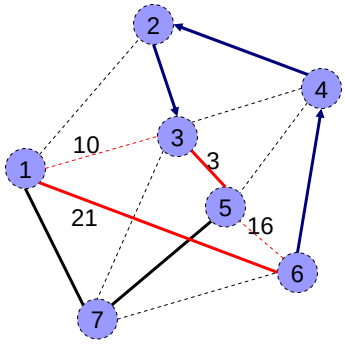
Esempio: vicinato 2-OPT

Es: 1-**3**-2-4-**6**-5-7-1 ⇒ 1-**6**-4-2-**3**-5-7-1

Corrisponde a eliminare (1,3) e (6,5) e aggiungere (1,6) e (3,5)



C = 72



C = 72 – 10 – 16 + 21 + 3 = 70

Con TSP simmetrico, la valutazione di un vicino si ottiene in tempo costante: se si inverte la sequenza tra i e j: $C_{new} = C_{old} - c_{(i-1),i} - c_{j,(j+1)} + c_{(i-1),j} + c_{i,(j+1)}$

Tenendo fissa la città di arrivo-partenza, il numero di vicini è (n-1) * (n-2) / 2

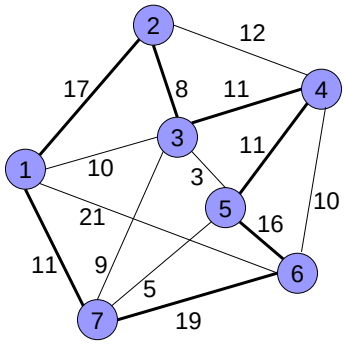
Esempio

Applicare la ricerca locale con vicinato 2-OPT, strategia F.I. a partire dalla soluzione 1-2-3-4-5-6-7-1 (costo 93) per determinare una soluzione per l'istanza di TSP simmetrico data.

Nota: gli estremi della stringa (1…1) sono fissati

Al primo passo i vicini sono (il costo tra parentesi)

1-2-3-4-5-6-7-1 ⇒ 1-3-2-4-5-6-7-1 (93-17-11+10+12 = 87)
1-2-3-4-5-6-7-1 ⇒ 1-4-3-2-5-6-7-1 (93-17-11+∞+∞ = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-5-4-3-2-6-7-1 (93-17-16+∞+∞ = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-6-5-4-3-2-7-1 (93-17-19+21+∞ = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-7-6-5-4-3-2-1 (93-17-11+11+17 = 93)
1-2-3-4-5-6-7-1 ⇒ 1-2-4-3-5-6-7-1 (93-8-11+12+3 = 89)
1-2-3-4-5-6-7-1 ⇒ 1-2-5-4-3-6-7-1 (93-8-16+∞+∞ = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-2-6-5-4-3-7-1 (93-8-19+∞+9 = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-2-7-6-5-4-3-1 (93-8-11+∞+10 = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-2-3-5-4-6-7-1 (93-11-16+3+10 = 79)
1-2-3-4-5-6-7-1 ⇒ 1-2-3-6-5-4-7-1 (93-11-19+∞+∞ = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-2-3-7-6-5-4-1 (93-11-11+9+∞ = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-2-3-4-6-5-7-1 (93-11-19+10+5 = 78)
1-2-3-4-5-6-7-1 ⇒ 1-2-3-4-7-6-5-1 (93-11-11+∞+∞ = ∞)
1-2-3-4-5-6-7-1 ⇒ 1-2-3-4-5-7-6-1 (93-16-11+5+21=92)



Prossimo nodo corrente cui applicare 2-OPT:

- F.I.: 1-3-2-4-5-6-7-1 (e non si valutano i successivi vicini!) al costo 87 < 93.
- S.D.: si valutano tutti i 15 vicini e si sceglie 1-2-3-4-6-5-7-1 al costo 78.

Minimi locali…

- La ricerca locale può rimanere intrappolata in minimi locali

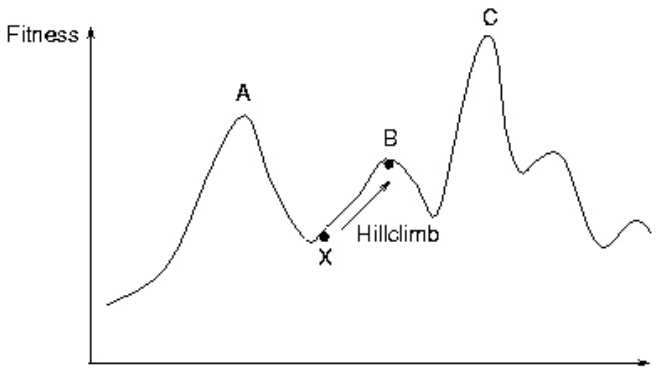


Figure 6: The hillclimbing problem

Metaeuristiche basate su ricerca locale

Come superare i minimi locali?

- Randomized Multistart: si applica ripetutamente una ricerca locale a partire da soluzioni iniziali generate casualmente
- **GRASP** (Greedy Randomized Adaptive Search Procedure): come multistart, ma le soluzioni iniziali derivano da una procedura costruttiva *randomizzata* e *dinamica*
- **Tabu search**: schema della ricerca locale S.D. ma accetta anche soluzioni non miglioranti. Memoria (tabu list delle mosse –vicini- proibite) per evitare di ciclare sugli stessi minimi locali
- Simulated annealing (raffreddamento): sceglie un vicino a caso e lo accetta sempre se migliorante e con una certa probabilità $e^{-(Z-Z')/T}$ se non migliorante. Tale probabilità diminuisce con il numero di iterazioni, agendo in diminuzione sul parametro *temperatura* T (profili di raffreddamento).

Algoritmi genetici

Definisce una popolazione iniziale e applica iterativamente i seguenti operatori

- Selezione: determina le coppie di genitori sulla base della fitness (≈ f.o.)
- Crossover: determina la formazione dei figli da inserire nella popolazione
- Mutazione: inserisce delle variazioni casuali nel processo di crossover o nei figli
- Aggiornamento della popolazione inserendo i nuovi individui ed eliminando i peggiori (generational replacement, elitismo, steady state …).

Esempi

- Single point cross-over

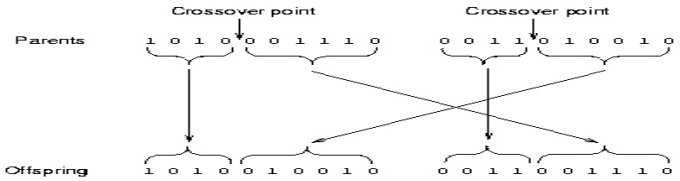


Figure 2: Single-point Crossover

- Mutazione

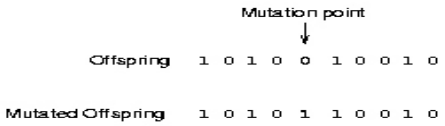


Figure 3: A single mutation

Algoritmo genetico per TSP

- Codifica degli individui (cromosomi=sequenza di geni): ogni gene una città
- Decodifica degli individui: si visitano le città nell'ordine definito dal cromosoma, ottenendo la fitness come 1/(costo totale)
- Popolazione iniziale: si generano casualmente *N* sequenze
- Selezione: vari metodi; in generale si tende a privilegiare i migliori senza escludere gli altri
- Crossover: order crossover (preserva ammissibilità)
- Mutazione: si scambiano a caso due città nel nuovo individuo
- Aggiornamento: si inseriscono i nuovi individui nella popolazione e si mantengono i migliori *N*

Attenzione:

- Le metaeuristiche si devono applicare quando un metodo esatto non sia praticabile o conveniente
- Attenzione alle mode: cercare la metaeuristica più adatta al problema da risolvere, che non è sempre la più immediata (o… gli algoritmi genetici…)
- Le metaeuristiche sono in continua evoluzione:
  - intensificazione e diversificazione
  - ibridizzazione di metaeuristiche (memetic algorithms)
  - metaeuristiche parallele
  - …
- Un riferimento tra i tanti: C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268-308, 2003.