Ricerca locale (1/2)

Indice (1/2)

Perché la ricerca locale?

Ricerca locale

I problemi, le ipotesi e le soluzioni di partenza
- Soluzioni di partenza

Ricerca locale: lo schema

Intorni per i quattro problemi
- KP: flip e swap
- BPP: move e swap
- TSP: 2-opt e Or-opt
- $P \,||\, C_{\max}$: move e swap
- Riepilogo

Indice (2/2)

First improvement e best improvement
- First improvement
- Best improvement
- Confronto

Valutare una mossa in modo incrementale

TSP: 2-opt e Or-opt
- La ricerca locale 2-opt
- La ricerca locale Or-opt
- First e best improvement su un'istanza media
- 3-opt e Lin–Kernighan

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- definire gli intorni del KP, del BPP, del TSP e di $P \,||\, C_{\max}$;
- eseguire una ricerca locale con le regole first improvement e best improvement;
- valutare una mossa in modo incrementale, in $O(1)$ per la mossa 2-opt;
- uscire da un plateau con un obiettivo ausiliario, e gestire le soluzioni non ammissibili con una penalità;
- combinare più intorni (variable neighbourhood descent) e più ripartenze (multi-start).

Prerequisiti

- Da questo modulo, il capitolo «Algoritmi euristici: definizioni»: intorni, ottimi locali, intensificazione e diversificazione.
- Da questo modulo, il capitolo «Euristiche costruttive»: le soluzioni di partenza usate qui.

Perché la ricerca locale?

Introduzione: Perché la ricerca locale?

- Le euristiche costruttive danno una prima soluzione ammissibile, spesso lontana dall'ottimo
- Non tornano mai su una scelta: niente in esse può correggere una decisione sbagliata
- Questo capitolo fa il passo successivo: partiamo da una soluzione e cerchiamo di migliorarla

Dalla costruzione al miglioramento

- Un'euristica costruttiva costruisce una soluzione un elemento alla volta e non rivede mai una scelta già fatta.
- Questo capitolo studia il passo successivo: **come migliorare una soluzione che abbiamo già**, modificandola un poco alla volta.
- L'euristica costruttiva dà la soluzione di partenza; la ricerca locale la migliora.

Una domanda da tenere a mente

Le soluzioni di partenza delle euristiche costruttive, sulle istanze di riferimento del corso, e i valori ottimi:

problema	euristica	valore	$z(I)$
KP	*GR*	16	17
BPP	*FF*	4	3
TSP	*NN*	25	18
$P \mid\mid C_{\max}$	*LPT*	11	9

- Piccole modifiche di queste soluzioni possono chiudere i gap?
- Il capitolo risponde su ciascuno dei quattro problemi.

Riepilogo

Dalla costruzione al miglioramento:

- Un'euristica costruttiva non rivede mai le sue scelte
- La ricerca locale migliora una soluzione che abbiamo già, un poco alla volta

La domanda:

- Piccole modifiche possono chiudere il gap fra le soluzioni di partenza e i valori ottimi?

Ricerca locale

Introduzione: Ricerca locale

- In questa sezione definiamo una ricerca locale, l'euristica di miglioramento di base
- Si basa sulle nozioni di intorno, mossa e ottimo locale
- Tutti gli algoritmi del capitolo sono ricerche locali, o combinazioni di ricerche locali

Definizione 1: Ricerca locale

*Dato un intorno N, una **ricerca locale** (indicata con LS) è un'euristica di miglioramento. Parte da una soluzione ammissibile e, a ogni iterazione, cerca nell'intorno della soluzione corrente una soluzione migliore. Se la trova, si sposta su di essa e ripete; se nessun vicino è migliore, la soluzione corrente è un ottimo locale rispetto a N e la ricerca si ferma.*

- L'**intorno** $N(S)$ di una soluzione ammissibile S, la **mossa** da S a un vicino e l'**ottimo locale** rispetto a N sono definiti nel capitolo «Algoritmi euristici: definizioni».

La ricerca locale in una frase

Una ricerca locale migliora una soluzione ammissibile spostandosi da una soluzione a un vicino migliore,

e si ferma in un ottimo locale rispetto a N.

Riepilogo

Ricerca locale:

- Un'euristica di miglioramento: parte da una soluzione ammissibile
- A ogni iterazione si sposta su un vicino migliore, se esiste
- Si ferma in un ottimo locale rispetto all'intorno N

I problemi, le ipotesi e le soluzioni di partenza

Introduzione: I problemi, le ipotesi e le soluzioni di partenza

- Richiamiamo i quattro problemi su cui sono illustrati gli algoritmi, e la loro notazione
- Richiamiamo le istanze di riferimento e le soluzioni di partenza calcolate dalle euristiche costruttive
- Spieghiamo come si generano le soluzioni di partenza casuali, in modo che ogni esperimento si possa ripetere

I quattro problemi

- Gli algoritmi sono illustrati sui quattro problemi del capitolo «Algoritmi euristici: definizioni»: il problema dello zaino 0/1 (KP), il problema del bin packing (BPP), il problema del commesso viaggiatore (TSP) e $P \,||\, C_{\max}$.

- Le loro definizioni e ipotesi sono nelle rispettive pagine; la prossima tabella richiama la notazione.

La notazione dei quattro problemi

problema	dati	soluzione	obiettivo
KP	p_j, w_j $(j \in \mathscr{I})$, capacità c	vettore binario x	max profitto
BPP	w_j $(j \in \mathscr{I})$, capacità c	vettore *bin*, schemi P_b	min numero di contenitori *nb*
TSP	d_{ij} $(i, j \in \mathscr{V})$	giro σ	min lunghezza $L(\sigma)$
$P \mid\mid C_{\max}$	t_j $(j \in \mathscr{J})$, m macchine	vettore *mach*, insiemi J_h	min makespan $C_{\max}$

Carichi e macchine critiche

- Per $P \,||\, C_{\max}$, $L_h = \sum_{j \in J_h} t_j$ è il **carico** della macchina h e

$$C_{\max} = \max_h L_h.$$

- Per il BPP, $L_b = \sum_{j \in P_b} w_j$ è il **carico** del contenitore b.

- Una macchina (o un contenitore) il cui carico è uguale al valore obiettivo si dice **critica** (o critico).

Le istanze di riferimento

Le **istanze di riferimento** sono quelle del capitolo «Algoritmi euristici: definizioni»:

- il KP con $p = (10, 7, 6, 4)$, $w = (5, 4, 3, 3)$, $c = 9$;
- il BPP con $w = (4, 4, 5, 3, 2, 3)$, $c = 7$;
- il TSP con 5 città;
- $P \,||\, C_{\max}$ con $t = (5, 5, 4, 4, 3, 3, 3)$ e $m = 3$.

I loro valori ottimi sono 17, 3, 18 e 9.

Soluzioni di partenza

Una ricerca locale ha bisogno di una **soluzione di partenza** ammissibile. Usiamo le soluzioni calcolate nel capitolo «Euristiche costruttive»:

problema	algoritmo	soluzione di partenza	valore	$z(I)$
KP	*GR* (greedy per efficienza)	$x = (1, 0, 1, 0)$, oggetti $\{1, 3\}$	16	17
KP	greedy per peso GR_w	$x = (0, 0, 1, 1)$, oggetti $\{3, 4\}$	10	17
BPP	*NF*	$\{1\}, \{2\}, \{3\}, \{4, 5\}, \{6\}$	5	3
BPP	*FF*	$\{1, 4\}, \{2, 5\}, \{3\}, \{6\}$	4	3
BPP	*FFD*	$\{3, 5\}, \{1, 4\}, \{2, 6\}$	3	3
TSP	*NN* dalla città 1	$\sigma = (1, 3, 2, 4, 5)$	25	18
$P \mid\mid C_{\max}$	*LPT*	$J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$	11	9

Quali soluzioni di partenza

- La soluzione di *FFD* è già ottima, quindi nessuna ricerca locale può migliorarla: per il BPP partiamo dalle soluzioni di *NF* e *FF*.
- Per studiare il comportamento di una ricerca locale su molte soluzioni di partenza usiamo anche **soluzioni di partenza casuali**.

Soluzioni di partenza casuali

- Le soluzioni di partenza casuali sono generate con il generatore minimal standard di Park–Miller: dato un **seme** intero $s_0 \in \{1, \ldots, 2^{31} - 2\}$, il k-esimo numero pseudo-casuale è

$$s_k = 16807 \cdot s_{k-1} \bmod (2^{31} - 1), \qquad u_k = \frac{s_k}{2^{31} - 1} \in (0, 1),$$

  e un intero pseudo-casuale in $\{0, 1, \ldots, q - 1\}$ è $\lfloor q\, u_k \rfloor$.

- La sequenza è interamente determinata dal seme: lo stesso seme dà le stesse soluzioni di partenza, nelle note, negli esercizi e nel codice.

Giri, assegnazioni e soluzioni del BPP casuali

- Un **giro casuale** tiene la città 1 in prima posizione e rimescola le altre città (Fisher–Yates: per $k = n, n-1, \ldots, 3$, la città in posizione k si scambia con la città in una posizione casuale fra $2, \ldots, k$).

- Un'**assegnazione casuale** dei lavori estrae la macchina dei lavori $1, 2, \ldots, n$ in quest'ordine.

- Una **soluzione casuale del BPP** si calcola con *FF* su un ordine casuale degli oggetti (Fisher–Yates su tutte le posizioni: per $k = n, n-1, \ldots, 2$, l'oggetto in posizione k si scambia con l'oggetto in una posizione casuale fra $1, \ldots, k$).

Riepilogo

I problemi:

- KP, BPP, TSP e $P \,||\, C_{\max}$, con i carichi L_b, L_h e i contenitori e le macchine critici
- Istanze di riferimento con valori ottimi 17, 3, 18 e 9

Le soluzioni di partenza:

- Le soluzioni delle euristiche costruttive: GR, GR_w, NF, FF, NN, LPT
- Soluzioni casuali dal generatore di Park–Miller: lo stesso seme dà le stesse soluzioni

Ricerca locale: lo schema

Introduzione: Ricerca locale: lo schema

- Scriviamo lo pseudocodice di una ricerca locale ed elenchiamo i suoi quattro ingredienti
- Dimostriamo che una ricerca locale si ferma sempre, in un ottimo locale
- Vediamo quale ottimo locale raggiunge: il fondo della valle in cui parte

Lo pseudocodice di *LS* è il seguente.

La funzione *Improve* esplora l'intorno
e restituisce un vicino migliorante di x,
oppure *NULL* se non ne esiste nessuno.

Algorithm: Ricerca locale

Input : una soluzione ammissibile x^0, un intorno N, una regola $\in \{first, best\}$

Output : un ottimo locale x rispetto a N

```
1  function LocalSearch(x^0, N, rule)
2      x := x^0                                  // soluzione corrente
3      while true do
4          y := Improve(x, N, rule)   // vicino migliorante,
             oppure NULL
5          if y = NULL then
6              return x     // nessun vicino migliorante:
                 ottimo locale
7          end
8          x ← y    // spostarsi sul vicino migliorante
9      end
10 end
```

I quattro ingredienti di una ricerca locale

1. La **soluzione di partenza** x^0, calcolata da un'euristica costruttiva o a caso.
2. L'**intorno** N, cioè la famiglia delle mosse, che dipende dalla rappresentazione delle soluzioni.
3. Il **criterio di accettazione**: qui un vicino è accettato solo se è strettamente migliore della soluzione corrente.
4. Il **criterio di arresto**: qui la ricerca si ferma nel primo ottimo locale; in pratica si aggiunge spesso un limite di tempo.

La funzione che guida la ricerca

- La funzione f che decide se un vicino è "migliore" è di solito l'obiettivo del problema.
- Ma non è obbligatorio: la ricerca può essere guidata da una funzione diversa, purché la soluzione restituita sia ammissibile.
- Useremo questa libertà più avanti nel capitolo, con gli obiettivi ausiliari e le penalità.

Una ricerca locale si ferma sempre?

E quante mosse può eseguire?

Proposizione 1

Per ogni soluzione di partenza, LS si ferma dopo un numero finito di mosse, in un ottimo locale rispetto a N. Se la funzione obiettivo assume valori interi e il problema è di minimizzazione, il numero di mosse è al più $f(x^0) - z(I)$; per un problema di massimizzazione, al più $z(I) - f(x^0)$.

Dimostrazione

- Ogni mossa migliora strettamente l'obiettivo, quindi la sequenza delle soluzioni visitate non visita mai due volte la stessa soluzione.
- L'insieme delle soluzioni ammissibili è finito, quindi la sequenza è finita.
- Quando si ferma, *Improve* ha restituito *NULL*, cioè nessun vicino è migliore: l'ultima soluzione è un ottimo locale.
- Se f assume valori interi, ogni mossa migliora f di almeno 1, e f non può mai andare oltre $z(I)$: da qui i limiti sul numero di mosse.

□

Quante mosse?

- Sull'istanza di riferimento del TSP, una ricerca locale che parte dal giro di *NN* (lunghezza 25) esegue al più $25 - 18 = 7$ mosse.
- Sulle istanze grandi questo limite è debole, e per alcuni intorni esistono istanze su cui il numero di mosse cresce esponenzialmente con n (per l'intorno 2-opt del TSP, si veda Englert, Röglin, Vöcking, *Algorithmica* 68, 2014).
- In pratica il numero di mosse è piccolo, e il tempo di esecuzione è dominato dall'esplorazione degli intorni.

La ricerca locale intensifica

- La ricerca locale **intensifica**: scende finché non raggiunge il fondo della valle in cui è partita.
- Quale valle, e quindi quale ottimo locale, dipende
  - dalla soluzione di partenza,
  - dall'intorno,
  - dalla regola usata per scegliere la mossa.

Riepilogo

Lo schema:

- *Improve* restituisce un vicino migliorante o *NULL*; *LS* si sposta finché non riceve *NULL*
- Quattro ingredienti: soluzione di partenza, intorno, criterio di accettazione, criterio di arresto

Terminazione:

- *LS* si ferma in un ottimo locale, dopo al più $f(x^0) - z(I)$ mosse per f intera (minimizzazione)
- In pratica poche mosse: il tempo va nell'esplorazione degli intorni

Intorni per i quattro problemi

Introduzione: Intorni per i quattro problemi

- In questa sezione definiamo gli intorni usati nel capitolo: due per ogni problema, e 3-opt per il TSP
- Per ogni intorno fissiamo l'ordine delle mosse e ne contiamo gli elementi
- La dimensione di un intorno è il numero di vicini da valutare a ogni iterazione

Vicini ammissibili, in un ordine fissato

- Gli intorni sono definiti da piccole modifiche della rappresentazione delle soluzioni.
- All'intorno appartengono solo i vicini **ammissibili**: una modifica che produce un oggetto non ammissibile non è una mossa.
- Per ogni intorno fissiamo l'**ordine** in cui le mosse sono esaminate, che conta quando la ricerca accetta il primo vicino migliorante che trova.

KP: flip e swap

- L'intorno **flip**, definito nel capitolo «Algoritmi euristici: definizioni», cambia una componente di x, cioè aggiunge o toglie un oggetto; ha n elementi prima del test di ammissibilità. I flip sono esaminati per $j = 1, 2, \ldots, n$.
- L'intorno **swap** toglie un oggetto selezionato j e aggiunge un oggetto non selezionato k. Se S è l'insieme degli oggetti selezionati, ha $|S|\,(n - |S|) \leq n^2/4$ elementi; gli swap sono esaminati per $j \in S$ in ordine crescente e, per ogni j, per $k \notin S$ in ordine crescente.
- I due intorni sono complementari: un flip cambia il numero di oggetti selezionati, uno swap lo mantiene. L'intorno **flip+swap** è la loro unione, con i flip esaminati per primi.

BPP: move

- L'intorno **move** sposta un oggetto j dal suo contenitore $a = bin_j$ a un altro contenitore non vuoto b con $L_b + w_j \leq c$.
- Ha al più $n(nb-1)$ elementi; le mosse sono esaminate per $j = 1, 2, \ldots, n$ e, per ogni j, per i contenitori in ordine crescente di etichetta.
- Spostare un oggetto in un contenitore vuoto non è mai utile: non può diminuire il numero di contenitori.
- I contenitori mantengono le etichette della soluzione di partenza; un contenitore svuotato da una mossa semplicemente scompare.

BPP: swap

- L'intorno **swap** scambia due oggetti $j < k$ in contenitori diversi, purché entrambi i contenitori restino ammissibili:

$$L_{bin_j} - w_j + w_k \leq c \qquad \text{e} \qquad L_{bin_k} - w_k + w_j \leq c.$$

- Scambiare due oggetti di peso uguale non cambia nessun carico, e non si considera.

- Ha al più $\frac{n(n-1)}{2}$ elementi, esaminati per $j = 1, \ldots, n$ e $k = j + 1, \ldots, n$.

TSP: la mossa 2-opt

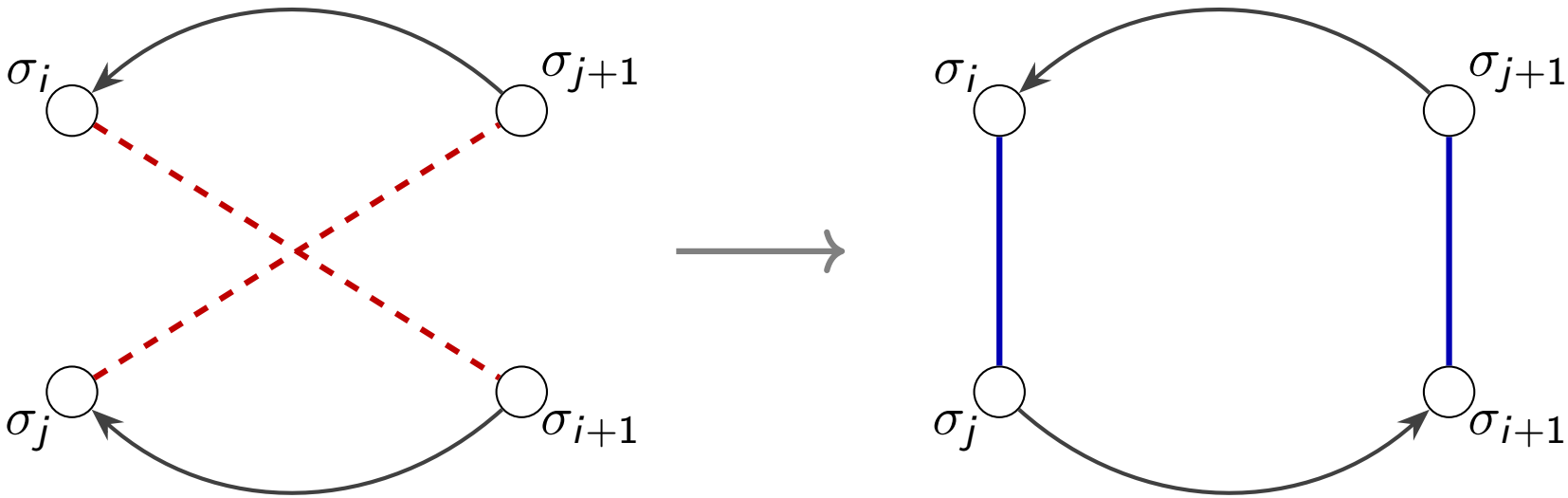
- Un giro è una permutazione $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ delle città, con $\sigma_{n+1} = \sigma_1$. Teniamo la città 1 in prima posizione, $\sigma_1 = 1$.
- La mossa **2-opt** con posizioni $i < j$ toglie i due lati $\{\sigma_i, \sigma_{i+1}\}$ e $\{\sigma_j, \sigma_{j+1}\}$, e richiude il giro nell'unico altro modo possibile: con i lati $\{\sigma_i, \sigma_j\}$ e $\{\sigma_{i+1}, \sigma_{j+1}\}$, percorrendo il cammino $\sigma_{i+1}, \ldots, \sigma_j$ nel verso opposto (Croes, *Operations Research* 6, 1958):

$$(\sigma_1, \ldots, \sigma_i, \underbrace{\sigma_{i+1}, \sigma_{i+2}, \ldots, \sigma_j}_{\text{cammino}}, \sigma_{j+1}, \ldots, \sigma_n)$$

$$\downarrow$$

$$(\sigma_1, \ldots, \sigma_i, \underbrace{\sigma_j, \ldots, \sigma_{i+2}, \sigma_{i+1}}_{\text{cammino invertito}}, \sigma_{j+1}, \ldots, \sigma_n).$$

TSP: la mossa 2-opt in una figura



prima: i due lati tratteggiati sono tolti

dopo: due lati nuovi, il cammino in basso è invertito

TSP: ordine e dimensione dell'intorno 2-opt

- Le mosse sono esaminate per $i = 1, \ldots, n-2$ e $j = i+2, \ldots, n$, escluso $i = 1$, $j = n$ (i due lati avrebbero in comune la città σ_1).

- L'intorno 2-opt ha

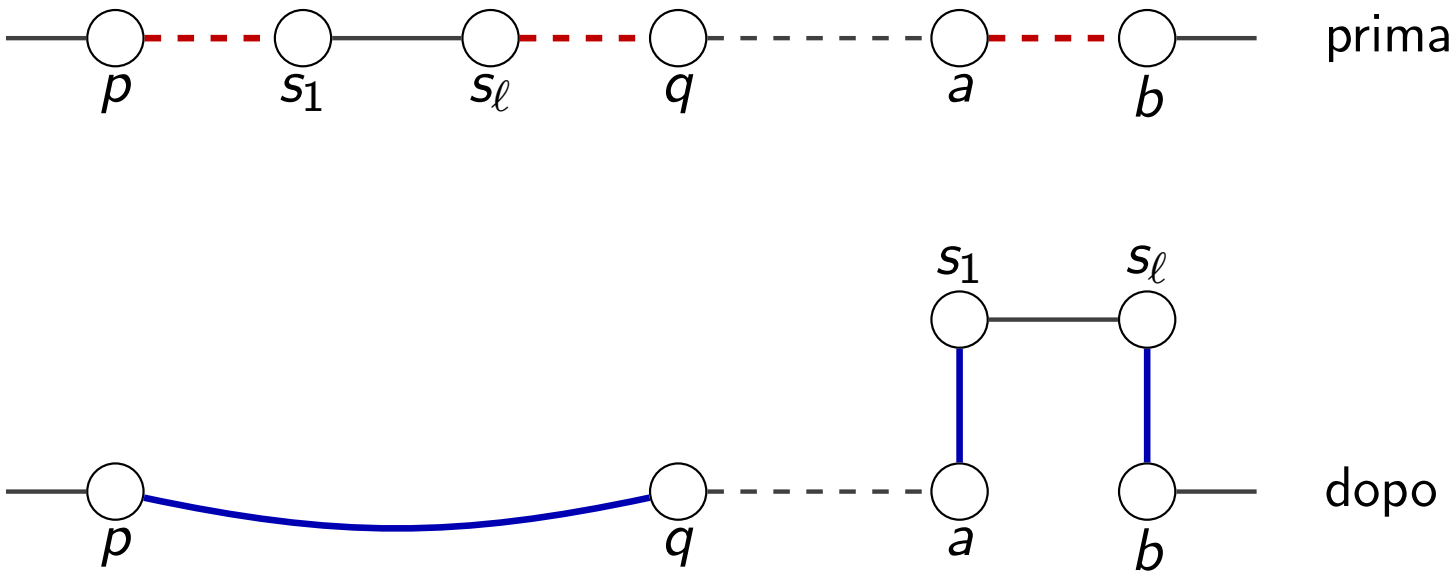
$$\frac{n(n-3)}{2}$$

elementi: 5 per l'istanza di riferimento.

TSP: la mossa Or-opt

- La mossa **Or-opt** (Or, tesi di dottorato, Northwestern University, 1976) prende un **segmento** di $\ell \in \{1, 2, 3\}$ città consecutive $\sigma_i, \ldots, \sigma_{i+\ell-1}$ $(2 \leq i,\ i + \ell - 1 \leq n)$.
- Lo inserisce, con lo stesso orientamento, fra due città consecutive σ_k e σ_{k+1} esterne al segmento $(k \notin \{i-1, \ldots, i+\ell-1\})$.
- Notazione:
  - $p = \sigma_{i-1}$ e $q = \sigma_{i+\ell}$ sono le città prima e dopo il segmento;
  - $s_1 = \sigma_i$ e $s_\ell = \sigma_{i+\ell-1}$ sono la sua prima e la sua ultima città;
  - $a = \sigma_k$ e $b = \sigma_{k+1}$.

TSP: la mossa Or-opt in una figura



- I lati rossi tratteggiati sono tolti, quelli blu sono aggiunti.

TSP: ordine e dimensione dell'intorno Or-opt

- Le mosse sono esaminate per $\ell = 1, 2, 3$, poi per $i = 2, \ldots, n - \ell + 1$, poi per $k = 1, \ldots, n$.
- Per ogni ℓ ci sono $n - \ell$ segmenti e $n - \ell - 1$ punti di inserimento, quindi l'intorno ha

$$\sum_{\ell=1}^{3} (n - \ell)(n - \ell - 1)$$

  elementi, cioè $O(n^2)$: 170 per $n = 10$.
- Alcune di queste mosse danno lo stesso giro (spostare una sola città in avanti di una posizione equivale a spostare il suo successore indietro di una posizione).

TSP: Or-opt e 3-opt

- Una mossa Or-opt toglie *tre* lati, $\{p, s_1\}$, $\{s_\ell, q\}$ e $\{a, b\}$.
- L'intorno **3-opt** (Lin, *Bell System Technical Journal* 44, 1965) contiene tutti i modi di togliere tre lati e ricollegare i tre cammini: ha $O(n^3)$ elementi.
- Or-opt è la parte di 3-opt che si può esplorare in tempo $O(n^2)$.

$P \,||\, C_{\max}$: move e swap

- L'intorno **move** sposta un lavoro j dalla sua macchina $a = mach_j$ a un'altra macchina $h \neq a$.
- Ogni mossa è ammissibile, e l'intorno ha $n\,(m-1)$ elementi, esaminati per $j = 1, \ldots, n$ e, per ogni j, per $h = 1, \ldots, m$.
- L'intorno **swap** scambia le macchine di due lavori $j < k$ su macchine diverse con $t_j \neq t_k$ (altrimenti nessun carico cambia).
- Ha al più $\frac{n(n-1)}{2}$ elementi, esaminati per $j = 1, \ldots, n$ e $k = j+1, \ldots, n$.
- Uno swap non cambia mai il numero di lavori su una macchina.

Gli intorni in sintesi (1/2)

problema	intorno	mossa	dimensione
KP	flip	aggiungere o togliere l'oggetto j	n
KP	swap	oggetto j fuori, oggetto k dentro	$\|S\|(n-\|S\|) \leq \frac{n^2}{4}$
BPP	move	oggetto j in un altro contenitore	$\leq n\,(nb-1)$
BPP	swap	scambiare gli oggetti j e k	$\leq \frac{n(n-1)}{2}$

Gli intorni in sintesi (2/2)

problema	intorno	mossa	dimensione
TSP	2-opt	togliere 2 lati, invertire un cammino	$\frac{n(n-3)}{2}$
TSP	Or-opt	spostare un segmento di 1, 2 o 3 città	$\sum_{\ell=1}^{3}(n-\ell)(n-\ell-1)$
TSP	3-opt	togliere 3 lati, ricollegare	$O(n^3)$
$P \mid\mid C_{\max}$	move	lavoro j su un'altra macchina	$n\,(m-1)$
$P \mid\mid C_{\max}$	swap	scambiare i lavori j e k	$\leq \frac{n(n-1)}{2}$

La dimensione di un intorno

- Le dimensioni sono quelle *prima* del test di ammissibilità (KP, BPP) e comprendono mosse che non cambiano l'obiettivo.
- La dimensione di un intorno è il numero di vicini da valutare a ogni iterazione, quindi determina il costo di un'iterazione.

Riepilogo

Gli intorni:

- KP: flip e swap; BPP e $P \,||\, C_{\max}$: move e swap; TSP: 2-opt, Or-opt e 3-opt
- Solo vicini ammissibili, esaminati in un ordine fissato

Le mosse del TSP:

- 2-opt toglie due lati e inverte un cammino: $\frac{n(n-3)}{2}$ mosse
- Or-opt sposta un segmento di 1, 2 o 3 città: $O(n^2)$ mosse, una parte di 3-opt

La dimensione:

- Il numero di vicini da valutare a ogni iterazione

First improvement e best improvement

Introduzione: First improvement e best improvement

- In questa sezione scriviamo le due regole che scelgono il vicino migliorante
- Confrontiamo il loro costo per iterazione e gli ottimi locali che raggiungono
- Eseguiamo a mano entrambe le regole sull'istanza di riferimento del KP

Due regole per *Improve*

- La funzione *Improve* di *LS* può fermarsi al *primo* vicino migliorante che trova, oppure esaminare tutto l'intorno e restituire il vicino *migliore*.

- Gli pseudocodici sono scritti per un problema di minimizzazione; per un problema di massimizzazione, "$<$" diventa "$>$".

First improvement

La regola *first improvement* esamina i vicini
della soluzione corrente in un ordine fissato
e restituisce il primo vicino
migliore della soluzione corrente.

Se nessun vicino è migliore, restituisce *NULL*.

Algorithm: First improvement

Input : una soluzione ammissibile x, un intorno N, una funzione obiettivo f (da minimizzare)

Output: il primo vicino $y \in N(x)$ con $f(y) < f(x)$, oppure *NULL*

```
1 function FirstImprovement(x, N, f)
2     for y ∈ N(x), nell'ordine dell'intorno do
3         if f(y) < f(x) then
4             return y          // vicino migliorante:
                                   accettarlo subito
5         end
6     end
7     return NULL               // x è un ottimo locale
8 end
```

Best improvement

La regola *best improvement* esamina tutti i vicini
della soluzione corrente e restituisce il migliore,
se è migliore della soluzione corrente;
i pareggi si risolvono a favore del vicino esaminato per primo.

Se nessun vicino è migliore, restituisce *NULL*.

Algorithm: Best improvement

Input : una soluzione ammissibile x, un intorno N, una funzione obiettivo f (da minimizzare)

Output : un vicino migliore $y \in N(x)$ se $f(y) < f(x)$, altrimenti *NULL*

```
1  function BestImprovement(x, N, f)
2      ȳ := NULL          // miglior vicino migliorante trovato
         finora
3      f̄ := f(x)                                  // il suo valore
4      for y ∈ N(x), nell'ordine dell'intorno do
5          if f(y) < f̄ then
6              ȳ ← y      // migliore di tutti i vicini visti
                 finora
7              f̄ ← f(y)
8          end
9      end
10     return ȳ                  // NULL se x è un ottimo locale
11 end
```

Confronto (1/2)

- In un ottimo locale entrambe le regole esaminano tutto l'intorno: l'ultima iterazione costa lo stesso.
- Prima di raggiungerlo, **first improvement** esegue iterazioni **meno costose**, perché si ferma al primo vicino migliorante, ma ogni mossa di solito migliora meno, quindi tende a eseguire **più** mosse.
- **Best improvement** esegue la discesa più ripida: ogni iterazione costa un'esplorazione completa dell'intorno, e ogni mossa migliora il più possibile.

Confronto (2/2)

- Le due regole seguono percorsi diversi, quindi possono fermarsi in **ottimi locali diversi**.
- Nessuna delle due regole è migliore dell'altra su tutte le istanze: la mossa più ripida ora può portare in una valle peggiore dopo.
- Con first improvement il risultato dipende dall'ordine in cui sono esaminati i vicini; rendere casuale quell'ordine è un modo semplice per ottenere ottimi locali diversi dalla stessa soluzione di partenza.

Esempio: first e best improvement sul KP (1/6)

- Partiamo dalla soluzione del greedy per peso GR_w sull'istanza di riferimento del KP ($c = 9$), cioè

$$x = (0, 0, 1, 1),$$

  di peso 6 e profitto 10.

- Usiamo l'intorno flip+swap.

- I vicini sono esaminati nell'ordine fissato nella sezione precedente: prima i flip, poi gli swap.

Esempio: first e best improvement sul KP (2/6)

I vicini di $x = (0, 0, 1, 1)$:

mossa	vicino	peso	profitto	
flip 1	$(1, 0, 1, 1)$	11	—	non ammissibile
flip 2	$(0, 1, 1, 1)$	10	—	non ammissibile
flip 3	$(0, 0, 0, 1)$	3	4	
flip 4	$(0, 0, 1, 0)$	3	6	
swap 3 fuori, 1 dentro	$(1, 0, 0, 1)$	8	14	$\leftarrow$ first improvement
swap 3 fuori, 2 dentro	$(0, 1, 0, 1)$	7	11	
swap 4 fuori, 1 dentro	$(1, 0, 1, 0)$	8	16	$\leftarrow$ best improvement
swap 4 fuori, 2 dentro	$(0, 1, 1, 0)$	7	13	

Esempio: first e best improvement sul KP (3/6)

First improvement accetta $(1, 0, 0, 1)$, di profitto 14, dopo aver esaminato 3 vicini ammissibili. Da $(1, 0, 0, 1)$ (peso 8) i vicini sono esaminati di nuovo dall'inizio:

mossa	vicino	peso	profitto	
flip 1	$(0, 0, 0, 1)$	3	4	
flip 2	$(1, 1, 0, 1)$	12	—	non ammissibile
flip 3	$(1, 0, 1, 1)$	11	—	non ammissibile
flip 4	$(1, 0, 0, 0)$	5	10	
swap 1 fuori, 2 dentro	$(0, 1, 0, 1)$	7	11	
swap 1 fuori, 3 dentro	$(0, 0, 1, 1)$	6	10	
swap 4 fuori, 2 dentro	$(1, 1, 0, 0)$	9	17	$\leftarrow$ first improvement

Esempio: first e best improvement sul KP (4/6)

- $(1, 1, 0, 0)$, di profitto 17, è accettata dopo 5 vicini ammissibili.
- È la soluzione ottima, quindi nessun vicino è migliore: i 6 vicini ammissibili sono esaminati e la ricerca si ferma.
- In totale, **first improvement** esegue

  2 mosse, $\quad 3 + 5 + 6 = 14$ vicini ammissibili esaminati.

Esempio: first e best improvement sul KP (5/6)

- **Best improvement** esamina tutti i 6 vicini ammissibili di $(0, 0, 1, 1)$ e si sposta su $(1, 0, 1, 0)$, di profitto 16.
- È la soluzione di *GR*, il cui intorno è nel capitolo «Algoritmi euristici: definizioni»: il vicino migliore è $(1, 1, 0, 0)$, di profitto 17, raggiunto con lo swap "3 fuori, 2 dentro".
- La ricerca si ferma lì dopo aver esaminato i suoi 6 vicini ammissibili. In totale:

  2 mosse, $\quad 6 + 6 + 6 = 18$ vicini ammissibili esaminati.

Esempio: first e best improvement sul KP (6/6)

- Le due regole raggiungono l'ottimo lungo percorsi diversi:

$$\text{first improvement:} \quad 10 \to 14 \to 17,$$
$$\text{best improvement:} \quad 10 \to 16 \to 17,$$

  e first improvement esamina meno vicini.

- Il solo intorno flip non si sarebbe mosso affatto: da $(0, 0, 1, 1)$ ogni flip ammissibile toglie un oggetto, quindi $(0, 0, 1, 1)$, di profitto 10, è un ottimo locale per flip.

Riepilogo

First improvement:

- Restituisce il primo vicino migliorante: iterazioni meno costose, di solito più mosse
- Il risultato dipende dall'ordine dei vicini

Best improvement:

- Esamina tutto l'intorno e restituisce il vicino migliore: la discesa più ripida
- I pareggi vanno al vicino esaminato per primo

Sul KP:

- Entrambe le regole raggiungono 17 da 10, lungo percorsi diversi, con 14 e 18 vicini esaminati

Valutare una mossa in modo incrementale

Introduzione: Valutare una mossa in modo incrementale

- La maggior parte del tempo di esecuzione di una ricerca locale si spende nella valutazione dei vicini
- In questa sezione calcoliamo la variazione dell'obiettivo dalle poche componenti che una mossa cambia
- Per la mossa 2-opt la variazione si calcola in tempo $O(1)$ invece che $O(n)$

Da zero o in modo incrementale

- Ogni iterazione di *LS* valuta molti vicini, e la maggior parte del tempo di esecuzione si spende lì.
- Calcolare $f(y)$ **da zero** costa $O(n)$ per ciascuno dei quattro problemi: il profitto di una soluzione del KP, i carichi dei contenitori o delle macchine, la lunghezza di un giro sono somme di n termini.
- Una mossa cambia solo poche componenti della soluzione.
- La **valutazione incrementale** calcola la **variazione** $\Delta = f(y) - f(x)$ dalle componenti che cambiano, conservando alcune informazioni sulla soluzione corrente (il suo profitto e il suo peso, i suoi carichi), senza ricalcolare $f(y)$.

Proposizione 2

Sia σ' il giro ottenuto da σ con la mossa 2-opt di posizioni $i < j$, e siano $u_1 = \sigma_i$, $v_1 = \sigma_{i+1}$, $u_2 = \sigma_j$, $v_2 = \sigma_{j+1}$. Allora $L(\sigma') = L(\sigma) + \Delta$, con

$$\Delta \;=\; d_{u_1 u_2} + d_{v_1 v_2} - d_{u_1 v_1} - d_{u_2 v_2}.$$

Dimostrazione

- I giri σ e σ' hanno gli stessi lati, tranne che
  - σ contiene $\{u_1, v_1\}$ e $\{u_2, v_2\}$,
  - σ' contiene $\{u_1, u_2\}$ e $\{v_1, v_2\}$.

- Il cammino invertito $v_1, \ldots, u_2$ usa gli stessi lati nel verso opposto, e la sua lunghezza non cambia perché le distanze sono simmetriche.

□

La variazione di lunghezza di una mossa 2-opt

La variazione di lunghezza di una mossa 2-opt si calcola in tempo $O(1)$:

$$\Delta = \underbrace{d_{u_1 u_2} + d_{v_1 v_2}}_{\text{lati aggiunti}} - \underbrace{d_{u_1 v_1} + d_{u_2 v_2}}_{\text{lati tolti}}$$

e la mossa è migliorante se e solo se $\Delta < 0$.

Esempio

- Sul giro $(1, 3, 2, 4, 5)$ dell'istanza di riferimento, di lunghezza 25, la mossa con posizioni $i = 2$, $j = 4$ toglie $\{3, 2\}$ e $\{4, 5\}$ e aggiunge $\{3, 4\}$ e $\{2, 5\}$:

$$\Delta = d_{34} + d_{25} - d_{32} - d_{45} = 4 + 4 - 4 - 7 = -3.$$

- Il nuovo giro $(1, 3, 4, 2, 5)$ ha lunghezza $25 - 3 = 22$, come conferma il calcolo diretto:

$$2 + 4 + 3 + 4 + 9 = 22.$$

Perché conta

- L'intorno 2-opt ha $\frac{n(n-3)}{2}$ mosse.
- Con la valutazione incrementale un'iterazione di best improvement costa $O(n^2)$; ricalcolare ogni giro da zero costerebbe $O(n^3)$.
- Per $n = 1000$ città, un'iterazione valuta 498 500 mosse: circa $5 \cdot 10^5$ operazioni invece di circa $5 \cdot 10^8$.
- L'inversione del cammino, che costa $O(n)$, si esegue una sola volta per iterazione, per la mossa accettata.

Gli altri intorni

La stessa idea si applica agli altri intorni:

mossa	variazione della soluzione	tempo
KP, flip di j	profitto $\pm p_j$, peso $\pm w_j$	$O(1)$
KP, swap j fuori, k dentro	profitto $+p_k - p_j$, peso $+w_k - w_j$	$O(1)$
BPP, move di j da a a b	$L_a - w_j$, $L_b + w_j$; il contenitore a si svuota sse $L_a = w_j$	$O(1)$
TSP, 2-opt	$d_{u_1 u_2} + d_{v_1 v_2} - d_{u_1 v_1} - d_{u_2 v_2}$	$O(1)$
TSP, Or-opt	$d_{pq} + d_{as_1} + d_{s_\ell b} - d_{ps_1} - d_{s_\ell q} - d_{ab}$	$O(1)$
$P \mid\mid C_{\max}$, move di j da a a h	$L_a - t_j$, $L_h + t_j$; $C_{\max} = \max\{L_a - t_j, L_h + t_j, \max_{q \neq a,h} L_q\}$	$O(m)$

Il costo di un'iterazione

- La formula di Or-opt si ottiene come per 2-opt: la mossa toglie i lati $\{p, s_1\}$, $\{s_\ell, q\}$, $\{a, b\}$ e aggiunge $\{p, q\}$, $\{a, s_1\}$, $\{s_\ell, b\}$, e il segmento mantiene il suo orientamento.
- Il costo di un'iterazione di *LS* è la dimensione dell'intorno per il costo della valutazione di una mossa (più il costo di applicare la mossa accettata, che si paga una volta per iterazione):

problema	intorno	valutazione di una mossa	costo di un'iterazione
KP	flip	$O(1)$	$O(n)$
KP	swap	$O(1)$	$O(n^2)$
BPP	move	$O(1)$	$O(n \cdot nb)$
BPP	swap	$O(1)$	$O(n^2)$
TSP	2-opt, Or-opt	$O(1)$	$O(n^2)$
TSP	3-opt	$O(1)$	$O(n^3)$
$P \mid\mid C_{\max}$	move	$O(m)$	$O(nm^2)$
$P \mid\mid C_{\max}$	swap	$O(m)$	$O(n^2 m)$

Quali mosse, e quanto velocemente?

- La valutazione incrementale non cambia gli ottimi locali, né il percorso seguito dalla ricerca: cambia solo il tempo necessario a seguirlo.
- Per questo il progetto di una ricerca locale parte sempre da due domande:

quali mosse? e *quanto velocemente si può valutare una mossa?*

Riepilogo

Valutazione incrementale:

- Calcolare $\Delta = f(y) - f(x)$ dalle componenti che cambiano, non $f(y)$ da zero
- 2-opt: $\Delta = d_{u_1 u_2} + d_{v_1 v_2} - d_{u_1 v_1} - d_{u_2 v_2}$, in $O(1)$; migliorante sse $\Delta < 0$

Costo di un'iterazione:

- Dimensione dell'intorno per il costo della valutazione di una mossa
- Stessi ottimi locali e stesso percorso, in meno tempo

TSP: 2-opt e Or-opt

Introduzione: TSP: 2-opt e Or-opt

- In questa sezione scriviamo le ricerche locali 2-opt e Or-opt per il TSP, con la valutazione incrementale
- Le eseguiamo sull'istanza di riferimento e su un'istanza media con 10 città
- Vediamo che le due regole, e i due intorni, possono fermarsi in ottimi locali diversi

La ricerca locale 2-opt

- La ricerca locale *2-opt* (indicata con $2OPT$) è una ricerca locale per il TSP con l'intorno 2-opt.
- A ogni iterazione valuta le mosse 2-opt del giro corrente con la formula incrementale.
- Applica una mossa migliorante scelta con la regola first improvement o best improvement, e si ferma quando nessuna mossa accorcia il giro.

Algorithm: Passo 2-opt (1/3)

Input : n città, distanze $d[1..n][1..n]$, giro $\sigma[1..n]$ con $\sigma[1] = 1$, regola $\in \{\textit{first}, \textit{best}\}$

Output : *true* se a σ è stata applicata una mossa 2-opt migliorante, *false* altrimenti

```
1 function TwoOptStep(n, d, σ, rule)
2 |   Δ̄ := 0          // miglior variazione di lunghezza
  |    trovata finora
3 |   (ī, j̄) := (0, 0)      // miglior mossa trovata finora
4 |   for i := 1 to n − 2 do
5 |   |   for j := i + 2 to n do
6 |   |   |   if i = 1 and j = n then continue// i due lati
  |   |   |    sono adiacenti
```

Algorithm: Passo 2-opt (2/3)

function *TwoOptStep*$(n, d, \sigma, rule)$ (continua)

```
    for i := 1 to n − 2 (continua) do
        for j := i + 2 to n (continua) do
7           u1 := σ[i]; v1 := σ[i + 1]  // primo lato tolto
8           u2 := σ[j]; v2 := σ[j mod n + 1]      // secondo
             lato tolto
9           Δ := d[u1][u2] + d[v1][v2] − d[u1][v1] − d[u2][v2]
10          if Δ < Δ̄ then
11              Δ̄ ← Δ; (ī, j̄) ← (i, j)                  // mossa
                 migliorante
12              if rule = first then
13                  uscire da entrambi i cicli
14              end
15          end
16      end
17  end
```

Algorithm: Passo 2-opt (3/3)

function *TwoOptStep*$(n, d, \sigma,$ *rule*$)$ (continua)

```
18    if ī = 0 then return false                    // nessuna mossa migliorante: ottimo locale
19    invertire il cammino σ[ī + 1], ..., σ[j̄]      // applicare la mossa
20    return true
21 end
```

Come funziona 2*OPT*

- Nell'implementazione, *Improve* e lo spostamento sul vicino migliorante sono riuniti in un'unica funzione *Step*:
  - esplora l'intorno con la valutazione incrementale;
  - se trova una mossa migliorante, la applica alla soluzione corrente e restituisce *true*;
  - altrimenti restituisce *false*.
- *LS* chiama *Step* finché non restituisce *false*.
- 2*OPT* è *LocalSearch* con *TwoOptStep*; non cambia mai $\sigma[1] = 1$, perché $i \geq 1$ e il cammino invertito parte dalla posizione $i + 1 \geq 2$.
- Un'iterazione costa tempo $O(n^2)$; l'inversione del cammino costa $O(n)$.

2-opt scioglie gli incroci

- Quando le distanze sono euclidee, un ottimo locale 2-opt non ha lati che si incrociano.
- Se i lati $\{u_1, v_1\}$ e $\{u_2, v_2\}$ si incrociano in un punto X, la disuguaglianza triangolare dà

$$d_{u_1 u_2} \leq d_{u_1 X} + d_{X u_2} \qquad \text{e} \qquad d_{v_1 v_2} \leq d_{v_1 X} + d_{X v_2},$$

  quindi $\Delta \leq 0$, con $\Delta < 0$ a meno che le quattro città non siano allineate.
- La mossa 2-opt "scioglie" l'incrocio dei due lati.

Esempio: $2OPT$ sull'istanza di riferimento del TSP (1/6)

Partiamo dal giro di NN dalla città 1, $\sigma = (1, 3, 2, 4, 5)$, di lunghezza 25. Le sue 5 mosse sono

(i, j)	lati tolti	lati aggiunti	Δ
$(1, 3)$	$\{1, 3\}, \{2, 4\}$	$\{1, 2\}, \{3, 4\}$	$5 + 4 - 2 - 3 = +4$
$(1, 4)$	$\{1, 3\}, \{4, 5\}$	$\{1, 4\}, \{3, 5\}$	$2 + 7 - 2 - 7 = 0$
$(2, 4)$	$\{3, 2\}, \{4, 5\}$	$\{3, 4\}, \{2, 5\}$	$4 + 4 - 4 - 7 = -3$
$(2, 5)$	$\{3, 2\}, \{5, 1\}$	$\{3, 5\}, \{2, 1\}$	$7 + 5 - 4 - 9 = -1$
$(3, 5)$	$\{2, 4\}, \{5, 1\}$	$\{2, 5\}, \{4, 1\}$	$4 + 2 - 3 - 9 = -6$

Esempio: $2OPT$ sull'istanza di riferimento del TSP (2/6)

I nuovi giri dati dalle 5 mosse di $\sigma = (1, 3, 2, 4, 5)$:

(i, j)	Δ	nuovo giro, lunghezza
$(1, 3)$	$+4$	$(1, 2, 3, 4, 5)$, 29
$(1, 4)$	0	$(1, 4, 2, 3, 5)$, 25
$(2, 4)$	-3	$(1, 3, 4, 2, 5)$, 22
$(2, 5)$	-1	$(1, 3, 5, 4, 2)$, 24
$(3, 5)$	-6	$(1, 3, 2, 5, 4)$, 19

Esempio: 2*OPT* sull'istanza di riferimento del TSP (3/6)

First improvement:

- Accetta la mossa $(2, 4)$, la prima con $\Delta < 0$, dopo 3 valutazioni: $\sigma = (1, 3, 4, 2, 5)$, di lunghezza 22.
- Le sue mosse hanno $\Delta = +1, +6, +3, -4, 0$; la mossa $(2, 5)$, che toglie $\{3, 4\}$ e $\{5, 1\}$ e aggiunge $\{3, 5\}$ e $\{4, 1\}$, è accettata dopo 4 valutazioni: $\sigma = (1, 3, 5, 2, 4)$, di lunghezza 18.
- Tutte le sue 5 mosse hanno $\Delta > 0$ $(+7, +4, +1, +4, +6)$: la ricerca si ferma dopo $3 + 4 + 5 = 12$ valutazioni.

Esempio: 2*OPT* sull'istanza di riferimento del TSP (4/6)

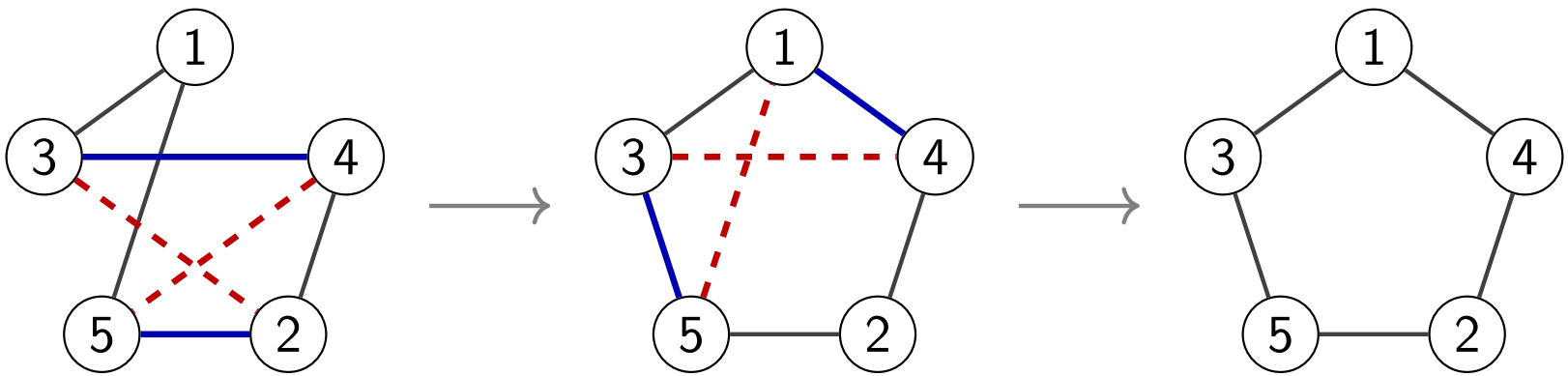
Best improvement:

- Accetta la mossa $(3, 5)$, con $\Delta = -6$: $\sigma = (1, 3, 2, 5, 4)$, di lunghezza 19.
- Le sue mosse hanno $\Delta = +6, +4, -1, +3, +6$, e la mossa $(2, 4)$, che toglie $\{3, 2\}$ e $\{5, 4\}$ e aggiunge $\{3, 5\}$ e $\{2, 4\}$, dà $\sigma = (1, 3, 5, 2, 4)$, di lunghezza 18.
- La ricerca si ferma dopo $5 + 5 + 5 = 15$ valutazioni.

Entrambe le regole raggiungono il giro ottimo, con 2 mosse ciascuna. Le prossime due slide mostrano i due percorsi: le città sono disposte nell'ordine del giro ottimo, quindi il giro ottimo è il bordo del pentagono (l'istanza non è euclidea: il disegno è solo uno schema). I lati rossi tratteggiati sono tolti, quelli blu sono aggiunti.

Esempio: $2OPT$ sull'istanza di riferimento del TSP (5/6)

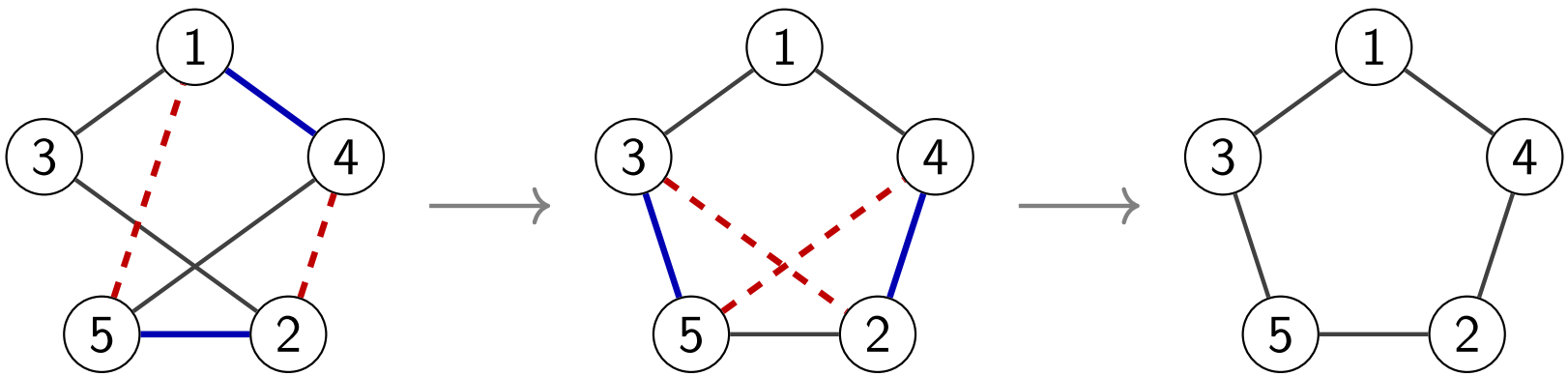
first improvement



$(1, 3, 2, 4, 5)$, 25 $(1, 3, 4, 2, 5)$, 22 $(1, 3, 5, 2, 4)$, 18

Esempio: $2OPT$ sull'istanza di riferimento del TSP (6/6)

best improvement



$(1, 3, 2, 4, 5)$, 25 $(1, 3, 2, 5, 4)$, 19 $(1, 3, 5, 2, 4)$, 18

La ricerca locale Or-opt

- La ricerca locale *Or-opt* (indicata con *OROPT*) è una ricerca locale per il TSP con l'intorno Or-opt.
- A ogni iterazione valuta, con la formula incrementale, l'inserimento di ogni segmento di 1, 2 o 3 città consecutive fra altre due città consecutive.
- Applica una mossa migliorante scelta con la regola first improvement o best improvement, e si ferma quando nessuna mossa accorcia il giro.

Algorithm: Passo Or-opt (1/3)

Input : n città, distanze $d[1..n][1..n]$, giro $\sigma[1..n]$ con $\sigma[1] = 1$, regola $\in \{first, best\}$

Output : *true* se a σ è stata applicata una mossa Or-opt migliorante, *false* altrimenti

```
1  function OrOptStep(n, d, σ, rule)
2      Δ̄ := 0; (ℓ̄, ī, k̄) := (0, 0, 0)        // miglior mossa trovata finora
3      for ℓ := 1 to 3 do
4          for i := 2 to n − ℓ + 1 do
5              p := σ[i − 1]; s_1 := σ[i]     // predecessore, prima città del segmento
6              s_ℓ := σ[i + ℓ − 1]; q := σ[(i + ℓ − 1) mod n + 1]
                // ultima città, successore
7              for k := 1 to n do
8                  if i − 1 ≤ k ≤ i + ℓ − 1 then continue // stesso posto, o dentro il segmento
```

Algorithm: Passo Or-opt (2/3)

function *OrOptStep*$(n, d, \sigma, rule)$ (continua)
    for $\ell := 1$ **to** 3 (continua) **do**
        for $i := 2$ **to** $n - \ell + 1$ (continua) **do**
            for $k := 1$ **to** n (continua) **do**
9                $a := \sigma[k]$; $b := \sigma[k \bmod n + 1]$ `// lato in cui va il segmento`
10                $\Delta := d[p][q] + d[a][s_1] + d[s_\ell][b] - d[p][s_1] - d[s_\ell][q] - d[a][b]$
11   **if** $\Delta < \bar{\Delta}$ **then**
12                    $\bar{\Delta} \leftarrow \Delta$; $(\bar{\ell}, \bar{\imath}, \bar{k}) \leftarrow (\ell, i, k)$ `// mossa migliorante`
13   **if** *rule* = *first* **then**
14   **uscire** dai tre cicli
15   **end**
16   **end**
17   **end**

Algorithm: Passo Or-opt (3/3)

function *OrOptStep*$(n, d, \sigma, rule)$ (continua)
    for $\ell := 1$ **to** 3 (continua) **do**
        for $i := 2$ **to** $n - \ell + 1$ (continua) **do**
18   **end**
19 **end**
20 **if** $\bar{\ell} = 0$ **then return** *false* `// nessuna mossa migliorante: ottimo locale`
21 spostare $\sigma[\bar{\imath}], \ldots, \sigma[\bar{\imath} + \bar{\ell} - 1]$ fra $\sigma[\bar{k}]$ e $\sigma[\bar{k} \bmod n + 1]$ `// applicare`
22 **return** *true*
23 **end**

Come funziona *OROPT*

- Or-opt sposta un breve tratto del giro in un posto migliore, senza invertire nulla.
- È l'intorno naturale quando alcune città sono state "lasciate indietro" dalla costruzione.
- Un'iterazione costa tempo $O(n^2)$.
- Un ottimo locale 2-opt non è necessariamente un ottimo locale Or-opt, e viceversa: i due intorni contengono mosse diverse.

First e best improvement su un'istanza media

- Con 5 città l'intorno 2-opt è piccolo ed entrambe le regole raggiungono l'ottimo.
- Per vedere la differenza fra le due regole, e il ruolo di Or-opt, usiamo un'istanza media con 10 città.
- Questa istanza non è metrica:

$$d_{68} = 8 > d_{67} + d_{78} = 4 + 3,$$
$$d_{6,10} = 11 > d_{67} + d_{7,10} = 4 + 6.$$

Esempio: un'istanza del TSP con 10 città (1/7)

Le città hanno coordinate intere, e d_{ij} è la distanza euclidea arrotondata all'intero più vicino:

città	1	2	3	4	5
coordinate	$(1, 1)$	$(3, 8)$	$(6, 10)$	$(10, 9)$	$(13, 6)$

città	6	7	8	9	10
coordinate	$(12, 1)$	$(8, 3)$	$(5, 4)$	$(8, 7)$	$(2, 5)$

Esempio: un'istanza del TSP con 10 città (2/7)

Le distanze:

d_{ij}	1	2	3	4	5	6	7	8	9	10
1	—	7	10	12	13	11	7	5	9	4
2	7	—	4	7	10	11	7	4	5	3
3	10	4	—	4	8	11	7	6	4	6
4	12	7	4	—	4	8	6	7	3	9
5	13	10	8	4	—	5	6	8	5	11
6	11	11	11	8	5	—	4	8	7	11
7	7	7	7	6	6	4	—	3	4	6
8	5	4	6	7	8	8	3	—	4	3
9	9	5	4	3	5	7	4	4	—	6
10	4	3	6	9	11	11	6	3	6	—

Esempio: un'istanza del TSP con 10 città (3/7)

- Fra i suoi 181 440 giri distinti, l'unico ottimo è

$$(1, 10, 2, 3, 9, 4, 5, 6, 7, 8), \qquad z(I) = 39.$$

- Il giro di *NN* dalla città 1 è

$$(1, 10, 2, 3, 4, 9, 7, 8, 5, 6), \qquad \text{di lunghezza } 49.$$

- L'intorno 2-opt ha $\frac{10 \cdot 7}{2} = 35$ mosse.

Esempio: un'istanza del TSP con 10 città (4/7)

First improvement esegue 4 mosse:

lati tolti	nuovo giro	Δ	lunghezza	valutazioni
$\{3,4\},\{8,5\}$	$(1,10,2,3,8,7,9,4,5,6)$	-2	47	23
$\{3,8\},\{6,1\}$	$(1,10,2,3,6,5,4,9,7,8)$	-1	46	25
$\{3,6\},\{4,9\}$	$(1,10,2,3,4,5,6,9,7,8)$	-3	43	22
$\{6,9\},\{7,8\}$	$(1,10,2,3,4,5,6,7,9,8)$	-2	41	33

e si ferma nell'ottimo locale $(1,10,2,3,4,5,6,7,9,8)$, di lunghezza 41, dopo 138 valutazioni in totale (le ultime 35 per certificare l'ottimo locale).

Esempio: un'istanza del TSP con 10 città (5/7)

Best improvement esegue 3 mosse:

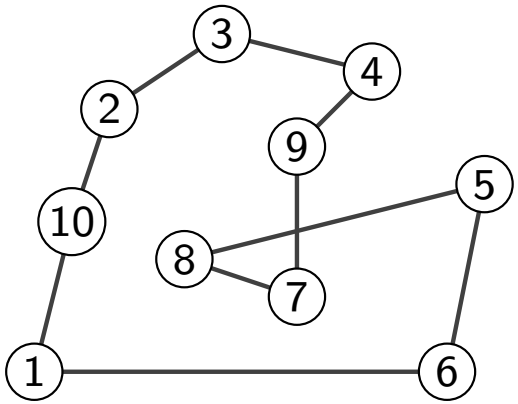
lati tolti	nuovo giro	Δ	lunghezza
$\{7,8\},\{6,1\}$	$(1,10,2,3,4,9,7,6,5,8)$	-5	44
$\{9,7\},\{5,8\}$	$(1,10,2,3,4,9,5,6,7,8)$	-4	40
$\{3,4\},\{9,5\}$	$(1,10,2,3,9,4,5,6,7,8)$	-1	39

e si ferma nel giro ottimo dopo $4 \cdot 35 = 140$ valutazioni.

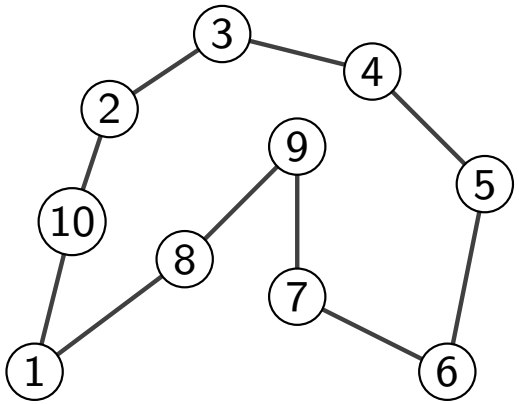
Esempio: un'istanza del TSP con 10 città (6/7)

- Dallo stesso giro di partenza, le due regole si fermano in **ottimi locali diversi**, dopo quasi lo stesso numero di valutazioni.
- Fra tutti i 181 440 giri, solo 3 sono ottimi locali 2-opt, di lunghezza 39, 41 e 43.

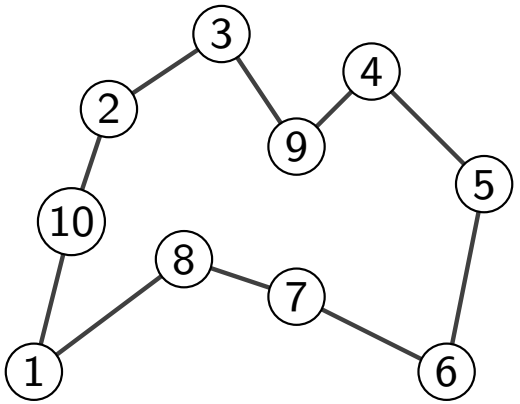
Esempio: un'istanza del TSP con 10 città (7/7)



NN: 49



first improvement: 41



best improvement: 39

- Il giro di partenza di *NN* e i due ottimi locali di 2-opt, con first e best improvement.

Esempio: *OROPT* da un ottimo locale 2-opt (1/2)

- Sull'istanza con 10 città, first improvement si è fermato nell'ottimo locale 2-opt

$$\sigma = (1, 10, 2, 3, 4, 5, 6, 7, 9, 8), \qquad \text{di lunghezza } 41.$$

- La mossa Or-opt che prende il segmento (9), con $\ell = 1$, fra $p = 7$ e $q = 8$, e lo inserisce fra $a = 3$ e $b = 4$ ha

$$\Delta = d_{78} + d_{39} + d_{94} - d_{79} - d_{98} - d_{34} = 3 + 4 + 3 - 4 - 4 - 4 = -2.$$

Esempio: *OROPT* da un ottimo locale 2-opt (2/2)

- È la prima mossa migliorante trovata, dopo 60 valutazioni, e anche la migliore delle 170 mosse.
- Il nuovo giro $(1, 10, 2, 3, 9, 4, 5, 6, 7, 8)$, di lunghezza 39, è ottimo.
- Fra tutti i 181 440 giri, solo 3 sono ottimi locali Or-opt, di lunghezza 39, 43 e 44, e solo il giro ottimo è un ottimo locale sia per 2-opt sia per Or-opt.

3-opt e Lin–Kernighan (1/2)

- L'intorno **3-opt** toglie tre lati e ricollega i tre cammini in tutti i modi possibili (Lin, 1965).
- I suoi ottimi locali sono migliori di quelli di 2-opt, ma un'iterazione costa tempo $O(n^3)$, e in pratica l'intorno si esplora solo in parte.

3-opt e Lin–Kernighan (2/2)

- L'euristica di **Lin–Kernighan** (Lin, Kernighan, *Operations Research* 21, 1973) costruisce un numero *variabile* k di scambi di lati, uno alla volta, scegliendo ogni scambio in modo greedy e fermandosi quando il guadagno totale non può più diventare positivo.
- Esplora parte dell'intorno k-opt per ogni k, in un tempo vicino a quello di 2-opt, e le sue implementazioni sono fra le migliori euristiche per il TSP.
- La sequenza 2-opt, 3-opt, k-opt mostra il compromesso del capitolo «Algoritmi euristici: definizioni»: un intorno più grande ha ottimi locali migliori, e costa di più da esplorare.

Riepilogo (1/2)

2-opt:

- *TwoOptStep* valuta le $\frac{n(n-3)}{2}$ mosse in $O(1)$ ciascuna e inverte un cammino
- Nel caso euclideo scioglie gli incroci
- Sull'istanza di riferimento entrambe le regole vanno da 25 all'ottimo 18 in 2 mosse

Or-opt:

- *OrOptStep* sposta un segmento di 1, 2 o 3 città, senza invertirlo
- I suoi ottimi locali sono diversi da quelli di 2-opt

Riepilogo (2/2)

L'istanza con 10 **città**:

- Da *NN* (49): first improvement si ferma a 41, best improvement all'ottimo 39
- Or-opt migliora l'ottimo locale 2-opt 41 fino all'ottimo 39
- Solo il giro ottimo è un ottimo locale per entrambi gli intorni

Intorni più grandi:

- 3-opt e Lin–Kernighan: ottimi locali migliori, esplorazione più costosa

Riepilogo della prima parte (1/2)

- Una ricerca locale parte da una soluzione ammissibile e si sposta su un vicino migliore finché ne esiste uno: si ferma in un ottimo locale rispetto al suo intorno.
- Ogni problema ha i suoi intorni: flip e swap per il KP, move e swap per il BPP e $P \,||\, C_{\max}$, 2-opt e Or-opt per il TSP.
- First improvement accetta il primo vicino migliorante, best improvement il migliore: possono fermarsi in ottimi locali diversi.

Riepilogo della prima parte (2/2)

- La valutazione incrementale calcola la variazione Δ di una mossa dalle componenti che cambiano: $O(1)$ per una mossa 2-opt.
- Sul TSP, 2-opt e Or-opt raggiungono l'ottimo dell'istanza di riferimento; sull'istanza con 10 città la regola e l'intorno decidono quale ottimo locale si raggiunge.
- Nella seconda parte: $P \,||\, C_{\max}$, plateau e obiettivi ausiliari, penalità per il KP, variable neighbourhood descent e multi-start.