Metaeuristiche (1/2)

Indice

- Metaeuristiche
- I problemi e le ipotesi
  - Gli intorni
  - Numeri casuali
  - Le istanze
- Come uscire da un ottimo locale
- Simulated annealing
  - Lo schema di raffreddamento e i parametri
  - Esempi
  - Soluzioni inammissibili: simulated annealing con penalità
  - Un esperimento sull'istanza media del TSP
  - Proprietà
- Tabu search
  - Esempio
  - Tabu search per il TSP, e l'effetto della durata tabu
- Iterated local search
  - Esempio

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- eseguire a mano simulated annealing, tabu search, iterated local search e variable neighbourhood search, con numeri casuali assegnati;
- eseguire GRASP, large neighbourhood search e un algoritmo genetico o memetico su istanze piccole;
- spiegare come ciascun metodo esce da un ottimo locale, e quali parametri lo regolano;
- confrontare i metodi sulle stesse istanze, con più semi;
- dire che cosa le metaeuristiche non danno: un bound duale.

Prerequisiti

- Da questo modulo, il capitolo «Ricerca locale»: intorni, first e best improvement, variable neighbourhood descent e multi-start.
- Da questo modulo, i capitoli «Euristiche costruttive» (costruzione randomizzata) e «Algoritmi euristici: definizioni» (intensificazione e diversificazione, valutazione sperimentale).

Questa prima parte presenta il quadro del capitolo e le prime tre metaeuristiche:
simulated annealing, tabu search e iterated local search.

Metaeuristiche

Introduzione: Metaeuristiche

- Una ricerca locale si ferma nel primo ottimo locale che incontra, che può essere lontano dall'ottimo
- In questa sezione definiamo che cos'è una **metaeuristica**: una strategia che guida la ricerca locale oltre quel primo ottimo locale
- Tutti i metodi del capitolo (simulated annealing, tabu search, iterated local search, VNS, GRASP, LNS, algoritmi genetici e memetici) sono metaeuristiche

Che cos'è una metaeuristica

Definizione 1: Metaeuristica

*Una **metaeuristica** è una strategia generale che guida una ricerca locale oltre il primo ottimo locale, per esplorare altre regioni dell'insieme delle soluzioni ammissibili.*

- Una metaeuristica è uno *schema*, non un algoritmo: un algoritmo si ottiene scegliendo la rappresentazione, gli intorni, la ricerca locale e i parametri.
- Ogni metaeuristica bilancia **intensificazione** e **diversificazione**, definite nel capitolo «Algoritmi euristici: definizioni».

Tre modi per uscire

Una ricerca locale si ferma nel primo ottimo locale;

una metaeuristica ne esce

- **accettando soluzioni peggiori**,
- **saltando** a una nuova soluzione di partenza,
- o **combinando** più soluzioni buone.

Riepilogo

Metaeuristica:

- una strategia generale che guida una ricerca locale oltre il primo ottimo locale
- uno schema, non un algoritmo: vanno scelti rappresentazione, intorni, ricerca locale e parametri

Come esce:

- accettando soluzioni peggiori, saltando a una nuova partenza, o combinando soluzioni buone
- bilanciando sempre intensificazione e diversificazione

I problemi e le ipotesi

Introduzione: I problemi e le ipotesi

- Fissiamo il quadro di tutto il capitolo: i quattro problemi, i loro intorni, i numeri casuali e le istanze
- Ogni esempio del capitolo si può verificare a mano, perché i numeri casuali sono assegnati e le istanze sono fissate
- Le istanze medie, di cui si conoscono i valori ottimi, servono a misurare quanto sono buone le metaeuristiche

I quattro problemi

- Le metaeuristiche sono illustrate 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),
  - $P \,||\, C_{\max}$.
- Le loro definizioni e ipotesi sono nelle rispettive pagine; ne richiamiamo la notazione.

La notazione dei quattro problemi

problema	dati	soluzione	obiettivo
KP	oggetti $\mathscr{I}$, profitti p_j, pesi w_j, capacità c	sottoinsieme	max profitto totale
BPP	oggetti $\mathscr{I}$, pesi w_j, capacità c	pattern P_b	min contenitori
TSP	città $\mathscr{V}$, distanze d_{ij}	giro σ	min lunghezza $L(\sigma)$
$P \mid\mid C_{\max}$	lavori $\mathscr{J}$, tempi t_j, macchine m	insiemi $J_1, \ldots, J_m$	min makespan

Ipotesi e notazione

- Ogni istanza soddisfa le ipotesi del suo problema:
  - $w_j \leq c$ per tutti gli oggetti e $\sum_j w_j > c$ per il KP e il BPP;
  - $n \geq 3$ e distanze simmetriche per il TSP;
  - $n > m$ per $P \,||\, C_{\max}$.
- Per $P \,||\, C_{\max}$, $L_h = \sum_{j \in J_h} t_j$ è il **carico** della macchina h e $C_{\max} = \max_h L_h$.
- Come nel capitolo «Algoritmi euristici: definizioni», $\mathscr{A}(I)$ è il valore della soluzione calcolata dall'algoritmo $\mathscr{A}$ sull'istanza I, e $z(I)$ è il valore ottimo.
- Scriviamo $f(S)$ per il valore di una soluzione S, e descriviamo gli algoritmi per un problema di **minimo**; per il KP, problema di massimo, le disuguaglianze si invertono.

Gli intorni (1/2)

- Tutte le metaeuristiche di questo capitolo sono costruite sugli intorni del capitolo «Ricerca locale». Li richiamiamo per nome, con le mosse che li definiscono; n è il numero di oggetti, di città o di lavori.

problema	nome	mossa	dimensione
KP	flip	aggiungere un oggetto allo zaino, o toglierne uno	n
	swap	togliere un oggetto selezionato e aggiungere un oggetto non selezionato	$\leq \frac{n^2}{4}$
BPP	move	spostare un oggetto in un altro contenitore	$\leq n \cdot nb$
	swap	scambiare due oggetti posti in contenitori diversi	$\leq \frac{n(n-1)}{2}$

Gli intorni (2/2)

problema	nome	mossa	dimensione
TSP	2-opt	togliere due lati non adiacenti del giro e richiuderlo con gli altri due lati, invertendo il segmento compreso fra loro	$\frac{n(n-3)}{2}$
	Or-opt	spostare un segmento di 1, 2 o 3 città consecutive in un'altra posizione	$O(n^2)$
$P \mid\mid C_{\max}$	move	spostare un lavoro su un'altra macchina	$n(m-1)$
	swap	scambiare due lavori assegnati a macchine diverse	$\leq \frac{n(n-1)}{2}$

La mossa 2-opt

- Un giro del TSP è memorizzato come una permutazione $\sigma = (\sigma_1, \ldots, \sigma_n)$ con $\sigma_1 = 1$.
- La **mossa 2-opt** (i, j), con $1 \leq i$, $i + 2 \leq j \leq n$ e $(i, j) \neq (1, n)$:
  - toglie i lati (σ_i, σ_{i+1}) e (σ_j, σ_{j+1}) (con $\sigma_{n+1} = \sigma_1$),
  - aggiunge i lati (σ_i, σ_j) e $(\sigma_{i+1}, \sigma_{j+1})$,
  - inverte il segmento $\sigma_{i+1}, \ldots, \sigma_j$.
- Il suo effetto sulla lunghezza si calcola in $O(1)$:

$$\delta(i, j) \;=\; d_{\sigma_i \sigma_j} + d_{\sigma_{i+1} \sigma_{j+1}} - d_{\sigma_i \sigma_{i+1}} - d_{\sigma_j \sigma_{j+1}}.$$

Obiettivi piatti e obiettivi ausiliari

- Gli obiettivi del BPP e di $P \,||\, C_{\max}$ sono **piatti**: la maggior parte delle mosse non cambia il numero di contenitori, né il makespan.
- Due soluzioni con lo stesso valore si confrontano allora con un **obiettivo ausiliario**:
  - BPP: la somma dei quadrati dei carichi $\sum_b \left(\sum_{j \in P_b} w_j \right)^2$, da *massimizzare*: cresce quando gli oggetti si concentrano in meno contenitori, più pieni;
  - $P \,||\, C_{\max}$: la somma dei quadrati dei carichi $\sum_h L_h^2$, da *minimizzare*: diminuisce quando i carichi si bilanciano.

Confronto lessicografico per $P \,||\, C_{\max}$

- Per $P \,||\, C_{\max}$ confrontiamo due soluzioni in modo **lessicografico**:
  - prima per $C_{\max}$,
  - e a parità di makespan per $\sum_h L_h^2$.

- Per move e swap questo seleziona le stesse mosse miglioranti dell'obiettivo ausiliario $\sum_h L_h^2$ del capitolo «Ricerca locale»:
  - un trasferimento che diminuisce $\sum_h L_h^2$ non aumenta mai $C_{\max}$,
  - e un trasferimento che diminuisce $C_{\max}$ diminuisce anche $\sum_h L_h^2$.

Soluzioni inammissibili e ricerche locali

- Per il KP l'intorno flip contiene soluzioni inammissibili (aggiungere un oggetto può superare la capacità). Una ricerca locale le scarta oppure le accetta con una **penalità** sul peso in eccesso; in questo capitolo compaiono entrambe le opzioni.

- Le **ricerche locali** usate dentro le metaeuristiche sono le discese del capitolo «Ricerca locale»:
  - TwoOptLS (2-opt, best improvement) per il TSP,
  - MoveDescent (move, best improvement, confronto lessicografico) per $P \,||\, C_{\max}$,
  - KPLocalSearch (flip e swap ammissibili, best improvement) per il KP.

- In ogni discesa best improvement, i pareggi fra mosse ugualmente buone si risolvono a favore della prima mossa trovata, scorrendo gli indici in ordine crescente.

Numeri casuali

- La maggior parte delle metaeuristiche fa **scelte casuali**. Per rendere riproducibili ogni esempio e ogni esercizio, tutti i numeri casuali di questo capitolo, e del suo codice C++, sono prodotti dal **minimal standard generator** di Park e Miller (1988):

$$seed \leftarrow 16807 \cdot seed \bmod (2^{31}-1), \qquad u = \frac{seed}{2^{31}-1} \in (0,1).$$

- Una chiamata $\textsc{Rand}()$ aggiorna *seed* e restituisce u.
- Un intero casuale in $\{0, 1, \ldots, K-1\}$ è $\lfloor u \cdot K \rfloor$, e un elemento casuale di una lista di K elementi è l'elemento in posizione $1 + \lfloor u \cdot K \rfloor$.

Il seme

- Il valore iniziale di *seed*, il **seme**, è un intero in $\{1, \ldots, 2^{31} - 2\}$.
- Lo stesso seme dà sempre la stessa sequenza di numeri; semi diversi danno esecuzioni diverse dello stesso algoritmo.
- Con $seed = 1$ i primi numeri sono, arrotondati a quattro decimali,

$$0.0000,\ 0.1315,\ 0.7556,\ 0.4587,$$
$$0.5328,\ 0.2190,\ 0.0470,\ 0.6789,\ \ldots$$

  Il primo numero prodotto da un seme piccolo è sempre molto vicino a 0; è una proprietà del generatore, non un errore.
- Negli esempi svolti i numeri casuali sono scritti nel testo, arrotondati a quattro decimali, così che ogni passo si possa verificare a mano; l'arrotondamento non cambia mai una decisione.

Le istanze di riferimento

- Gli esempi svolti usano le quattro **istanze di riferimento** del capitolo «Algoritmi euristici: definizioni», di cui sono noti i valori ottimi:

problema	istanza di riferimento	valore ottimo e soluzione
KP	$p = (10, 7, 6, 4)$, $w = (5, 4, 3, 3)$, $c = 9$	$z(I) = 17$, oggetti $\{1, 2\}$
BPP	$w = (4, 4, 5, 3, 2, 3)$, $c = 7$	$z(I) = 3$, pattern $\{1, 4\}, \{2, 6\}, \{3, 5\}$
TSP	5 città, distanze d_{ij} nella slide successiva	$z(I) = 18$, giro $(1, 3, 5, 2, 4)$
$P \mid\mid C_{\max}$	$t = (5, 5, 4, 4, 3, 3, 3)$, $m = 3$	$z(I) = 9$, insiemi $\{1, 3\}, \{2, 4\}, \{5, 6, 7\}$

L'istanza di riferimento del TSP

Le distanze d_{ij} delle 5 città dell'istanza di riferimento del TSP:

d_{ij}	1	2	3	4	5
1	—	5	2	2	9
2	5	—	4	3	4
3	2	4	—	4	7
4	2	3	4	—	7
5	9	4	7	7	—

Il giro ottimo è $(1, 3, 5, 2, 4)$, di lunghezza $z(I) = 18$.

Le istanze medie

- Le istanze di riferimento sono abbastanza piccole da seguire a mano ogni iterazione, ma troppo piccole per mostrare il comportamento di una metaeuristica su migliaia di iterazioni.
- Per gli esperimenti usiamo un'**istanza media** per problema, generata una volta con il generatore di numeri casuali visto sopra e fissata: TSP20, KP20, PCM15 e BPP24.
- I loro valori ottimi sono stati calcolati con un algoritmo esatto, e servono solo a misurare la qualità delle soluzioni euristiche.

Esempio: l'istanza media del TSP (TSP20) (1/2)

- 20 città nel piano, con coordinate intere (x_i, y_i), e distanze

$$d_{ij} = \left\lfloor \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + \frac{1}{2} \right\rfloor$$

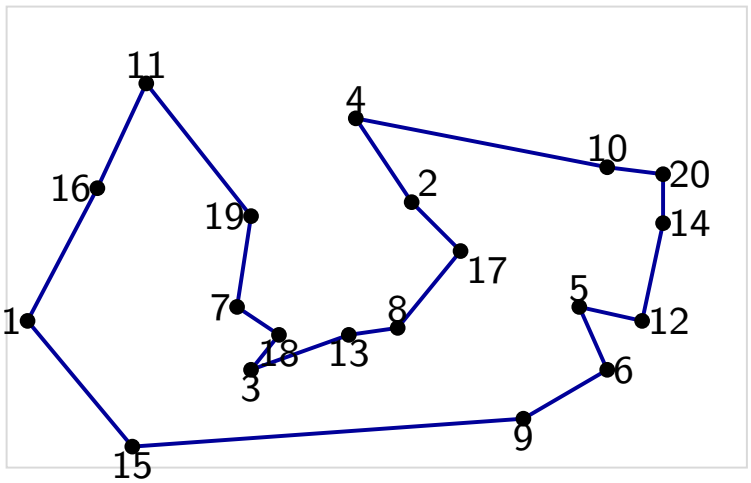
(la distanza euclidea arrotondata all'intero più vicino):

i	1	2	3	4	5	6	7	8	9	10
x_i	0	55	32	47	79	83	30	53	71	83
y_i	18	35	11	47	20	11	20	17	4	40

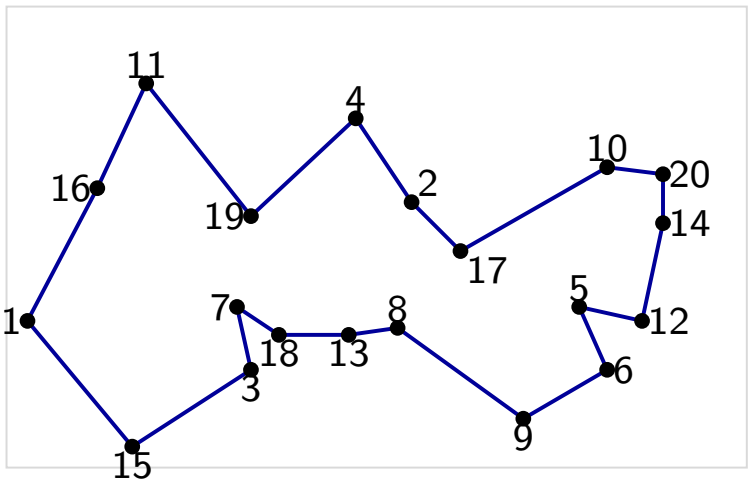
i	11	12	13	14	15	16	17	18	19	20
x_i	17	88	46	91	15	10	62	36	32	91
y_i	52	18	16	32	0	37	28	16	33	39

Esempio: l'istanza media del TSP (TSP20) (2/2)

nearest neighbour dalla città 1: $L(\sigma) = 326$



un giro ottimo: $L(\sigma) = 291$



- Il giro nearest neighbour dalla città 1 ha lunghezza 326; la ricerca locale 2-opt avviata da esso si ferma in un ottimo locale di lunghezza 319, dopo due mosse miglioranti.
- Il valore ottimo è $z(I) = 291$: l'ottimo locale è superiore del 9.6%.

Esempio: le istanze medie del KP, del BPP e di $P \,||\, C_{\max}$ (1/2)

KP20: $n = 20$ oggetti, capacità $c = 389$ (metà del peso totale, arrotondata per difetto),

j	1	2	3	4	5	6	7	8	9	10
p_j	54	55	55	52	58	42	53	53	22	64
w_j	39	43	49	47	44	35	43	44	16	50

j	11	12	13	14	15	16	17	18	19	20
p_j	54	51	39	49	53	46	28	64	58	34
w_j	40	42	28	38	46	39	14	50	45	27

- L'algoritmo greedy per efficienza trova profitto 490, e la ricerca locale con flip e swap avviata da esso arriva a 514; il valore ottimo è $z(I) = 521$.

Esempio: le istanze medie del KP, del BPP e di $P \,||\, C_{\max}$ (2/2)

PCM15: $m = 4$ macchine e $n = 15$ lavori con tempi di esecuzione

$$t = (10, 23, 26, 16, 23, 11, 14, 18, 18, 24, 29, 22, 13, 13, 15).$$

Il tempo totale è 275, quindi $z(I) \geq \lceil 275/4 \rceil = 69$, ed esiste un assegnamento di makespan 69: $z(I) = 69$. LPT trova makespan 73.

BPP24: contenitori di capacità $c = 100$ e $n = 24$ oggetti con pesi

$$w = (30, 40, 57, 22, 37, 23, 34, 34, 55, 51, 21, 47,$$
$$22, 24, 22, 34, 20, 51, 25, 54, 35, 27, 55, 48).$$

Il peso totale è 868, quindi servono almeno $\lceil 868/100 \rceil = 9$ contenitori, ed esiste un impacchettamento in 9 contenitori: $z(I) = 9$. FF e FFD usano entrambi 10 contenitori.

Riepilogo (1/2)

I quattro problemi:

- KP, BPP, TSP e $P \,||\, C_{\max}$, con la notazione delle loro pagine
- algoritmi descritti per il minimo; per il KP le disuguaglianze si invertono

Intorni e ricerche locali:

- flip e swap (KP), move e swap (BPP, $P \,||\, C_{\max}$), 2-opt e Or-opt (TSP)
- obiettivi piatti: obiettivo ausiliario, confronto lessicografico per $P \,||\, C_{\max}$
- TwoOptLS, MoveDescent, KPLocalSearch: best improvement, prima mossa a parità

Riepilogo (2/2)

Numeri casuali:

- minimal standard generator di Park e Miller: lo stesso seme dà la stessa esecuzione
- negli esempi i numeri sono scritti nel testo, arrotondati a quattro decimali

Istanze:

- quattro istanze di riferimento, abbastanza piccole da seguirle a mano
- quattro istanze medie, TSP20, KP20, PCM15 e BPP24, con valori ottimi 291, 521, 69 e 9

Come uscire da un ottimo locale

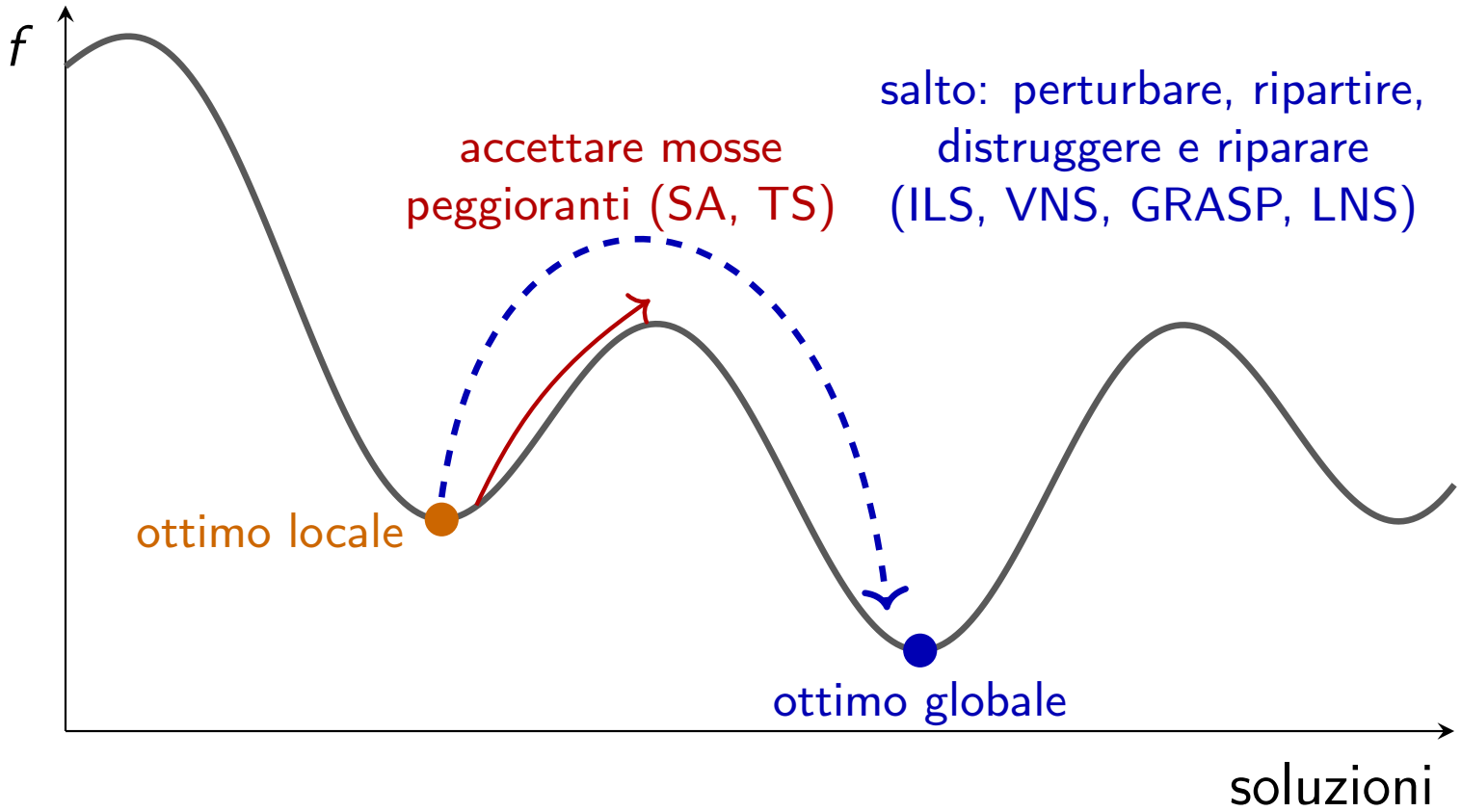
Introduzione: Come uscire da un ottimo locale

- Una ricerca locale resta bloccata al fondo della valle da cui parte
- In questa sezione elenchiamo i modi per uscire da un ottimo locale, e la metaeuristica che usa ciascuno di essi
- L'elenco è la mappa del capitolo: ognuna delle sezioni successive presenta un modo

Ricerca locale e multi-start

- Una ricerca locale passa da una soluzione a un vicino migliore, e si ferma nel primo **ottimo locale**: una soluzione senza vicini migliori.
- Nel paesaggio della funzione obiettivo, scende fino al fondo della valle da cui parte, e lì rimane. È pura **intensificazione**.
- Ripetere la ricerca locale da molte soluzioni di partenza (multi-start) esplora molte valli, ma
  - ogni partenza ignora tutto ciò che è stato appreso prima,
  - e le valli buone si trovano di solito vicino ad altre valli buone.

Il paesaggio della funzione obiettivo



I modi per uscire (1/3)

come uscire	idea	metaeuristica
accettare mosse peggioranti *a caso*	un vicino peggiore è accettato con una probabilità che diminuisce con il peggioramento e con il tempo	simulated annealing
accettare mosse peggioranti *in modo deterministico, con memoria*	passare al miglior vicino anche se è peggiore, e vietare il ritorno alle soluzioni appena visitate	tabu search

I modi per uscire (2/3)

come uscire	idea	metaeuristica
perturbare l'ottimo locale e ripartire	applicare una modifica casuale che la ricerca locale non può annullare, e scendere di nuovo	iterated local search
cambiare l'intorno	quando la ricerca è bloccata, usare un intorno più grande, di cui l'ottimo locale non è più un ottimo locale	variable neighbourhood search

I modi per uscire (3/3)

come uscire	idea	metaeuristica
ripartire da una costruzione randomizzata	costruire una soluzione buona diversa a ogni ripartenza, e migliorarla	GRASP
distruggere e riparare	togliere una parte grande della soluzione e ricostruirla con un'euristica o un algoritmo esatto	large neighbourhood search
lavorare con una *popolazione*	combinare le parti di più soluzioni buone in soluzioni nuove	algoritmi genetici e memetici

Camminare, saltare, o tenere molte soluzioni

- I primi due modi mantengono **una sola soluzione corrente** e cambiano la **regola di accettazione** della ricerca locale: escono dalla valle passando per soluzioni peggiori.
- I quattro successivi **saltano**: generano un nuovo punto di partenza, abbastanza lontano da lasciare la valle ma abbastanza vicino da conservare ciò che è buono.
- L'ultimo mantiene **molte soluzioni** contemporaneamente.
- Ogni metaeuristica deve bilanciare **intensificazione** (scendere fino al fondo della valle corrente) e **diversificazione** (lasciarla per un'altra). Per ciascun metodo indichiamo quale componente fa che cosa.

Uno schema, non un algoritmo

- Una metaeuristica è uno *schema*, non un algoritmo: per ottenere un algoritmo occorre scegliere
  - la rappresentazione,
  - gli intorni,
  - la ricerca locale,
  - e i parametri,

  e queste scelte contano almeno quanto lo schema.

- Presentiamo ogni metaeuristica sui problemi in cui la sua idea è più naturale, non su tutti e quattro.

Riepilogo

Perché uscire:

- una ricerca locale si ferma al fondo della sua valle: pura intensificazione
- il multi-start esplora molte valli ma ignora ciò che ha appreso

Come uscire:

- camminare attraverso soluzioni peggiori: simulated annealing, tabu search
- saltare: iterated local search, VNS, GRASP, large neighbourhood search
- tenere una popolazione: algoritmi genetici e memetici

Simulated annealing

Introduzione: Simulated annealing

- Il simulated annealing esce da un ottimo locale accettando mosse peggioranti *a caso*
- Vediamo il suo pseudocodice, la regola di accettazione e i parametri, poi lo eseguiamo a mano sul TSP e sul KP, e sull'istanza media del TSP
- È la prima metaeuristica del capitolo, e la più semplice da implementare

L'idea del simulated annealing

- Il *simulated annealing* (indicato con SA) è una ricerca locale che sceglie un vicino *casuale* della soluzione corrente e vi si sposta se è migliore.
- Se è peggiore, vi si sposta comunque con una probabilità che diminuisce con il peggioramento e con un parametro T, la *temperatura*.
- La temperatura parte alta, così che all'inizio quasi ogni mossa è accettata, e viene abbassata gradualmente, così che alla fine sono accettate solo mosse miglioranti.
- Si restituisce la migliore soluzione visitata.

Lo pseudocodice di SA, per un problema di minimo con obiettivo f e intorno N,
è nella slide successiva.

Algorithm: Simulated annealing (1/2)

Input : soluzione di partenza S, temperatura iniziale T_0, fattore di raffreddamento $\alpha \in (0,1)$, numero L di mosse per temperatura, temperatura finale $T_{\min}$

Output : migliore soluzione trovata

```
1 function SimulatedAnnealing(S, T_0, α, L, T_min)
2 |   cur := S;  fcur := f(cur)        // soluzione corrente
3 |   best := cur;  fbest := fcur      // migliore soluzione
  |    trovata
4 |   T := T_0                                  // temperatura
5 |   while T > T_min do
6 |   |   for l := 1 to L do
7 |   |   |   S' := vicino casuale in N(cur)
8 |   |   |   δ := f(S') − fcur
9 |   |   |   v := RAND()
```

Algorithm: Simulated annealing (2/2)

```
function SimulatedAnnealing(S, T_0, α, L, T_min) (continua)
    while T > T_min (continua) do
        for l := 1 to L (continua) do
            if δ ≤ 0 or v < e^{-δ/T} then
                cur := S'; fcur := fcur + δ          // mossa
                 accettata
                if fcur < fbest then
                    best := cur; fbest := fcur
                end
            end
        end
        T := α · T                                   // raffreddamento
    end
    return best
end
```

Da dove viene il nome

- Il nome viene dalla metallurgia: l'*annealing* (ricottura) è il processo che scalda un metallo e lo raffredda lentamente, così che i suoi atomi si dispongano in uno stato di energia minima.
- Raffreddato troppo in fretta, il metallo si blocca in uno stato disordinato di energia più alta.
- Kirkpatrick, Gelatt e Vecchi (1983) e, indipendentemente, Černý (1985) hanno proposto di simulare questo processo su un problema di ottimizzazione, con l'obiettivo f nel ruolo dell'energia.
- La regola di accettazione è quella usata da Metropolis et al. (1953) per simulare gli atomi.

Regola di accettazione di Metropolis

- Alla temperatura T, un vicino S' della soluzione corrente, con $\delta = f(S') - f(cur)$, è accettato con probabilità

$$P(\text{accettare } S') = \begin{cases} 1 & \text{se } \delta \leq 0, \\ e^{-\delta/T} & \text{se } \delta > 0. \end{cases}$$

- La probabilità si realizza estraendo $v = \textsc{Rand}()$, uniforme in $(0, 1)$, e accettando se $v < e^{-\delta/T}$, il che accade con probabilità $e^{-\delta/T}$.

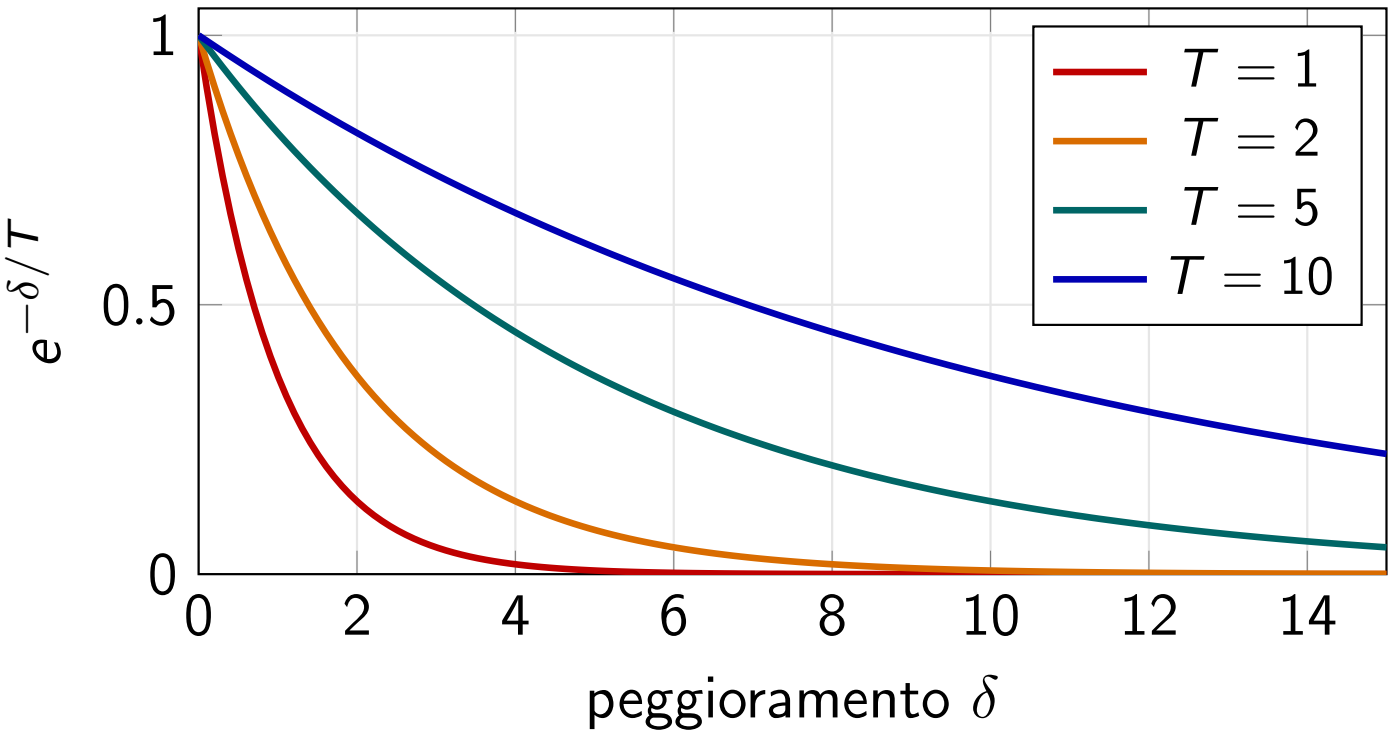
Come funziona la regola (1/2)

- Una mossa peggiorante è accettata più facilmente se il peggioramento δ è **piccolo**, e se la temperatura T è **alta**.
- Per $T \to \infty$ ogni mossa è accettata, e *SA* diventa una passeggiata casuale (pura diversificazione).
- Per $T \to 0$ sono accettate solo le mosse con $\delta \leq 0$, e *SA* diventa una ricerca locale con scelta casuale del vicino (pura intensificazione).
- Abbassare la temperatura porta gradualmente dal primo comportamento al secondo.
- Le mosse con $\delta = 0$ sono sempre accettate: *SA* si muove liberamente sui plateau degli obiettivi piatti.

Come funziona la regola (2/2)

- *SA* non si ferma mai in un ottimo locale: un ottimo locale viene lasciato appena una mossa peggiorante è estratta e accettata.
- Per questo la soluzione **migliore** visitata va memorizzata a parte: la soluzione corrente alla fine non è necessariamente la migliore.
- A ogni iterazione si valuta **un solo vicino casuale**, non l'intero intorno: un'iterazione di *SA* costa pochissimo, e *SA* esegue molte più iterazioni di una ricerca locale.
- Per il TSP, una mossa 2-opt casuale si estrae numerando le $M = \frac{n(n-3)}{2}$ mosse (i, j) in ordine lessicografico e scegliendo quella in posizione $1 + \lfloor \textsc{Rand}() \cdot M \rfloor$; il suo δ si calcola in $O(1)$.

La probabilità di accettazione



- A $T = 10$ un peggioramento di 5 è accettato con probabilità $e^{-0.5} \approx 0.61$; a $T = 1$ con probabilità $e^{-5} \approx 0.007$.

Temperatura e peggioramento

- La figura mostra la probabilità di accettazione in funzione del peggioramento, per quattro temperature.
- La temperatura ha la stessa unità di misura dell'obiettivo: T è «grande» o «piccola» solo rispetto ai valori tipici di δ.

Vediamo ora come scegliere la successione delle temperature e gli altri parametri.

Lo schema di raffreddamento

- Lo **schema di raffreddamento** è la successione delle temperature.
- Lo pseudocodice usa lo schema **geometrico** $T \leftarrow \alpha T$, il più comune in pratica, con α fra 0.8 e 0.99, e L mosse a ogni temperatura.
- Altri schemi sono
  - quello *lineare*, $T_k = T_0 - k\beta$,
  - e quello *logaritmico*, $T_k = \frac{\Gamma}{\ln(k+1)}$, che conta nella teoria di *SA* ma è decisamente troppo lento in pratica.

I parametri (1/2)

- **Temperatura iniziale** T_0**.** Deve essere abbastanza alta da accettare all'inizio la maggior parte delle mosse peggioranti. Una regola comune
  - fissa un tasso di accettazione iniziale obiettivo χ_0 (per esempio $\chi_0 = 0.8$),
  - stima il peggioramento medio $\bar{\delta}$ di un campione di mosse casuali,
  - e pone $T_0 = -\bar{\delta}/\ln \chi_0$, così che $e^{-\bar{\delta}/T_0} = \chi_0$.

- **Mosse per temperatura** L**.** Bisogna dare alla ricerca il tempo di adattarsi a ogni temperatura; L si prende spesso proporzionale alla dimensione dell'intorno.

I parametri (2/2)

- **Criterio di arresto.** Lo pseudocodice si ferma quando $T \leq T_{\text{min}}$.
  - Con lo schema geometrico il numero di temperature è il più piccolo intero K con $T_0 \alpha^K \leq T_{\text{min}}$, cioè $K = \lceil \ln(T_{\text{min}}/T_0)/\ln \alpha \rceil$ quando il rapporto non è intero,
  - e il numero di iterazioni è $K \cdot L$.

  Altri criteri comuni si fermano dopo un certo numero di temperature senza mosse accettate, o a un limite di tempo.

- Un raffreddamento lento (α vicino a 1) dà soluzioni migliori ma più iterazioni: i parametri scambiano qualità con tempo, come in ogni euristica.

Esempio: *SA* sull'istanza di riferimento del TSP (1/3)

- Partiamo dal giro calcolato dall'euristica nearest neighbour dalla città 1 nel capitolo «Euristiche costruttive», $\sigma = (1, 3, 2, 4, 5)$ di lunghezza 25.
- Usiamo $T_0 = 5$, $\alpha = 0.8$, $L = 2$ e $T_{\text{min}} = 3$: le temperature sono 5, 4 e 3.2 (poi $2.56 \leq 3$ ferma la ricerca), per un totale di 6 iterazioni.
- Le $M = 5$ mosse 2-opt su 5 città sono numerate

$$1 : (1, 3), \quad 2 : (1, 4), \quad 3 : (2, 4), \quad 4 : (2, 5), \quad 5 : (3, 5).$$

- I numeri casuali, con *seed* $= 1$, sono quelli della sottosezione «Numeri casuali»: il primo numero di ogni iterazione (u) sceglie la mossa $1 + \lfloor 5u \rfloor$, il secondo (v) serve nel test di accettazione.

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

it	T	u	mossa	δ	v	$e^{-\delta/T}$	decisione	giro corrente	L
1	5	0.0000	$1:(1,3)$	4	0.1315	0.4493	accetta	$(1,2,3,4,5)$	29
2	5	0.7556	$4:(2,5)$	-7	—	—	accetta	$(1,2,5,4,3)$	22
3	4	0.5328	$3:(2,4)$	2	0.2190	0.6065	accetta	$(1,2,4,5,3)$	24
4	4	0.0470	$1:(1,3)$	-6	—	—	accetta	$(1,4,2,5,3)$	18
5	3.2	0.6793	$4:(2,5)$	4	0.9347	0.2865	rifiuta	$(1,4,2,5,3)$	18
6	3.2	0.3835	$2:(1,4)$	4	0.5194	0.2865	rifiuta	$(1,4,2,5,3)$	18

Esempio: *SA* sull'istanza di riferimento del TSP (3/3)

- Alle iterazioni 1 e 3 viene accettato un giro **peggiore** ($25 \to 29$ e $22 \to 24$), perché v è minore della probabilità di accettazione.
- Alle iterazioni 5 e 6 la temperatura è più bassa, le mosse peggioranti hanno meno probabilità di essere accettate, ed entrambe sono rifiutate.
- Il giro migliore, $(1, 4, 2, 5, 3)$, è il giro ottimo $(1, 3, 5, 2, 4)$ percorso nel verso opposto: $SA(I) = 18 = z(I)$.
- Su questa istanza piccola il giro ottimo è l'unico ottimo locale 2-opt fra i 12 giri, quindi anche una ricerca locale 2-opt lo raggiungerebbe: l'esempio mostra il meccanismo di *SA*; la sua capacità di uscire dagli ottimi locali si vede su istanze con ottimi locali cattivi.

Simulated annealing con penalità

- Per il KP, l'intorno flip di una soluzione vicina alla capacità contiene molte soluzioni inammissibili. Invece di scartarle, *SA* può **visitarle**, con una penalità sul peso in eccesso che le rende meno attraenti:

$$f_\beta(x) = \sum_{j \in \mathscr{I}} p_j x_j - \beta \cdot \max\left\{0, \sum_{j \in \mathscr{I}} w_j x_j - c\right\}, \qquad \beta > 0.$$

- *SA* massimizza f_β su tutti gli $x \in \{0,1\}^n$.
- Il peggioramento di una mossa da x a y è $\delta = f_\beta(x) - f_\beta(y)$.
- La soluzione migliore si aggiorna solo con soluzioni **ammissibili**.

La regione inammissibile come ponte

- Su una soluzione ammissibile f_β è il profitto.
- Passando per una soluzione inammissibile, *SA* può raggiungere soluzioni ammissibili che non sono raggiungibili con soli flip ammissibili: la regione inammissibile fa da **ponte** fra le valli.
- Il **coefficiente di penalità** β è un parametro:
  - se è troppo piccolo la ricerca vaga nella regione inammissibile, dove il profitto è maggiore;
  - se è troppo grande le soluzioni inammissibili non sono mai accettate, e il ponte è chiuso.

Esempio: *SA* con penalità sull'istanza di riferimento del KP (1/3)

- Partiamo da $x = (1, 0, 1, 0)$, la soluzione $\{1, 3\}$ dell'algoritmo greedy per efficienza, di profitto 16.
- È un ottimo locale per l'intorno flip (capitolo «Algoritmi euristici: definizioni»): i suoi vicini ammissibili hanno profitto 6 e 10.
- Usiamo $\beta = 3$, $T_0 = 4$, $\alpha = 0.8$, $L = 2$, $T_{\text{min}} = 2.5$ (temperature 4, 3.2, 2.56: 6 iterazioni) e $seed = 3$.
- A ogni iterazione u sceglie l'oggetto $k = 1 + \lfloor 4u \rfloor$ da invertire, e v serve nel test di accettazione.

Esempio: *SA* con penalità sull'istanza di riferimento del KP (2/3)

it	T	u	k	y	$\sum w_j y_j$	$f_3(y)$	δ	v vs $e^{-\delta/T}$	decisione
1	4	0.0000	1	$(0,0,1,0)$	3	6	10	$0.3946 \geq 0.0821$	rifiuta
2	4	0.2668	2	$(1,1,1,0)$	12	14	2	$0.3760 < 0.6065$	accetta
3	3.2	0.5983	3	$(1,1,0,0)$	9	17	-3	—	accetta
4	3.2	0.1411	1	$(0,1,0,0)$	4	7	10	$0.0366 < 0.0439$	accetta
5	2.56	0.0379	1	$(1,1,0,0)$	9	17	-10	—	accetta
6	2.56	0.1505	1	$(0,1,0,0)$	4	7	10	$0.5582 \geq 0.0201$	rifiuta

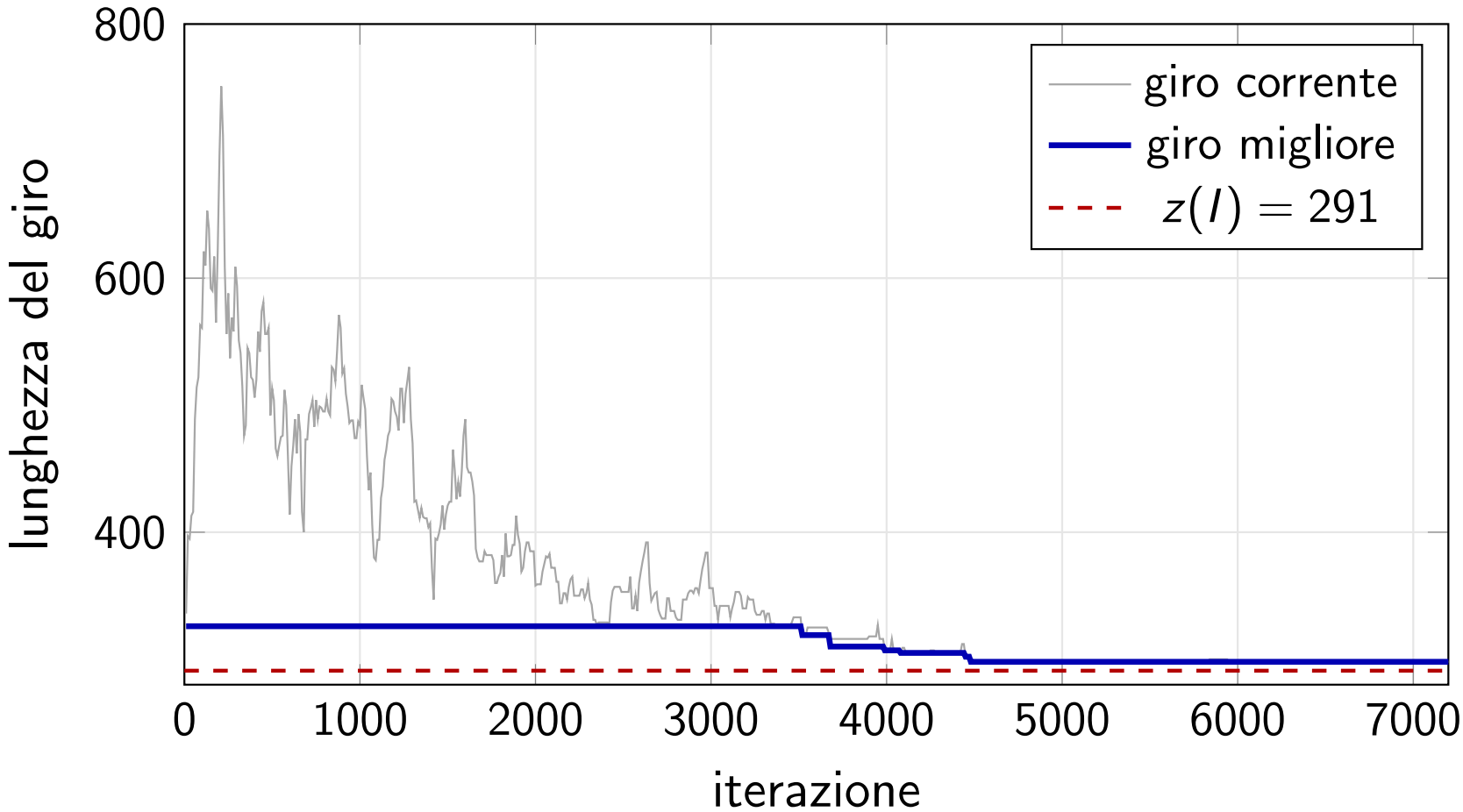
Esempio: *SA* con penalità sull'istanza di riferimento del KP (3/3)

- All'iterazione 2 la ricerca passa alla soluzione **inammissibile** $(1, 1, 1, 0)$, di peso $12 > 9$: il suo profitto è 23 e il suo valore penalizzato $23 - 3 \cdot 3 = 14$, un peggioramento di 2 accettato con probabilità 0.6065.
- Da lì, togliendo l'oggetto 3 si ottiene la soluzione ammissibile $(1, 1, 0, 0)$ di profitto 17: l'ottimo, $SA(I) = 17 = z(I)$.
- L'iterazione 4 mostra che a questa temperatura anche un peggioramento di 10 può essere accettato, con una probabilità piccola; l'iterazione 5 torna all'ottimo.

Un esperimento su TSP20

- Eseguiamo *SA* con l'intorno 2-opt su TSP20, dal giro nearest neighbour di lunghezza 326, con $L = 100$ e $T_{\min} = 0.5$.
- La figura della slide successiva mostra un'esecuzione, con $T_0 = 20$, $\alpha = 0.95$ e *seed* $= 1$:
  - la lunghezza del giro corrente (grigio) e del giro migliore (blu),
  - lungo le 7200 iterazioni, ogni 10 iterazioni.
- I dati sono prodotti dal codice `C++` di questo capitolo (`make data`).

Un'esecuzione di *SA* su TSP20



Che cosa mostra l'esecuzione

- All'inizio la temperatura è alta rispetto al peggioramento tipico di una mossa 2-opt, e il giro corrente diventa molto **peggiore** di quello di partenza (fino a lunghezza 765): la ricerca è quasi una passeggiata casuale.
- Al diminuire di T, sono accettate meno mosse peggioranti e il giro corrente scende in una valle; alla fine la ricerca è una ricerca locale.
- Il giro migliore, di lunghezza 298, è trovato all'iterazione 4479: è migliore dell'ottimo locale 2-opt (319), ma non è ottimo.
- La tabella della slide successiva riporta, per 10 esecuzioni con $seed = 1, \ldots, 10$ e diversi parametri, il numero di iterazioni, la lunghezza minima, media e massima del giro migliore, e il numero di esecuzioni che raggiungono l'ottimo 291.

SA su TSP20 con dieci semi

α	T_0	iterazioni	min	media	max	esecuzioni ottime
0.9	1	700	291	310.3	322	1
0.9	5	2200	291	297.3	305	4
0.9	20	3600	291	300.3	323	4
0.9	100	5100	291	297.8	311	3
0.95	1	1400	291	302.3	319	3
0.95	5	4500	291	294.3	298	5
0.95	20	7200	291	294.3	298	5
0.95	100	10400	291	293.2	299	7
0.99	1	6900	291	298.8	305	4
0.99	5	23000	291	291.7	298	9
0.99	20	36800	291	291.0	291	10
0.99	100	52800	291	291.0	291	10

L'effetto dei parametri

- Con $T_0 = 1$ la temperatura è troppo bassa fin dall'inizio: sono accettate poche mosse peggioranti, e *SA* si comporta come una ricerca locale randomizzata.
- Un raffreddamento più lento ($\alpha = 0.99$) passa più iterazioni in ogni intervallo di temperatura e raggiunge l'ottimo in quasi tutte le esecuzioni, al prezzo di molte più iterazioni.
- Una singola esecuzione dice poco: il confronto deve guardare la distribuzione su più semi, come ricordato nel capitolo «Algoritmi euristici: definizioni».

La distribuzione di Boltzmann

- A temperatura fissata T, la successione delle soluzioni correnti di SA è una catena di Markov: la soluzione successiva dipende solo da quella corrente.
- Se l'intorno è
  - simmetrico ($S' \in N(S)$ se e solo se $S \in N(S')$, con tutti gli intorni della stessa dimensione)
  - e connesso (ogni soluzione può raggiungere ogni altra attraverso mosse),

  la catena ha un'unica distribuzione stazionaria, la **distribuzione di Boltzmann**:

$$\pi_T(S) = \frac{e^{-f(S)/T}}{\sum_{S'} e^{-f(S')/T}} \qquad \text{per ogni soluzione } S.$$

Convergenza al limite

- Per due soluzioni S_1 e S_2, $\frac{\pi_T(S_1)}{\pi_T(S_2)} = e^{(f(S_2)-f(S_1))/T}$: le soluzioni migliori sono più probabili, e tanto più quanto più bassa è la temperatura.
- Per $T \to 0$, π_T si concentra sulle soluzioni ottime.
- Hajek (1988) ha dimostrato che con lo schema logaritmico $T_k = \frac{\Gamma}{\ln(k+1)}$, con Γ almeno pari alla profondità della valle più profonda che non contiene una soluzione ottima, la probabilità che la soluzione corrente sia ottima tende a 1.
- Il risultato è una garanzia *al limite*: il numero di iterazioni necessarie può superare il numero di soluzioni, e in pratica si usa lo schema geometrico, più veloce, senza garanzie.

Riepilogo (1/2)

Simulated annealing:

- un vicino casuale per iterazione, accettato con la regola di Metropolis: sempre se $\delta \leq 0$, con probabilità $e^{-\delta/T}$ altrimenti
- temperatura alta: passeggiata casuale; temperatura bassa: ricerca locale
- la migliore soluzione visitata è memorizzata a parte

Parametri:

- raffreddamento geometrico $T \leftarrow \alpha T$, temperatura iniziale T_0, L mosse per temperatura, T_{min}
- raffreddamento lento: soluzioni migliori, più iterazioni

Riepilogo (2/2)

Esempi ed esperimenti:

- sulle istanze di riferimento del TSP e del KP *SA* raggiunge l'ottimo, accettando mosse peggioranti lungo la strada
- una penalità sul peso in eccesso permette a *SA* di attraversare la regione inammissibile, un ponte fra le valli
- su TSP20 il risultato dipende da T_0 e α: bisogna guardare più semi

Proprietà:

- a T fissata la distribuzione stazionaria è quella di Boltzmann
- lo schema logaritmico converge a un ottimo solo al limite

Tabu search

Introduzione: Tabu search

- La tabu search esce da un ottimo locale accettando mosse peggioranti *in modo deterministico*, con una memoria delle mosse recenti
- Vediamo il suo pseudocodice per $P \,||\, C_{\max}$, le sue componenti (attributi, aspirazione, durata tabu), e la eseguiamo a mano sull'istanza di riferimento
- Poi misuriamo l'effetto della durata tabu sulle istanze medie del TSP e di $P \,||\, C_{\max}$

L'idea della tabu search

- La *tabu search* (indicata con TS) esplora completamente l'intorno della soluzione corrente e si sposta sul *miglior* vicino, anche se è peggiore della soluzione corrente.
- Per evitare di tornare alle soluzioni appena visitate, mantiene una memoria a breve termine, la *lista tabu*: le mosse che annullerebbero le mosse recenti sono vietate (*tabu*) per un certo numero di iterazioni, la *durata tabu* (tenure),
- a meno che non portino a una soluzione migliore della migliore trovata (*criterio di aspirazione*).

Vediamo ora lo pseudocodice di TS per $P \,||\, C_{\max}$, con l'intorno move e il confronto lessicografico; l'attributo (j, h) vieta di riportare il lavoro j sulla macchina h.

Algorithm: Tabu search per $P \,||\, C_{\max}$ (1/3)

Input : tempi $t[1..n]$, m macchine, assegnamento di partenza $mach[1..n]$, $tenure$, $maxIter$

Output : migliore assegnamento trovato

```
1 function TabuSearch(t, m, mach, tenure, maxIter)
2     L := carichi di mach;  tabu[j][h] := 0 per ogni j, h
3     best := mach;  bestC := max_h L[h]
4     LB := max{max_j t[j], ⌈∑_j t[j]/m⌉}        // arresto se
        raggiunto
5     it := 0
```

Algorithm: Tabu search per $P \,||\, C_{\max}$ (2/3)

function *TabuSearch*$(t, m, mach, tenure, maxIter)$ (continua)

```
6   |   while it < maxIter and bestC > LB do
7   |   |   it := it + 1;  bj := 0 // migliore mossa non tabu
    |   |    (bj, bh)
8   |   |   for j := 1 to n do
9   |   |   |   for h := 1 to m, h ≠ mach[j] do
10  |   |   |   |   L' := i carichi dopo lo spostamento del lavoro
    |   |   |   |    j da mach[j] a h
11  |   |   |   |   if tabu[j][h] ≥ it and not max_k L'[k] < bestC
    |   |   |   |    then continue                          // tabu
12  |   |   |   |   if bj = 0 or L' è lessicograficamente migliore
    |   |   |   |    di bL then (bj, bh, bL) := (j, h, L')
13  |   |   |   end
14  |   |   end
```

Algorithm: Tabu search per $P \,||\, C_{\max}$ (3/3)

```
function TabuSearch(t, m, mach, tenure, maxIter) (continua)
    while it < maxIter and bestC > LB (continua) do
15      if bj = 0 then break
16      tabu[bj][mach[bj]] := it + tenure      // vietare il
         ritorno
17      mach[bj] := bh;  L := bL      // anche se peggiore
18      if max_k L[k] < bestC then
19          bestC := max_k L[k];  best := mach
20      end
21  end
22  return best
23 end
```

Deterministica, con memoria

- La tabu search è stata proposta da Glover (1986); il libro di riferimento è Glover e Laguna, *Tabu Search*, Kluwer, 1997.
- A differenza di *SA*, è **deterministica**: esce da un ottimo locale perché si sposta sempre, sul vicino meno cattivo se nessun vicino è migliore.
- Senza memoria, questa regola **cicla**:
  - da un ottimo locale S la ricerca passa al miglior vicino S', che è peggiore,
  - e da S' il miglior vicino è tipicamente di nuovo S.
- La lista tabu spezza il ciclo vietando la mossa inversa.

Attributi e aspirazione

- **Attributi.** Memorizzare le soluzioni visitate e confrontare con esse ogni vicino sarebbe troppo costoso. La lista tabu memorizza invece **attributi** delle mosse recenti:
  - qui la coppia (j, h) «il lavoro j ha lasciato la macchina h»;
  - per il TSP, i lati appena tolti dal giro.

- Un attributo vieta più della sola soluzione appena lasciata: ogni soluzione con il lavoro j sulla macchina h è tabu, anche quelle mai visitate.

- Il **criterio di aspirazione** rimedia a questo eccesso di prudenza: una mossa tabu è ammessa se porta a una soluzione migliore della migliore trovata, che certamente non è stata visitata.

La durata tabu

- **Durata tabu.** Se è troppo breve, la ricerca può ancora ciclare su cicli più lunghi; se è troppo lunga, sono vietate troppe mosse e la ricerca viene spinta lontano dalle regioni buone. Valori tipici sono fra 5 e 20, o proporzionali a $\sqrt{n}$; estrarre la durata a caso in un intervallo a ogni iterazione (robust tabu search, Taillard 1991) rende improbabili i cicli.

Criterio di arresto e memoria a lungo termine

- **Criterio di arresto.** Un numero massimo di iterazioni, o di iterazioni senza miglioramento della soluzione migliore. Per $P \,||\, C_{\max}$ possiamo anche fermarci quando il makespan raggiunge il lower bound LB: allora la soluzione è ottima, e la ricerca può solo perdere tempo.

- **Memoria a lungo termine.** Oltre alla lista tabu (memoria a breve termine), *TS* può registrare quanto spesso ogni attributo compare nelle soluzioni visitate. La *diversificazione* penalizza gli attributi frequenti, per spingere la ricerca verso regioni inesplorate; l'*intensificazione* riparte dalle migliori soluzioni trovate, e fissa gli attributi che hanno in comune.

Esempio: *TS* sull'istanza di riferimento di $P \,||\, C_{\max}$ (1/4)

- Partiamo dalla soluzione calcolata da LPT nel capitolo «Euristiche costruttive», $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$, con carichi $(11, 8, 8)$, $C_{\max} = 11$ e $\sum_h L_h^2 = 249$.
- È un ottimo locale per l'intorno move: alla prima iterazione i 14 vicini sono

mossa	$1 \to M_2$	$1 \to M_3$	$2 \to M_1$	$2 \to M_3$	$3 \to M_1$
$(C_{\max}, \sum_h L_h^2)$	$(13, 269)$	$(13, 269)$	$(16, 329)$	$(13, 299)$	$(15, 305)$
mossa	$3 \to M_2$	$4 \to M_1$	$4 \to M_2$	$5 \to M_2$	$5 \to M_3$
$(C_{\max}, \sum_h L_h^2)$	$(12, 281)$	$(15, 305)$	$(12, 281)$	$(11, 249)$	$(11, 249)$
mossa	$6 \to M_1$	$6 \to M_3$	$7 \to M_2$	$7 \to M_3$	
$(C_{\max}, \sum_h L_h^2)$	$(14, 285)$	$(11, 267)$	$(11, 249)$	$(11, 249)$	

Esempio: *TS* sull'istanza di riferimento di $P \,||\, C_{\max}$ (2/4)

- Nessun vicino ha $C_{\max} < 11$, e i quattro migliori hanno $C_{\max} = 11$ e $\sum_h L_h^2 = 249$, come la soluzione corrente: una ricerca locale si ferma qui.
- *TS*, con *tenure* $= 6$, si sposta invece sul miglior vicino non tabu (il primo dei quattro, il lavoro 5 su M_2), e prosegue come nella tabella della slide successiva.

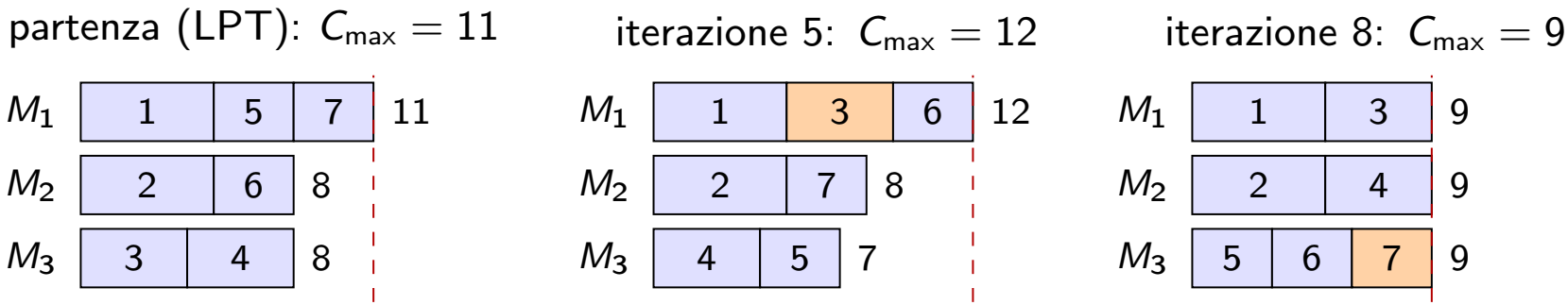
Esempio: *TS* sull'istanza di riferimento di $P \,||\, C_{\max}$ (3/4)

it	mossa	carichi	$C_{\max}$	$\sum_h L_h^2$	*bestC*	nuovo attributo tabu
1	lavoro 5: $M_1 \to M_2$	$(8, 11, 8)$	11	249	11	$(5, M_1)$ fino all'it. 7
2	lavoro 5: $M_2 \to M_3$	$(8, 8, 11)$	11	249	11	$(5, M_2)$ fino all'it. 8
3	lavoro 6: $M_2 \to M_1$	$(11, 5, 11)$	11	267	11	$(6, M_2)$ fino all'it. 9
4	lavoro 7: $M_1 \to M_2$	$(8, 8, 11)$	11	249	11	$(7, M_1)$ fino all'it. 10
5	lavoro 3: $M_3 \to M_1$	$(12, 8, 7)$	12	257	11	$(3, M_3)$ fino all'it. 11
6	lavoro 6: $M_1 \to M_3$	$(9, 8, 10)$	10	245	10	$(6, M_1)$ fino all'it. 12
7	lavoro 4: $M_3 \to M_2$	$(9, 12, 6)$	12	261	10	$(4, M_3)$ fino all'it. 13
8	lavoro 7: $M_2 \to M_3$	$(9, 9, 9)$	9	243	9	$(7, M_2)$ fino all'it. 14

Esempio: *TS* sull'istanza di riferimento di $P \,||\, C_{\max}$ (4/4)

- All'iterazione 2 il miglior vicino riporterebbe il lavoro 5 su M_1, tornando alla soluzione di partenza: la mossa è tabu (attributo $(5, M_1)$ fino all'iterazione 7), e la ricerca sposta il lavoro 5 su M_3.
- All'iterazione 5 i quattro vicini con $C_{\max} = 11$ sono tutti tabu, e nessuno soddisfa il criterio di aspirazione ($C_{\max} < bestC = 11$): la migliore mossa non tabu **peggiora** il makespan a 12. È così che *TS* lascia il plateau.
- All'iterazione 6 il makespan scende a 10, e all'iterazione 8 a 9: $C_{\max} = \mathrm{LB} = 9$, quindi la soluzione $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$ è ottima e la ricerca si ferma: $TS(I) = 9 = z(I)$.

Tre soluzioni visitate da *TS*



- I diagrammi di Gantt della soluzione di partenza, della soluzione dopo la mossa peggiorante dell'iterazione 5 (il lavoro spostato in arancione), e della soluzione ottima raggiunta all'iterazione 8.
- Con *tenure* ≤ 5, *TS* non raggiunge l'ottimo su questa istanza entro 50 iterazioni: la lista tabu è troppo corta per impedire alla ricerca di tornare sul plateau di makespan 11.

Tabu search per il TSP

- Per il TSP con l'intorno 2-opt, gli attributi naturali sono i **lati**: quando una mossa toglie i lati (σ_i, σ_{i+1}) e (σ_j, σ_{j+1}), questi due lati non possono essere rimessi nel giro per *tenure* iterazioni, a meno che la mossa non dia un giro più corto del migliore.
- Tutto il resto è come nello pseudocodice per $P \,||\, C_{\max}$: a ogni iterazione tutte le $\frac{n(n-3)}{2}$ mosse sono valutate in $O(1)$ ciascuna, e si esegue la migliore non tabu.
- Le tabelle della slide successiva riportano il risultato di *TS* sulle istanze medie, per diversi valori della durata tabu:
  - dal giro nearest neighbour (326) per TSP20, con 200 iterazioni;
  - dalla soluzione LPT (73) per PCM15, con 300 iterazioni; la tabella dà anche il numero di iterazioni eseguite (300 salvo che si raggiunga il lower bound).

L'effetto della durata tabu

TSP20: *TS* con 2-opt, 200 iterazioni ($z(I) = 291$)						
tenure	0	1	2	5	10	20
lunghezza migliore	319	291	291	291	291	291

PCM15: *TS* con l'intorno move, al più 300 iterazioni ($z(I) = 69$)										
tenure	0	1	2	3	5	7	10	15	20	30
C_{max} migliore	73	73	73	73	73	72	71	69	70	70
iterazioni	300	300	300	300	300	300	300	231	300	300

La durata tabu va calibrata

- Con *tenure* $= 0$ non c'è memoria, e *TS* cicla: su TSP20 non lascia mai il primo ottimo locale 2-opt.
- Su PCM15 l'intorno move ha un ampio plateau, e *TS* ha bisogno di una lista tabu lunga per attraversarlo;
- ma una lista tabu troppo lunga vieta troppo, e con *tenure* ≥ 20 l'ottimo si perde di nuovo.
- La durata tabu va calibrata, e il suo valore migliore dipende dal problema e dall'istanza.

Riepilogo

Tabu search:

- si sposta sul miglior vicino non tabu, anche se è peggiore: è deterministica
- la lista tabu memorizza attributi delle mosse recenti; il criterio di aspirazione ammette una mossa tabu che migliora la soluzione migliore

Durata tabu:

- troppo corta: la ricerca cicla; troppo lunga: vieta troppo
- sull'istanza di riferimento di $P \,||\, C_{\max}$ *tenure* $= 6$ raggiunge l'ottimo, *tenure* ≤ 5 no
- su PCM15 solo *tenure* $= 15$ raggiunge 69 entro 300 iterazioni

Iterated local search

Introduzione: Iterated local search

- L'iterated local search esce da un ottimo locale *perturbandolo* e scendendo di nuovo
- Vediamo il suo pseudocodice per il TSP, con il kick double bridge, e i ruoli delle sue tre componenti
- Poi seguiamo due iterazioni sull'istanza media del TSP, dove raggiunge l'ottimo

L'idea dell'iterated local search

- L'*iterated local search* (indicata con *ILS*) applica una ricerca locale, e poi ripete:
  - *perturba* l'ottimo locale corrente con una modifica casuale (un *kick*) che la ricerca locale non può annullare facilmente,
  - applica la ricerca locale alla soluzione perturbata,
  - e decide con un *criterio di accettazione* se il nuovo ottimo locale sostituisce quello corrente.

- È una passeggiata fra gli ottimi locali.

Vediamo ora lo pseudocodice di *ILS* per il TSP,
con la ricerca locale 2-opt e il kick double bridge.

Algorithm: Iterated local search per il TSP (1/2)

Input : distanze d, giro di partenza $\sigma[1..n]$, numero di iterazioni *maxIter*

Output : l'ultimo ottimo locale accettato

```
1  function IteratedLocalSearch(d, σ, maxIter)
2      cur := TwoOptLS(σ); lcur := L(cur)        // primo
         ottimo locale
3      for it := 1 to maxIter do
4          nw := DoubleBridge(cur)      // perturbazione
5          nw := TwoOptLS(nw); lnew := L(nw)
             // ricerca locale
6          if lnew < lcur then
7              cur := nw; lcur := lnew       // accettazione
8          end
9      end
10     return cur
11 end
```

Algorithm: Iterated local search per il TSP (2/2)

```
10 function DoubleBridge(σ)
11     p1 := 1 + ⌊Rand() · (n − 3)⌋
12     p2 := p1 + 1 + ⌊Rand() · (n − 2 − p1)⌋
13     p3 := p2 + 1 + ⌊Rand() · (n − 1 − p2)⌋
14     A := σ[1..p1]; B := σ[p1+1..p2]; C := σ[p2+1..p3];
         D := σ[p3+1..n]
15     return (A, C, B, D)
16 end
```

Le tre componenti di *ILS*

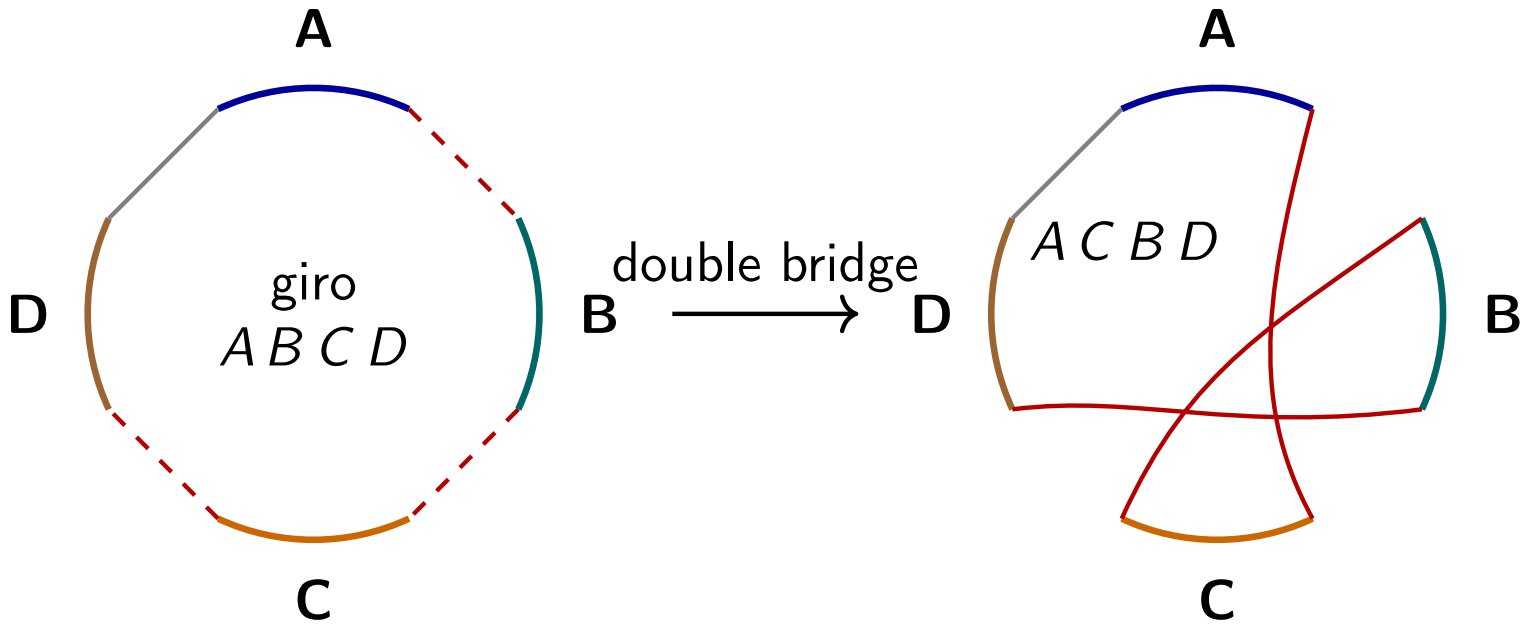
- Lo schema è stato descritto nella sua forma generale da Lourenço, Martin e Stützle (2003), nello *Handbook of Metaheuristics*; per il TSP risale alle «large-step Markov chains» di Martin, Otto e Felten (1991), che hanno introdotto il kick double bridge.

- *ILS* ha tre componenti, e ciascuna ha un ruolo:
  - la **ricerca locale** intensifica: porta ogni soluzione perturbata al fondo della sua valle;
  - la **perturbazione** diversifica: sposta la ricerca in un'altra valle;
  - il **criterio di accettazione** bilancia le due: accettare solo ottimi locali migliori (come nello pseudocodice) intensifica, accettare ogni nuovo ottimo locale diversifica; regole intermedie accettano un ottimo locale peggiore con una certa probabilità, come in *SA*.

L'intensità della perturbazione

- L'**intensità** della perturbazione è il parametro chiave.
- Se è troppo debole, la ricerca locale la annulla e torna allo stesso ottimo locale.
- Se è troppo forte, la soluzione perturbata non ha niente in comune con l'ottimo locale, e *ILS* diventa un multi-start casuale.
- Un buon kick cambia la soluzione in un modo che la ricerca locale *non può* annullare, ma ne conserva la maggior parte.

Il double bridge



- Il **double bridge** taglia il giro in quattro segmenti A, B, C, D e li ricollega come $A\,C\,B\,D$: toglie tre lati (tratteggiati) e ne aggiunge tre nuovi (rossi), mentre il lato fra la fine di D e l'inizio di A si conserva.

Perché 2-opt non annulla il double bridge

- Nessun segmento viene invertito, mentre ogni mossa 2-opt inverte un segmento.
- Quando B e C hanno almeno due città ciascuno, una singola mossa 2-opt non può annullare il kick, e la ricerca locale 2-opt, che accetta solo mosse miglioranti, torna raramente allo stesso ottimo locale.
- Se B e C sono città singole, il kick cambia solo due lati ed è esso stesso una mossa 2-opt: l'esercizio «Iterated local search e VNS 2» usa questo caso.

Esempio: *ILS* sull'istanza media del TSP (1/3)

Su TSP20, la ricerca locale 2-opt dal giro nearest neighbour (lunghezza 326) si ferma nell'ottimo locale

$$\begin{aligned} cur &= (1, 16, 11, 19, 7, 18, 13, 8, 17, 2, \\ &\qquad 4, 10, 20, 14, 12, 5, 6, 9, 3, 15), \\ L(cur) &= 319. \end{aligned}$$

Con $seed = 1$, i tre numeri casuali del primo kick sono 0.0000, 0.1315, 0.7556, quindi

$$\begin{aligned} p_1 &= 1 + \lfloor 0.0000 \cdot 17 \rfloor = 1, \\ p_2 &= 1 + 1 + \lfloor 0.1315 \cdot 17 \rfloor = 4, \\ p_3 &= 4 + 1 + \lfloor 0.7556 \cdot 15 \rfloor = 16. \end{aligned}$$

Esempio: *ILS* sull'istanza media del TSP (2/3)

I segmenti sono

$$A = (1),$$
$$B = (16, 11, 19),$$
$$C = (7, 18, 13, 8, 17, 2, 4, 10, 20, 14, 12, 5),$$
$$D = (6, 9, 3, 15).$$

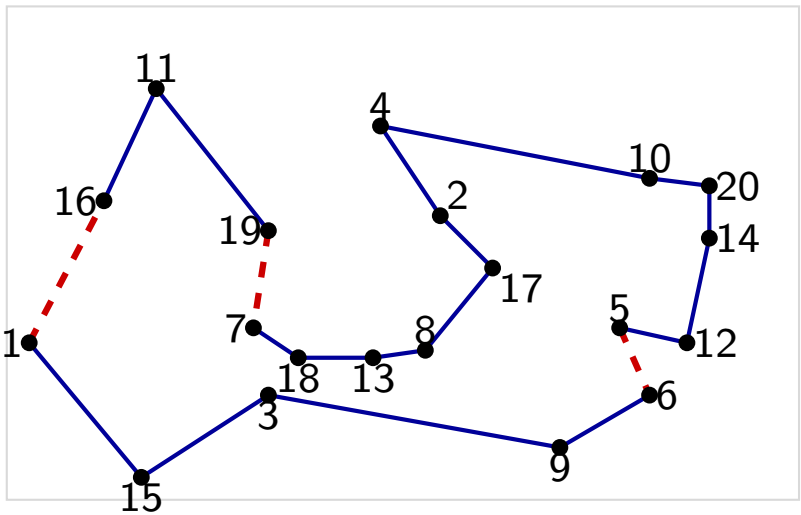
Il kick toglie i lati $(1, 16)$, $(19, 7)$, $(5, 6)$ e aggiunge $(1, 7)$, $(5, 16)$, $(19, 6)$.

Esempio: *ILS* sull'istanza media del TSP (3/3)

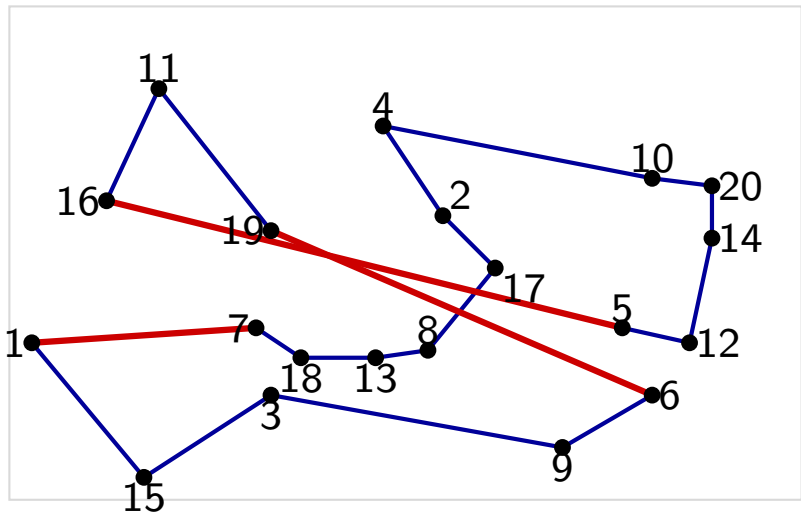
- Il giro perturbato $A\,C\,B\,D$ ha lunghezza 432, molto peggiore di 319.
- La ricerca locale 2-opt lo porta a un nuovo ottimo locale di lunghezza $303 < 319$, che viene accettato.
- Il secondo kick ($p = (8, 14, 16)$) dà, dopo 2-opt, un giro di lunghezza $291 = z(I)$.
- Il terzo dà di nuovo un giro di lunghezza 291, non migliore, e viene rifiutato.

La prima iterazione di *ILS* su TSP20

(a) ottimo locale: $L = 319$



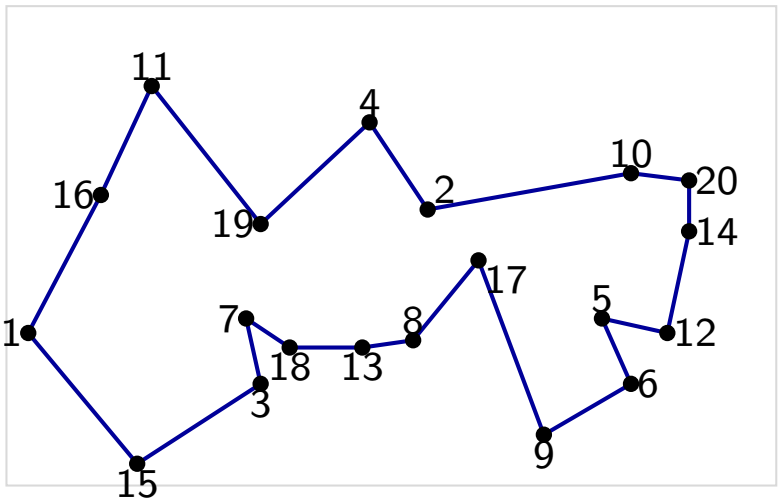
(b) dopo il double bridge: $L = 432$



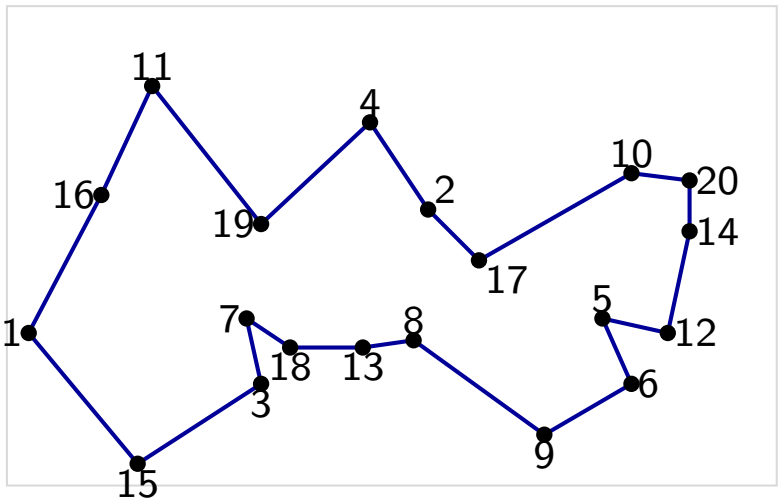
- Il riquadro (a) mostra l'ottimo locale con i tre lati tolti dal kick (tratteggiati), il riquadro (b) il giro perturbato con i tre lati nuovi.

Gli ottimi locali dopo la prima e la seconda iterazione

(c) dopo 2-opt: $L = 303$



(d) dopo la seconda iterazione: $L = 291$



- I riquadri (c) e (d) mostrano gli ottimi locali dopo la prima e la seconda iterazione. Il kick sposta il segmento C prima di B; la ricerca locale 2-opt ripara poi i lati lunghi che esso ha creato, conservando le parti buone del vecchio giro.

ILS su TSP20 con dieci semi

- Con 100 iterazioni e $seed = 1, \ldots, 10$, *ILS* raggiunge l'ottimo 291 in ogni esecuzione.
- L'iterazione in cui l'ottimo viene raggiunto è

  $$2,\ 3,\ 1,\ 7,\ 8,\ 11,\ 15,\ 6,\ 2,\ 3$$

  per i dieci semi.
- Ogni iterazione costa un kick e una ricerca locale 2-opt, che parte da una soluzione vicina a un ottimo locale e richiede poche mosse.

Riepilogo

Iterated local search:

- una passeggiata fra gli ottimi locali: kick, ricerca locale, accettazione
- la ricerca locale intensifica, la perturbazione diversifica, l'accettazione bilancia le due

Il double bridge:

- ricollega $A\,B\,C\,D$ come $A\,C\,B\,D$ senza invertire alcun segmento
- una singola mossa 2-opt non può annullarlo quando B e C hanno almeno due città
- su TSP20 *ILS* raggiunge l'ottimo 291 in ogni esecuzione, entro 15 iterazioni

Riepilogo della prima parte

- Una metaeuristica guida una ricerca locale oltre il primo ottimo locale, bilanciando intensificazione e diversificazione.
- I quattro problemi, i loro intorni, i numeri casuali e le istanze di riferimento e medie sono il quadro di tutto il capitolo.
- Simulated annealing e tabu search escono da una valle passando per soluzioni peggiori, a caso o con memoria.
- L'iterated local search salta in un'altra valle con un kick che la ricerca locale non può annullare.
- La seconda parte presenta VNS, GRASP, large neighbourhood search, algoritmi genetici e memetici, e confronta tutti i metodi sulle istanze medie.