Intelligenza Artificiale

Anno Accademico 2022 - 2023

Tabu Search

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

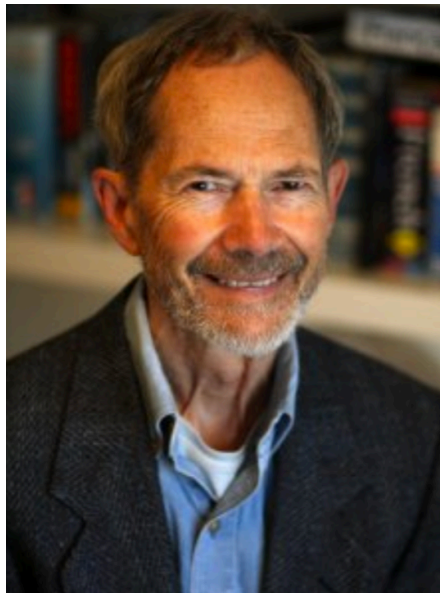
L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Sommario

- Introduzione
- Classificazione di metodi di search
- Confronto tra Tabu Search e Simulated Annealing
- Idea base della Tabu Search
- Esempi di applicazione della Tabu Search (SAT, TSP)

Introduzione

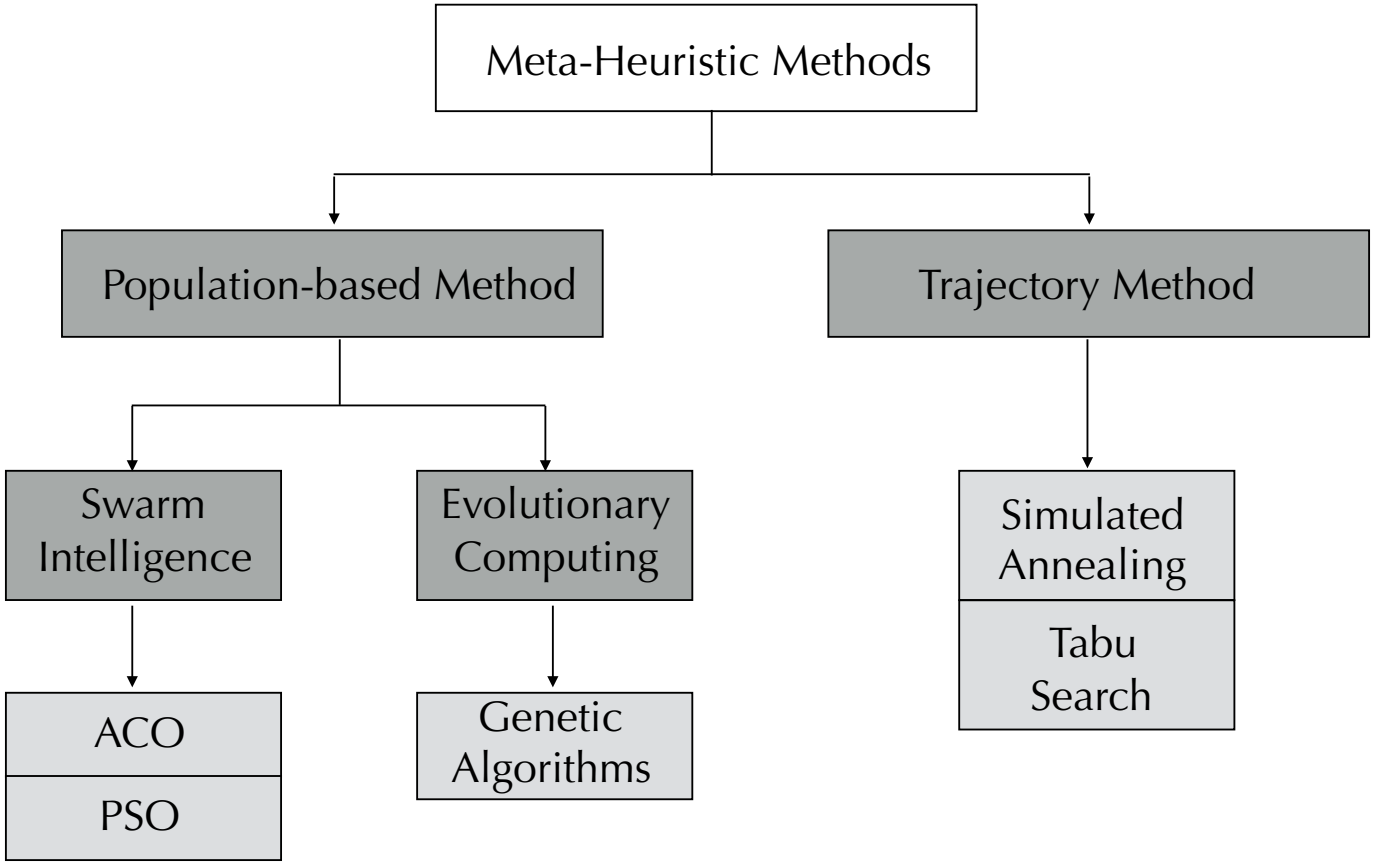
- L'algoritmo **Tabu Search** è stato proposto da Fred W. Glover nel 1989:



- E' uno dei metodi più citati per problemi di ottimizzazione.

Introduzione
Meta-euristiche

- Tassonomia per le meta-euristiche:



Introduzione
Tabu Search e SA

- **Tabu Search** è molto simile al **Simulated Annealing** relativamente alla struttura dell'algoritmo. Ad alto livello di astrazione i due algoritmi possono essere descritti come segue:

```
function simulated_annealing(problem)
current ← un certo stato iniziale in S
Inizializza best
repeat
   current ← improve?(current, T)
   update(T)
   update(best)
until criterio-terminazione
return STATE(best)
end
```

```
function tabu_search(problem)
current ← un certo stato iniziale in S
Inizializza best
repeat
   current ← improve?(current, H)
   update(H)
   update(best)
until criterio-terminazione
return STATE(best)
end
```

Introduzione
Tabu Search e SA

- Come sappiamo, nell'algoritmo **Simulated Annealing** la funzione `improve?(current,T)` restituisce uno stato "accettabile" dal neighborhood di `current`, che non è necessariamente migliore di `current`, la cui accettazione dipende dal parametro `T`.

- Analogamente, in **Tabu Search** `improve?(current,H)` restituisce uno stato "accettabile" dal neighborhood di `current`.

- Tale stato non è necessariamente migliore di `current`, ma la sua accettazione è basata sulla storia `H` della ricerca.

Tabu Search
Idea principale

- L'idea principale alla base della **Tabu Search** è molto semplice:
  - Una "memoria" forza l'algoritmo ad esplorare nuove aree dello spazio di ricerca. Memorizziamo gli stati che sono stati esaminati di recente e questi diventano stati **tabu** (proibiti) da evitare nel prendere decisioni sulla scelta dello stato successivo.
  - Accettazione **deterministica** di soluzioni "peggiori" per sfuggire da ottimi locali, ossia nel caso in cui gli stati del neighborhood dello stato corrente siano tutti peggiori di tale stato.

Tabu Search
Idea principale

- Vedremo che **Tabu Search** di solito fa mosse verso stati "peggiori" solo quando è bloccato in un ottimo locale, mentre **Simulated Annealing** può fare mosse verso stati "peggiori" in qualsiasi momento.

- Da notare che **Tabu Search**, a differenza del **Simulated Annealing**, è sostanzialmente deterministico, anche se è possibile inserire elementi probabilistici in esso.

- Per descrivere il funzionamento dell'algoritmo di **Tabu Search** vediamo come possa essere applicato ad un problema specifico, ad esempio al problema **SAT** (Boolean Satisfiability Problem).

Esempio di Applicazione
Descrizione problema SAT

- Data una formula logica $\mathcal{F}(\mathbf{x})$, l'obiettivo è quello di assegnare i valori di verità alle sue variabili booleane in modo tale che il valore di verità della formula $\mathcal{F}$ sia **TRUE**.

- Ad esempio, data la seguente funzione:

$$\mathcal{F}(\mathbf{x}) = (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee x_3)$$

dobbiamo individuare l'assegnazione di valori di verità alle variabili x_1, x_2 e x_3 in modo tale che la $\mathcal{F}$ sia **TRUE**.

Esempio di Applicazione
Spazio degli Stati

- Il generico **stato** per un problema SAT a `n` variabili è costituito da una lista di `n` valori di verità (True/False oppure 0/1):

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)$$

- La dimensione dello spazio degli stati è uguale a 2^n.

Esempio di Applicazione
Funzione di Valutazione

- Occorre ora definire una **funzione di valutazione**, che ci consenta di confrontare tra loro stati distinti.

- Considerando il nostro esempio con tre variabili, non è facile confrontare i due stati seguenti, che danno luogo allo stesso valore di verità (False) per la funzione $\mathcal{F}$:

$$\mathbf{x}_a = (1, 0, 0) \qquad \mathbf{x}_b = (0, 0, 0)$$

Esempio di Applicazione
Funzione di Valutazione

- Una possibilità è quella di definire una funzione di valutazione $eval_{\mathcal{F}}$ come il rapporto tra il numero di congiunzioni soddisfatte e il numero totale delle congiunzioni. Nel nostro caso:

$$eval_{\mathcal{F}}(\mathbf{x}_a) = 0{,}333 \qquad eval_{\mathcal{F}}(\mathbf{x}_b) = 0{,}666$$

- Un'altra possibilità consiste nel calcolare una somma pesata del numero di clausole soddisfatte, dove i pesi dipendono dal numero di variabili presenti nella clausola.

Esempio di Applicazione
Risoluzione problema SAT

- Supponiamo dunque di dover risolvere un problema SAT per una formula logica $\mathcal{F}(\mathbf{x})$ con $\mathbf{x} = (x_1, x_2, \ldots, x_8)$, ossia con un numero di variabili $n = 8$.

- Utilizziamo una delle funzioni di valutazione citate, ad esempio la seconda. Tale funzione deve essere massimizzata.

- Per quanto riguarda gli operatori da applicare ad uno stato per la determinazione del suo **neighborhood**, scegliamo di poter effettuare l'inversione di uno dei bit dello stato.

Esempio di Applicazione
Risoluzione problema SAT

Ricapitolando:

- **Stati**: una qualsiasi stringa binaria lunga **8 bit**. Cardinalità dello spazio di ricerca S: $\mathbf{2^8}$
- **Operatori**: inversione (flip) di uno degli 8 bit di uno stato. Cardinalità del neighborhood: **8**.
- **Funzione di valutazione**: somma pesata del numero di clausole soddisfatte.

Esempio di Applicazione
Risoluzione problema SAT

- Supponiamo di aver generato il seguente stato iniziale:

$$\mathbf{x} = (0, 1, 1, 1, 0, 0, 0, 1)$$

  e che la sua valutazione $eval_{\mathcal{F}}(\mathbf{x})$ sia uguale a 27.

- A questo punto dobbiamo esaminare il neighborhood di **x**, che consiste in 8 stati ciascuno dei quali ottenuto invertendo un singolo bit di **x**.

- Valutiamo questi 8 stati e scegliamo il migliore. Supponiamo che sia quello che deriva dalla inversione del terzo bit, e che la sua valutazione sia di 31. Fino a questo stadio della ricerca l'algoritmo si è comportato come l'hill climbing.

Esempio di Applicazione
Risoluzione problema SAT

- A questo punto entra in gioco la "memoria" a cui abbiamo accennato in precedenza.

- In generale, nella Tabu Search si hanno due possibili approcci:

  1. Memorizzazione degli **stati** visitati: oneroso dal punto di vista computazionale per state spaces molto grandi.

  2. Memorizzazione dei **cambiamenti** effettuati di recente su alcune **features** (**feature-based**). Nel nostro esempio, possiamo memorizzare le **mosse** (flip), ossia le **azioni** effettuate per passare dallo stato corrente al successore prescelto.

Esempio di Applicazione
Risoluzione problema SAT

- Dobbiamo memorizzare sia la mossa che abbiamo fatta per passare dallo stato corrente al next, sia il "tempo" in cui ciò è avvenuto.
- Nel nostro contesto il "tempo" consiste in quale iterazione la mossa è stata fatta.
- Utilizziamo per questo un vettore **M** di lunghezza pari a 8.

Esempio di Applicazione
Risoluzione problema SAT

- Questo vettore **M** è inizializzato a zero. Dopo la prima mossa di cui sopra facciamo la seguente assegnazione:

$$\mathbf{M}(3) = j \text{ (con } j \neq 0)$$

- Il valore di "j" potrebbe essere interpretato come la più recente iterazione in cui il terzo bit è stato invertito.

- E però utile per i nostri scopi cambiare tale interpretazione in modo tale da modellare un ulteriore aspetto della memoria.

Esempio di Applicazione
Risoluzione problema SAT

- Noi vogliamo che tale informazione sulla mossa fatta stia in memoria non per sempre ma, ad esempio, per 5 iterazioni.

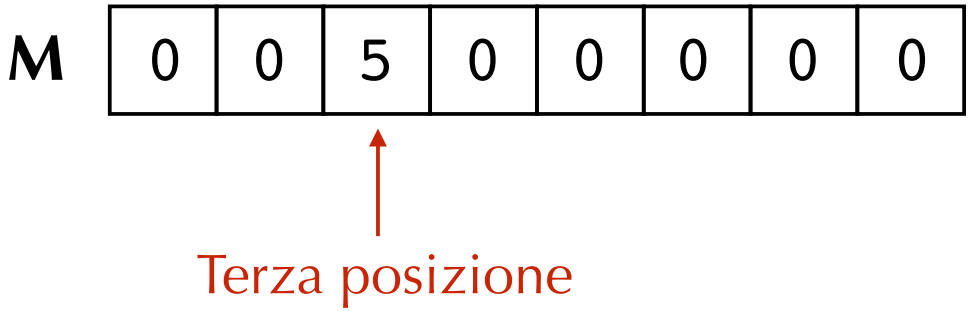
- La nuova interpretazione di

$$\mathbf{M}(3) = j \text{ (con } j \neq 0)$$

è la seguente: il terzo bit è stato invertito 5 - j iterazioni fa.

Esempio di Applicazione
Risoluzione problema SAT

- Con questa interpretazione il contenuto di **M** subito dopo aver invertito il terzo bit di **x** è il seguente:

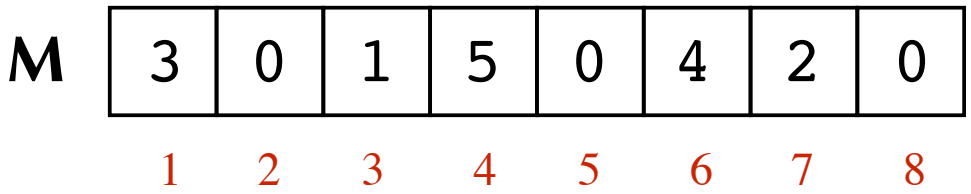


- Il valore 5 può essere interpretato così: per le prossime 5 iterazioni il bit in terza posizione nello stato corrente non è disponibile per effettuare una mossa (ossia è **tabu**). Tale valore è detto **tabu tenure**.

- Ad ogni iterazione successiva **M**(3) verrà decrementato di 1.

Esempio di Applicazione
Risoluzione problema SAT

- Supponiamo che dopo ulteriori 4 iterazioni la memoria **M** (che possiamo chiamare **tabu list**) abbia il seguente contenuto:



- I bit 2, 5 e 8 sono disponibili per eventuali mosse. Il bit 1 non è disponibile per le prossime 3 iterazioni. Il bit 3 non è disponibile ma solo per la prossima iterazione. Il bit 4 (che è appena stato usato per la mossa scelta) non è disponibile per le prossime 5 iterazioni.

Esempio di Applicazione
Risoluzione problema SAT

- E' immediato notare che lo stato corrente a questo punto è il seguente:

$$\mathbf{x} = (1, 1, 0, 0, 0, 1, 1, 1)$$

- Supponiamo che la valutazione di $\mathbf{x}$ sia di $\mathbf{33}$.

- Analizziamo attentamente i suoi 8 successori:

$$\mathbf{x}_1 = (0, 1, 0, 0, 0, 1, 1, 1)$$
$$\mathbf{x}_2 = (1, 0, 0, 0, 0, 1, 1, 1)$$
$$\mathbf{x}_3 = (1, 1, 1, 0, 0, 1, 1, 1)$$
$$\mathbf{x}_4 = (1, 1, 0, 1, 0, 1, 1, 1)$$
$$\mathbf{x}_5 = (1, 1, 0, 0, 1, 1, 1, 1)$$
$$\mathbf{x}_6 = (1, 1, 0, 0, 0, 0, 1, 1)$$
$$\mathbf{x}_7 = (1, 1, 0, 0, 0, 1, 0, 1)$$
$$\mathbf{x}_8 = (1, 1, 0, 0, 0, 1, 1, 0)$$

Esempio di Applicazione
Risoluzione problema SAT

- E' immediato notare che lo stato corrente a questo punto è il seguente:

$$\mathbf{x} = (1, 1, 0, 0, 0, 1, 1, 1)$$

- Supponiamo che la valutazione di **x** sia di **33**.

- Analizziamo attentamente i suoi 8 successori:

$\mathbf{x}_1 = (0, 1, 0, 0, 0, 1, 1, 1)$ **T**
$\mathbf{x}_2 = (1, 0, 0, 0, 0, 1, 1, 1)$
$\mathbf{x}_3 = (1, 1, 1, 0, 0, 1, 1, 1)$ **T**
$\mathbf{x}_4 = (1, 1, 0, 1, 0, 1, 1, 1)$ **T**
$\mathbf{x}_5 = (1, 1, 0, 0, 1, 1, 1, 1)$
$\mathbf{x}_6 = (1, 1, 0, 0, 0, 0, 1, 1)$ **T**
$\mathbf{x}_7 = (1, 1, 0, 0, 0, 1, 0, 1)$ **T**
$\mathbf{x}_8 = (1, 1, 0, 0, 0, 1, 1, 0)$

Esempio di Applicazione
Risoluzione problema SAT

- Considerando la **tabu list** possiamo dunque dire che lo stato successore di **x** dovrà essere scelto dall'insieme degli stati $\mathbf{x}_2$, $\mathbf{x}_5$, e $\mathbf{x}_8$.
- Supponiamo che lo stato con la migliore valutazione dei tre sia $\mathbf{x}_5$, e che valga 32. L'algoritmo sceglierà $\mathbf{x}_5$, anche se la sua valutazione è minore di quella di **x** (che è 33).
- Dopo questa iterazione la situazione di **M** sarà la seguente, in cui la quinta posizione è stata impostata a 5 e tutti i valori diversi da zero sono stati decrementati di uno:

M	2	0	0	4	5	3	1	0
	1	2	3	4	5	6	7	8

Tabu Search
Prima Versione dell'Algoritmo

- Le successive iterazioni sono eseguite nello stesso modo. Ad ogni stadio si considera il neighborhood dello stato corrente da cui si eliminano le mosse tabu:

```
function tabu_search(problem)
Definizione tabu tenure
Inizializza la tabu list M
current ← un certo stato iniziale in S
Inizializza best
repeat
  Calcolo del neighborhood N di current
  current ← migliore stato ∈ non_tabu_N
  update(M)
  best ← migliore(current, best)
until condizione-terminazione
return STATE(best)
```

Tabu Search
Prima Versione dell'Algoritmo

- Per quanto riguarda la condizione di terminazione ci sono varie possibilità, come ad esempio le seguenti:
  - Numero massimo di iterazioni raggiunto.
  - Nessun significativo miglioramento nelle ultime iterazioni.
  - Se è possibile stabilire che il **best** corrente è la soluzione ottima cercata.

Tabu Search
Aspiration Criterion

- L'approccio appena illustrato potrebbe però essere un po' troppo restrittivo.
- Potrebbe ad esempio accadere che uno degli stati tabu del neighborhood abbia una valutazione particolarmente alta, ad esempio migliore del **best** corrente e di tutti gli stati non-tabu.
- In tali casi eccezionali potrebbe allora essere conveniente derogare dalla regola tabu, ed accettare come stato successore lo stato tabu che ha la suddetta alta valutazione (**override** della classificazione tabu).
- Tale criterio è noto come **aspiration criterion**.

Tabu Search
Versione con l'Aspiration Criterion

- L'algoritmo precedente può essere aggiornato inserendo l'aspiration criterion come segue, dove A è l'insieme degli stati tabu che soddisfano il criterio:

```
function tabu_search(problem)
Definizione tabu tenure
Inizializza la tabu list M
current ← un certo stato iniziale in S
Inizializza best
repeat
  Calcolo del neighborhood N di current
  Calcolo di A
  current ← migliore stato ∈ {non_tabu_N U A}
  update(M)
  best ← migliore(current, best)
until condizione-terminazione
return STATE(best)
```

Tabu Search
Altri Possibili Criteri

- E' possibile cambiare la precedente procedura di selezione **deterministica** dello stato successivo usando invece un approccio **probabilistico**, con il quale gli stati del neighborhood con valutazioni migliori abbiano una maggiore probabilità di essere selezionati.

- E' possibile inoltre cambiare il **tabu tenure** durante la search: a volte potrebbe convenire ricordare "di più" e altre volte ricordare "meno" (ad esempio, quando l'algoritmo scala un'area promettente dello spazio di ricerca).

- E' possibile anche collegare il **tabu tenure** alla dimensione del problema.

Tabu Search
Uso della frequency-based memory

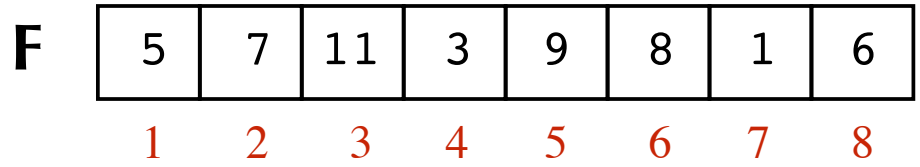
- C'è anche un'altra opzione molto interessante. La memoria **M** discussa finora può essere definita come **recency-based memory**. A tale struttura possiamo aggiungerne un'altra, che possiamo chiamare **frequency-based memory**.

- Riferendoci al nostro esempio del problema SAT, possiamo definire un vettore **F** di 8 elementi tale per cui **F**(i) rappresenti il numero di volte in cui il bit i-esimo è stato invertito all'interno di una certo **orizzonte**, ossia nelle ultime h iterazioni.

- Normalmente il parametro h è impostato ad un valore piuttosto grande. In tal modo il vettore **F** gioca in buona sostanza il ruolo di **long-term memory**.

Tabu Search
Uso della frequency-based memory

- Tornando al nostro esempio per il problema SAT, dopo un certo numero di iterazioni (e.g., 100) con $h = 50$, la long-term memory **F** potrebbe avere i seguenti valori:



- La memoria **F** potrebbe essere utile per **diversificare** la ricerca.
- Tale memoria fornisce infatti informazioni su quali flip sono stati sottorappresentati (cioè meno frequenti) o non rappresentati affatto. Possiamo dunque cercare di diversificare la ricerca esplorando queste possibilità.

Tabu Search
Uso della frequency-based memory

- L'uso di tale memoria nella tabu search è solitamente limitato ad alcune circostanze speciali.

- Ad esempio, potremmo incontrare una situazione in cui tutte le mosse non-tabu portino ad uno stato peggiore di quello corrente.

- In questo caso ci si può avvalere della long-term memory. Il tipico approccio consiste nel considerare meno attrattive le mosse più frequenti.

- In tal caso la **valutazione** di uno stato raggiunto con una certa mossa viene **decrementata** di una certa **penalità** dipendente dalla frequenza della mossa.

Tabu Search
Uso della frequency-based memory

- Per illustrare questo punto, supponiamo di trovarci in uno stato corrente **x** la cui valutazione sia di 35 e che il **best** corrente valga 37.

- Supponiamo che le mosse non-tabu siano relative ai bit 2, 3 e 7, che che esse diano luogo rispettivamente a stati con le valutazioni 30, 33 e 31.

- Supponiamo ancora che nessuna delle mosse tabu superi **best**, ossia 37, in modo da non poter applicare l'**aspiration criterion**.

Tabu Search
Uso della frequency-based memory

- In tale circostanza ci possiamo avvalere della frequency-based memory vista prima:

F	5	7	11	3	9	8	1	6
	1	2	3	4	5	6	7	8

- La funzione di valutazione per un nuovo stato **x′** in tal caso può essere rimodulata come segue:

$$eval_{\mathcal{F}}(\mathbf{x}') - penalty(\mathbf{x}')$$

dove *eval* restituisce la valutazione originaria per i tre possibili successori (i.e., 30, 33 e 31), mentre, ad esempio

$$penalty(\mathbf{x}') = 0.7\ \mathbf{F}(i)$$

Tabu Search
Uso della frequency-based memory

- Le valutazioni complessive per i tre possibili stati successori, includendo le *penalty*, sarebbero le seguenti:

$$eval_{\mathcal{F}}(\mathbf{x}_2) - penalty(\mathbf{x}_2) = 30 - 0.7 \times 7 = 25.1$$
$$eval_{\mathcal{F}}(\mathbf{x}_3) - penalty(\mathbf{x}_3) = 33 - 0.7 \times 11 = 25.3$$
$$eval_{\mathcal{F}}(\mathbf{x}_7) - penalty(\mathbf{x}_7) = 31 - 0.7 \times 1 = 30.3$$

che porterebbero a scegliere lo stato $\mathbf{x}_7$ come successore di $\mathbf{x}$.

Esempio di Applicazione
Problema del Commesso Viaggiatore

- Vediamo un altro esempio di applicazione della **Tabu Search**, questa volta applicata al **Problema del Commesso Viaggiatore** (**TSP**: Traveling Salesman problem).

- Il problema è il seguente: Il commesso viaggiatore deve visitare ogni città del suo territorio esattamente una volta e poi tornare al punto di partenza percorrendo la distanza più breve.

- Possiamo rappresentare il problema avvalendoci di un **grafo completo pesato**, in cui i nodi sono le città e gli archi i collegamenti pesati, che rappresentano la distanza tra le due città collegate.

Esempio di Applicazione
Problema del Commesso Viaggiatore

- Per questo problema (*ciclo hamiltoniano di lunghezza minima*) possiamo rappresentare gli **stati** come una lista ordinata di elementi, ciascuno dei quali rappresenta una città. Ad esempio, nel caso di 8 città:

$$\mathbf{x} = (2, 4, 7, 5, 1, 8, 3, 6)$$

- Dobbiamo ora definire il tipo di **operatori**, ossia le possibili **mosse** da poter utilizzare per individuare il neighborhood di un certo stato.

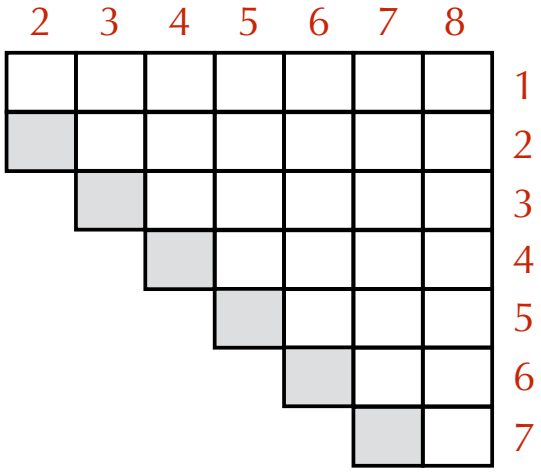
Esempio di Applicazione
Problema del Commesso Viaggiatore

- Abbiamo varie possibilità di scelta per la definizione del neighborhood di uno stato.
- Per il nostro esempio definiamo come mossa uno scambio di posizione (**swap**) tra due città.
- Con tale definizione di mossa, per ogni stato **x** abbiamo 28 possibili mosse, ossia 28 possibili successori di **x** dati dal numero di combinazioni di classe 2 di 8 oggetti:

$$C_{8,2} = \binom{8}{2} = \frac{8 \cdot 7}{2!} = 28$$

Esempio di Applicazione
Problema del Commesso Viaggiatore

- Sia per la **recency-based memory** **M** sia per la **frequency-based memory** **F** possiamo usare la struttura che segue, dove il dato relativo allo swap tra le città i e j è memorizzato nell'elemento di riga i e di colonna j:



- Interpretiamo i e j come città, e non come loro posizioni nello stato **x**, anche se quest'ultima è un'altra possibilità da poter considerare.

Esempio di Applicazione
Problema del Commesso Viaggiatore

- Supponiamo di aver inizializzato a zero entrambe le memorie, che siano state effettuate 500 iterazioni con un **best** che vale 171, e che lo stato corrente sia:

$$\mathbf{x} = (7, 3, 5, 6, 1, 2, 4, 8)$$

corrispondente ad un percorso lungo 173.

- Supponiamo anche che lo stato della tabu list **M** e della frequency-based memory **F** sia quello rappresentato nelle due figure che seguono.

Esempio di Applicazione
Problema del Commesso Viaggiatore

- Le memorie a breve termine **M** e a lungo termine **F** dopo 500 iterazioni:

2	3	4	5	6	7	8	
0	0	1	0	0	0	0	1
	0	0	0	5	0	0	2
		0	0	0	4	0	3
			3	0	0	0	4
				0	0	2	5
					0	0	6
						0	7

Recency-based memory **M**

2	3	4	5	6	7	8	
0	2	2	3	0	1	1	1
	2	1	3	1	1	0	2
		2	3	2	4	0	3
			1	1	2	1	4
				4	2	1	5
					3	1	6
						6	7

Frequency-based memory **F**

Esempio di Applicazione
Problema del Commesso Viaggiatore

- E' facile interpretare i numeri presenti in tali memorie.
- Ad esempio, nel caso cui il tabu tenure sia di 5 come nel SAT, il valore 5 nell'elemento **M**(2,6) indica che l'ultimo swap è stato fatto tra le città 2 e 6. Lo stato precedente era dunque il seguente:

$$\mathbf{x} = (7, 3, 5, 2, 1, 6, 4, 8)$$

- Anche la memoria **F** fornisce utili statistiche. Lo swap tra le città 7 e 8 è stato quello più frequente all'interno dell'orizzonte h (supponiamo sempre che $h = 50$). Ci sono coppie di città (e.g., 3 e 8) che invece non sono state mai scambiate nelle ultime 50 iterazioni.

Esempio di Applicazione
Problema del Commesso Viaggiatore

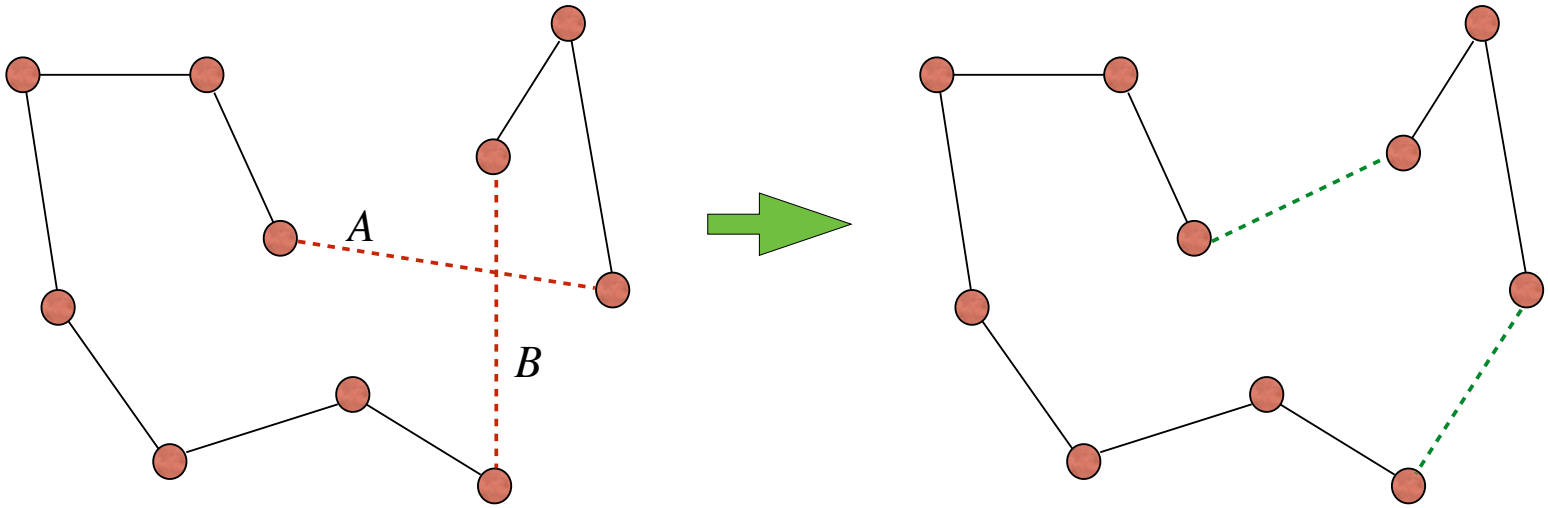
- In questo esempio di applicazione sul TSP abbiamo definito il neighborhood specificando come possibili mosse lo scambio di due città.

- Vari ricercatori hanno scelto invece neighborhoods più estesi.

- Ad esempio, è possibile considerare neighborhoods ottenuti mediante un altro tipo di mossa (*2-interchange move,* approccio **2-opt**) che consiste nella eliminazione di due edges non adiacenti dal tour (ossia dallo stato) corrente e nell'aggiunta di altri due edges per ottenere un nuovo stato possibile.

Esempio di Applicazione
Problema del Commesso Viaggiatore

- In figura è rappresentato un esempio di **2-opt**:



Sintesi degli Argomenti Trattati nella Lezione

- Gli algoritmi **Simulated Annealing** e **Tabu Search** sono stati entrambi ideati allo scopo di poter sfuggire da ottimi locali. Tuttavia, differiscono nei metodi utilizzati per raggiungere questo obiettivo.

- **Tabu Search** di solito fa mosse verso stati "peggiori" solo quando è bloccato in un ottimo locale, mentre **Simulated Annealing** può fare mosse verso stati "peggiori" in qualsiasi momento. Inoltre, **Simulated Annealing** è un algoritmo stocastico, mentre **Tabu Search** è deterministico.

- L'algoritmo **Tabu Search** si avvale di una "memoria" per ricordare gli stati già visitati, che diventano **tabu** per un certo numero di iterazioni (**tabu tenure**). Un approccio un po' diverso (**feature-based**), che abbiamo visto negli esempi, consiste nel ricordare le **mosse** (ossia le **azioni**) che hanno portato da uno stato corrente allo stato scelto come successivo (next), mosse che vengono memorizzate in una **tabu list** (**memoria a breve termine**) e che quindi non possono essere utilizzate per un certo numero di iterazioni.

- Nella **Tabu Search** ci si può anche avvalere dell'**aspiration criterion**, che consiste nel poter scegliere come stato successivo (next) anche uno stato tabu se la **valutazione** di tale stato è particolarmente buona.

- Un altro criterio da poter utilizzare è quello che si avvale di una **frequency-based memory**, che è una **memoria a lungo termine** che registra per ogni possibile mossa il numero di volte in cui è stata applicata all'interno di un dato **orizzonte**, ossia nelle ultime **h** iterazioni. Il criterio favorisce le mosse meno frequenti (**diversification**).

- Nella **Tabu Search**, così come nel **Simulated Annealing**, occorre decidere come scegliere vari **parametri** dell'algoritmo (e.g., tabu tenure, orizzonte h, ecc.) in modo da ottenere le migliori prestazioni. Questo è un problema pervasivo che accompagna la stragrande maggioranza degli algoritmi concepiti per cercare di sfuggire da ottimi locali. E' insomma una caratteristica distintiva di tali sofisticati metodi che dobbiamo gestire. Del resto, più sofisticato è il metodo, più è necessario usare il nostro giudizio sul come tale metodo debba essere utilizzato.

Riferimenti

Glover, F. “Tabu Search - Part I”, *ORSA Journal on Computing*, Vol. 1, No. 3, 1989, pp. 190-206.

Glover, F. “Tabu Search - Part II”, *ORSA Journal on Computing*, Vol. 2, No. 1, 1990, pp. 4-32.

Michalewicz, Z. E Fogel, D.B. *How to Solve It: Modern Heuristics*, Springer, 2010.

Luke, S. *Essentials of Metaheuristics*, Second Edition, 2013.