UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laurea triennale in Informatica

Fondamenti di Intelligenza Artificiale

Lezione 6 - Algoritmi di ricerca locale (I)

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laurea triennale in Informatica
Anno accademico 2020/2021

Link al PadLet di oggi… ma serve un* segretari* che mi aiuti!

Algoritmi di Ricerca Locale

Nelle lezioni precedenti…

Agenti Risolutori di Problemi

Esaminare le azioni future: che significa?

Ipotizziamo che l'ambiente sia **osservabile**, pertanto l'agente conosce *sempre* lo stato corrente del mondo. Nell'esempio precedente, questo significherebbe che ogni città abbia un cartello che indichi la sua presenza ai guidatori in arrivo.

Ipotizziamo che l'ambiente sia **discreto**, pertanto in qualsiasi stato esiste un numero finito di azioni tra cui scegliere.

Ipotizziamo che l'ambiente sia **noto**, pertanto l'agente saprà sempre quali stati saranno raggiunti da ciascuna azione. Nei problemi di navigazione, questo requisito è soddisfatto a patto che si abbia a disposizione una mappa accurata.

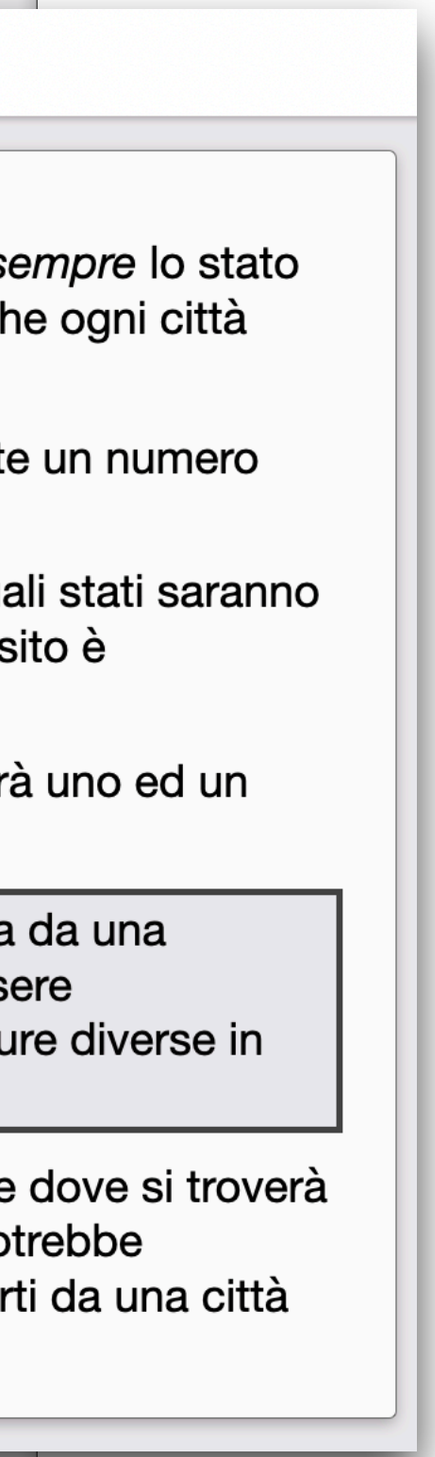
Ipotizziamo che l'ambiente sia **deterministico**, pertanto ogni azione avrà uno ed un solo risultato.

Sotto queste ipotesi, la soluzione di qualsiasi problema è rappresentata da una **sequenza fissata di azioni**. In altri termini, una soluzione potrebbe essere implementata come una strategia ramificata che raccomandi azioni future diverse in base alle percezioni giunte.

In una situazione del genere, l'agente in esempio saprebbe esattamente dove si troverà dopo la prima azione e che cosa percepirà, e così via. Così facendo, potrebbe identificare una soluzione specificata da una sequenza di azioni che porti da una città all'altra fino a Bucarest.

Algoritmi di Ricerca Locale

Verso la ricerca locale



In questa parte del corso, inizieremo a rilassare queste ipotesi in maniera da avvicinarci ad una formulazione più reale dei problemi di ricerca.

Gli algoritmi che abbiamo studiato fino adesso sono progettati per esplorare sistematicamente lo spazio degli stati: per farlo, tengono in memoria uno o più cammini e registrano quali alternative sono state esplorate in ogni punto del cammino.

Quando viene raggiunto uno stato obiettivo, **il cammino** verso quello stato costituisce una soluzione del problema.

Tuttavia, in molti casi il cammino che porta alla soluzione è *irrilevante*; ad esempio, nel problema delle N regine ciò che conta è la disposizione finale delle regine sulla scacchiera, non l'ordine con cui sono state aggiunte.

Questo vale per molti problemi reali: la configurazione degli spazi nelle fabbriche, la programmazione automatica degli orari di lavoro, la gestione di portafogli azionari, l'ottimizzazione di reti di telecomunicazione, e altri.

Nel contesto di questa classe di problemi, possiamo considerare algoritmi diversi che ignorano il cammino che porta alla soluzione.

Algoritmi di Ricerca Locale

Algoritmi di ricerca locale

Quindi, in molti problemi di ottimizzazione lo **stato obiettivo è esso stesso la soluzione** al problema, indipendentemente da come ci si è arrivati.

Lo spazio degli stati è dato dall'insieme delle configurazioni "complete", ovvero quelle che portano alla risoluzione del problema e, quindi, ad uno stato obiettivo.

In tali casi, si possono utilizzare **algoritmi di miglioramento iterativo**, i quali mantengono in memoria solo lo stato corrente e tentano di migliorarlo.

Algoritmi di ricerca "tradizionali"	*Algoritmi di ricerca locale*
Considerano interi cammini dallo stato iniziale a quello obiettivo	Considerano lo stato corrente con l'obiettivo di migliorarlo
Hanno una complessità temporale e spaziale generalmente esponenziale	Usano poca memoria, molto spesso avendo una complessità costante
Non possono essere applicati in problemi con spazio degli stati grandi/infiniti	Possono trovare soluzioni ragionevoli (sub-ottimali) a problemi complessi

Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - La funzione obiettivo

Gli algoritmi di ricerca locale **non hanno un test obiettivo**, ma affidano le loro azioni ad una **funzione obiettivo** che indica quanto la ricerca sta migliorando nelle iterazioni.

Questo li rende adatti alla risoluzione di classici problemi di ottimizzazione, come ad esempio la disposizione dei tavoli in un ristorante. In questi casi, infatti, non esiste un vero e proprio test obiettivo.

La scelta della funzione obiettivo è fondamentale per l'efficienza del sistema e definisce *l'insieme delle soluzioni che possono essere raggiunte da una soluzione s in un singolo passo di ricerca dell'algoritmo*.

Tipicamente, la funzione obiettivo è definita attraverso le possibili mosse che l'algoritmo può effettuare. È da notare che la ricerca locale si basa sull'esplorazione iterativa delle "soluzioni vicine" che possono migliorare quella corrente mediante modifiche locali.

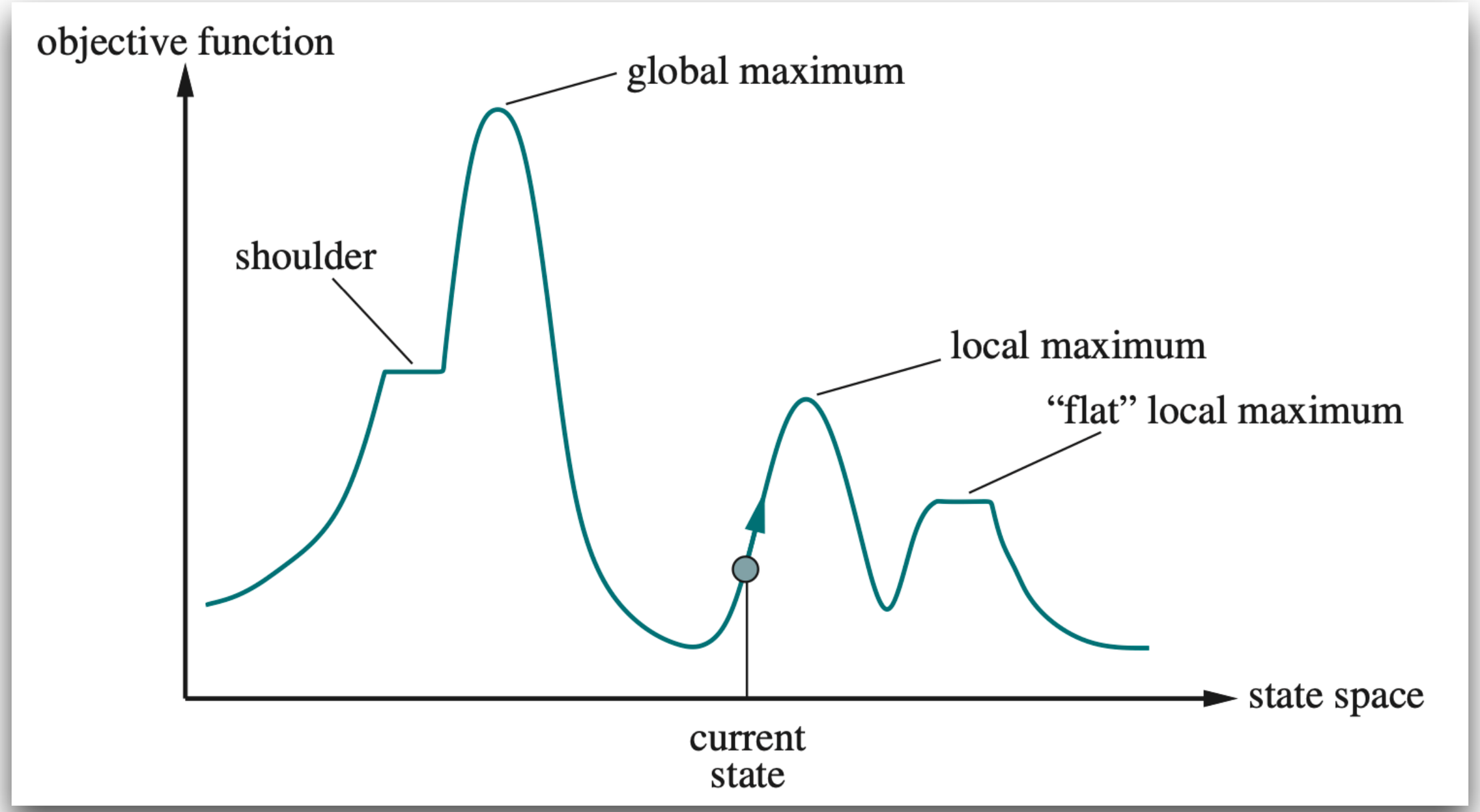
Struttura dei vicini (neighborhood): Una struttura dei vicini è una funzione F che assegna a ogni soluzione s dell'insieme di soluzioni S un insieme di soluzioni N(s) sottoinsieme di S.

È importante notare che la soluzione trovata da un algoritmo di ricerca locale non è detto che sia **globalmente ottima**, ma può essere **ottima rispetto ai cambiamenti locali**. In altri termini, la soluzione può essere la migliore possibile date le circostanze.

Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

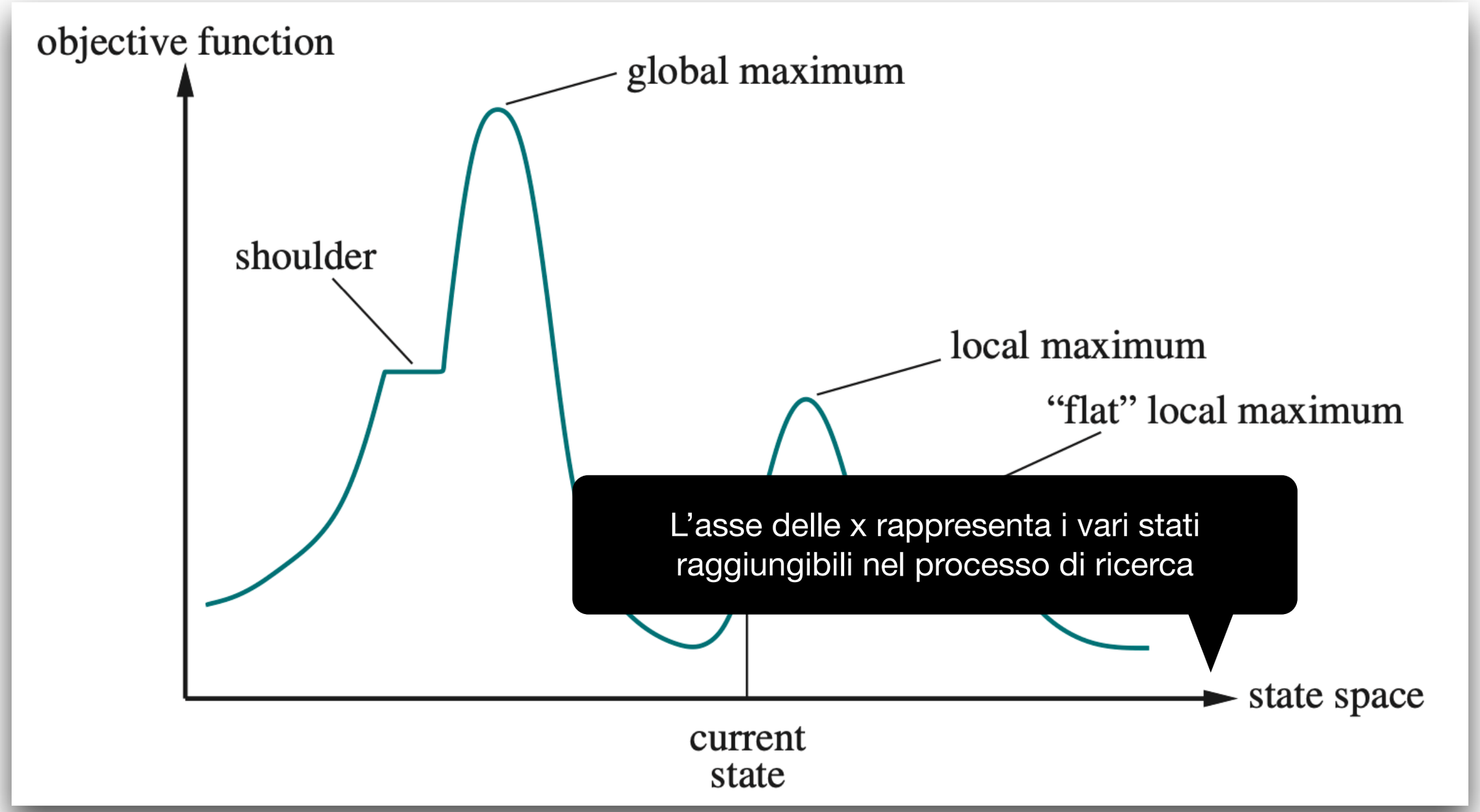
Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

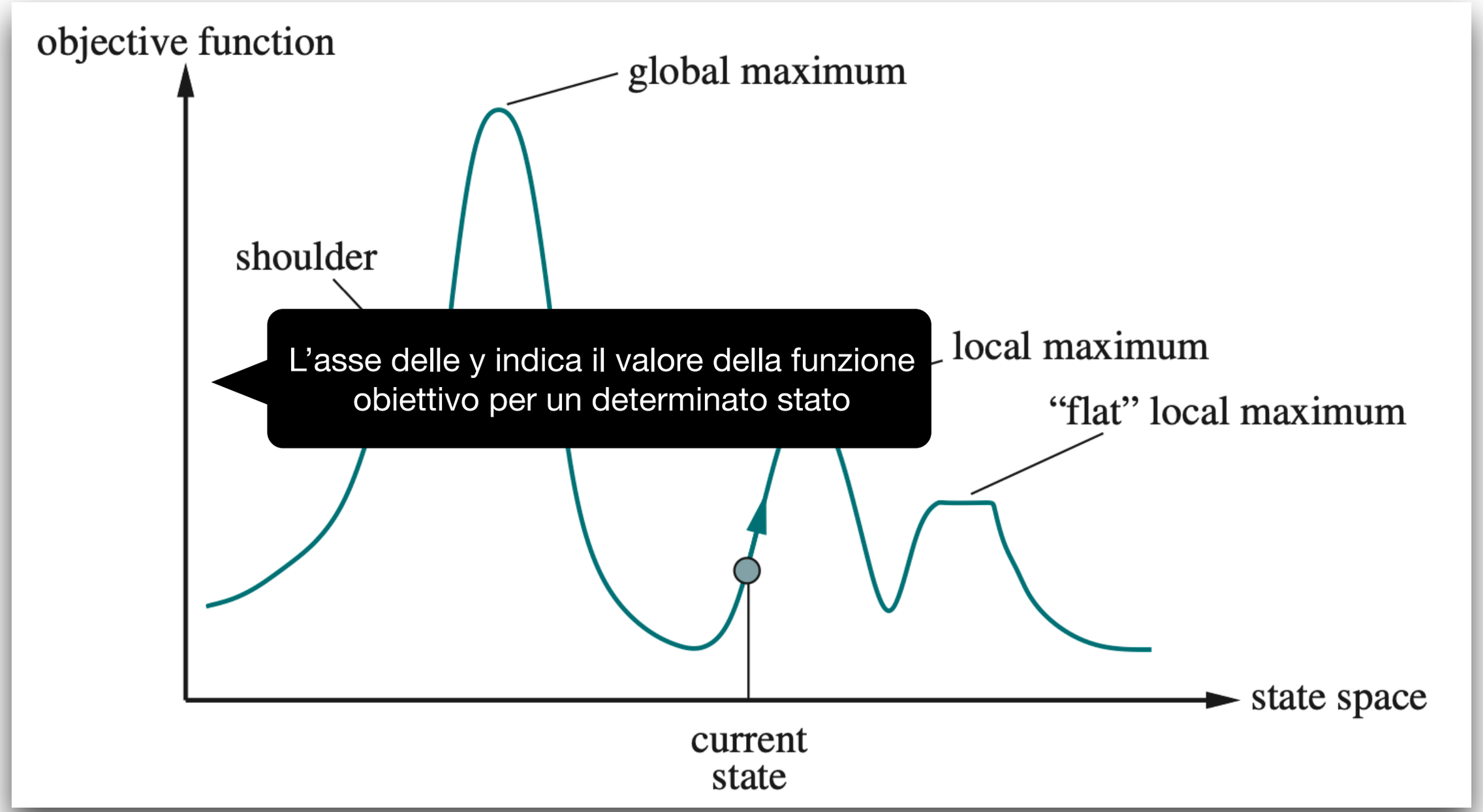
Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

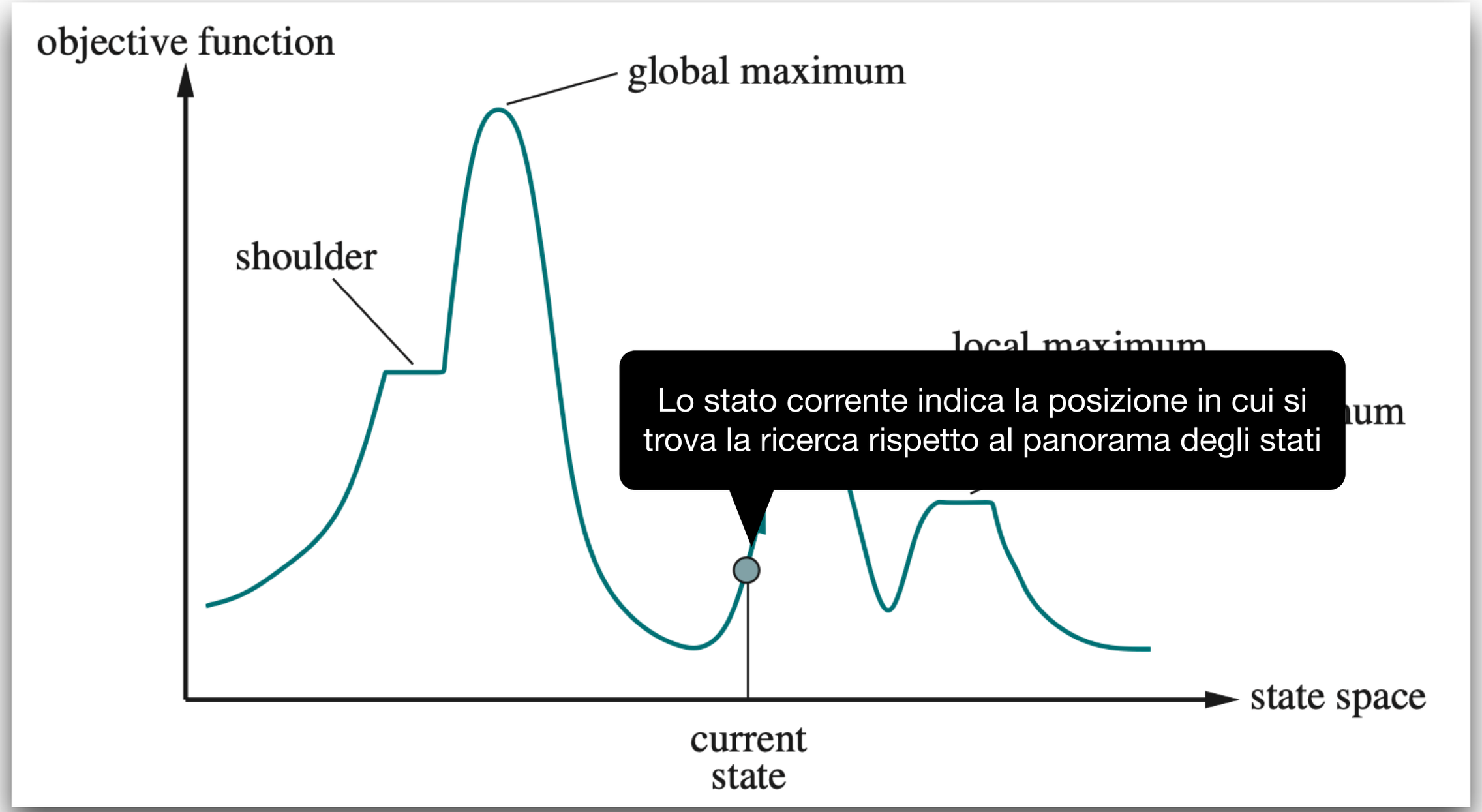
Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

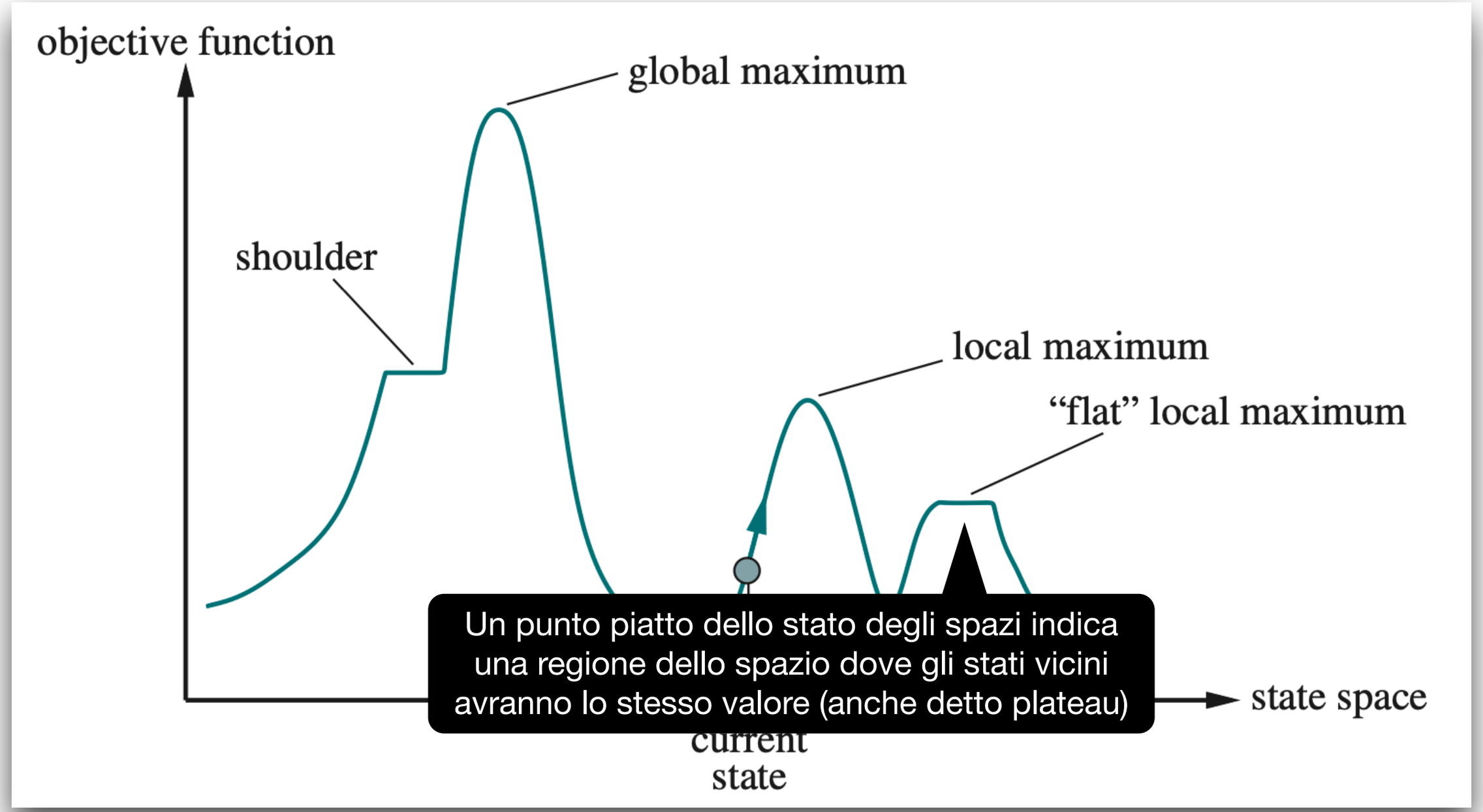
Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

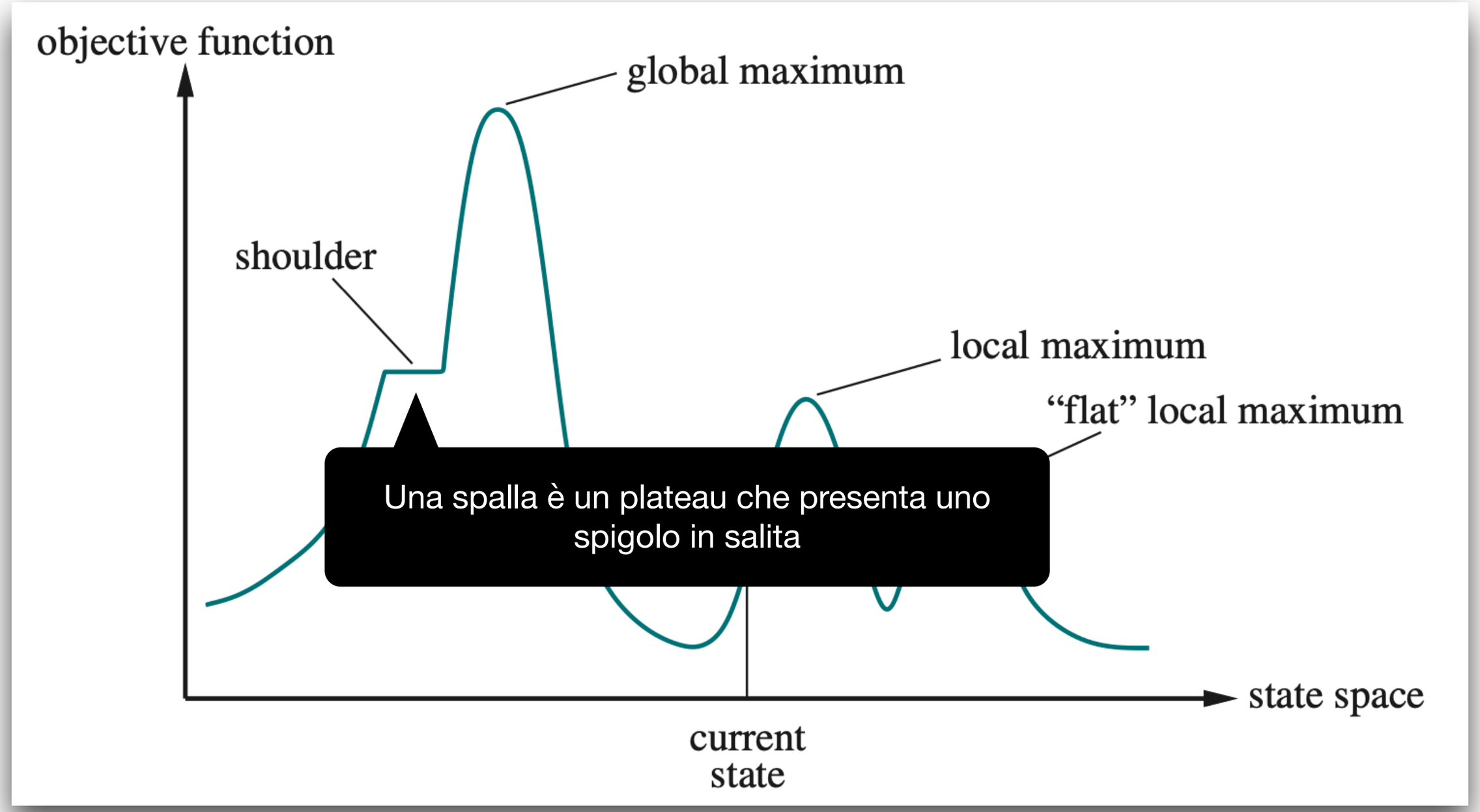
Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

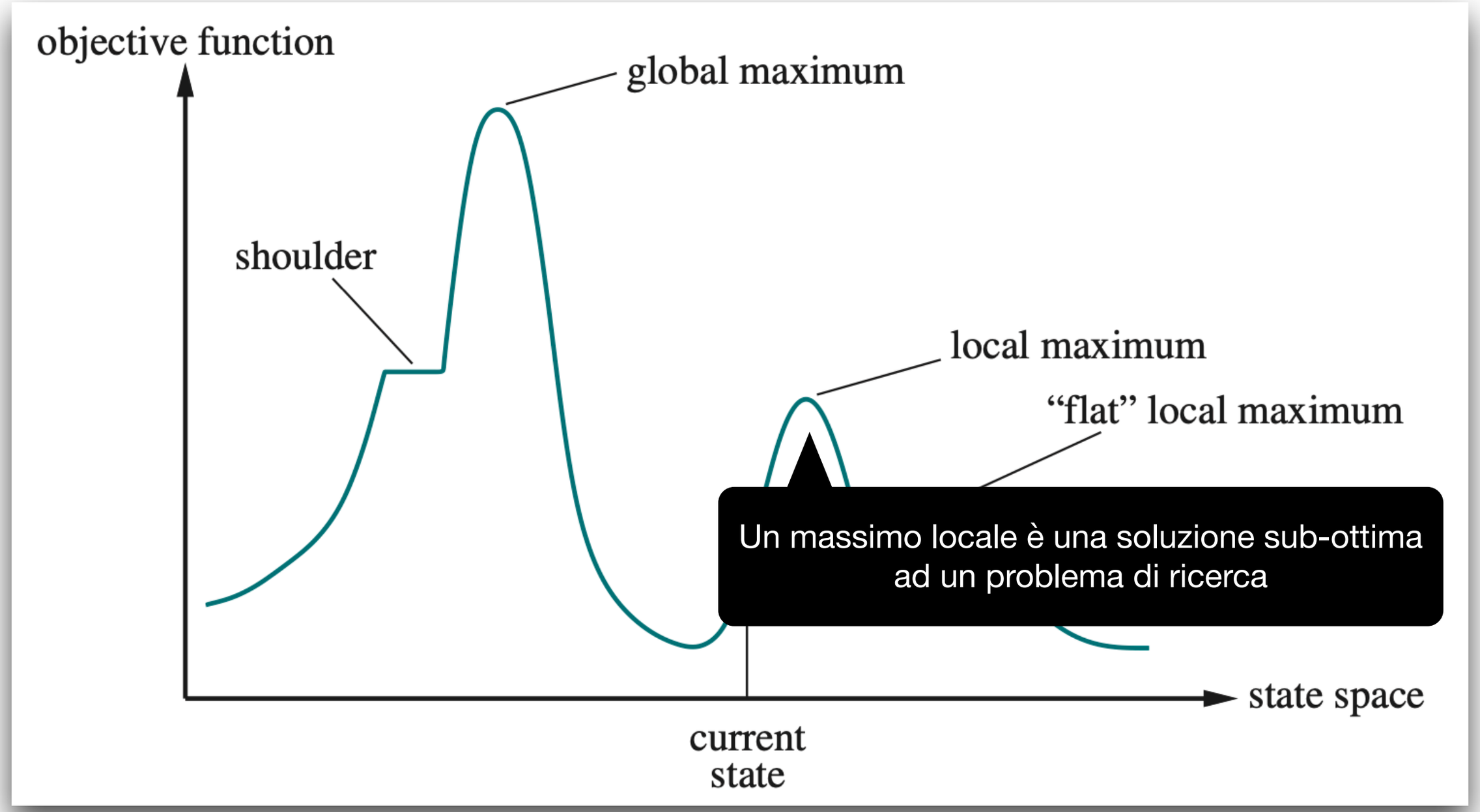
Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

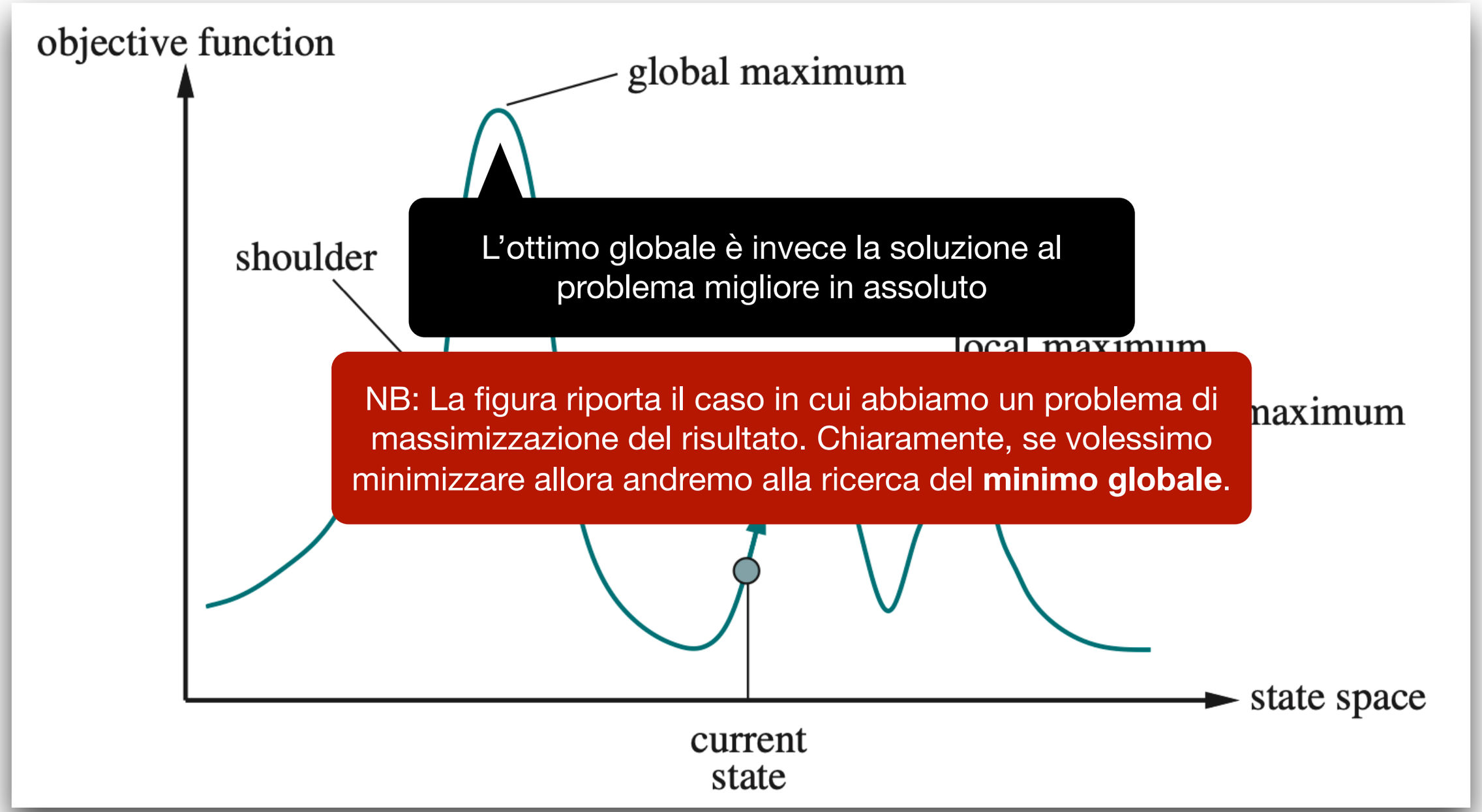
Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



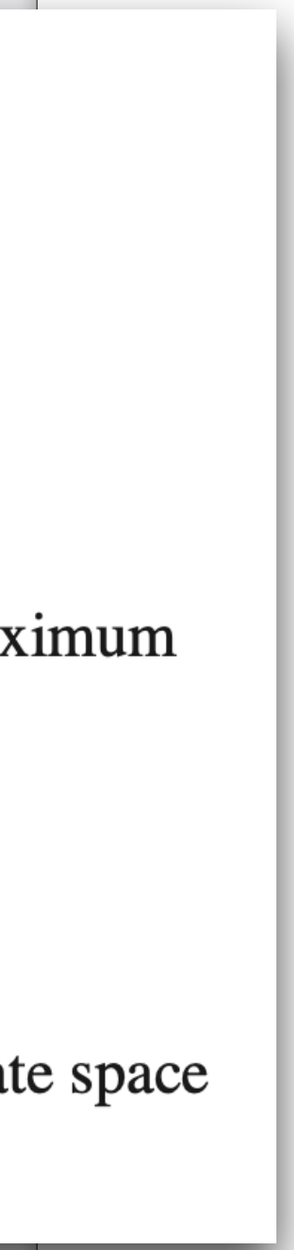
L'ottimo globale è invece la soluzione al problema migliore in assoluto

NB: La figura riporta il caso in cui abbiamo un problema di massimizzazione del risultato. Chiaramente, se volessimo minimizzare allora andremo alla ricerca del **minimo globale**.

Algoritmi di Ricerca Locale

Algoritmi di ricerca locale - Il panorama dello spazio degli stati

Per poter meglio comprendere il funzionamento degli algoritmi di ricerca locale, è utile visualizzare lo spazio degli stati in maniera diversa rispetto ad un albero.



Un algoritmo di ricerca locale **completo** trova sempre un obiettivo, se questo esiste.

Un algoritmo di ricerca locale **ottimo** trova sempre un minimo/ massimo globale.

Più formalmente, un massimo (minimo) *locale* è una soluzione *s* tale che, data una funzione di valutazione *f*, avveri la seguente equazione:

$$\forall s' \in N(s), f(s) \geq f(s')$$

Quando risolviamo un problema di massimizzazione (minimizzazione) cerchiamo un massimo (minimo) *globale*, s_{opt}, cioè una soluzione tale che

$$\forall s, f(s_{opt}) \geq f(s)$$

Banalmente, più è largo il neighborhood più è probabile che un massimo/minimo locale sia anche globale -> più è largo il neighborhood e migliora sarà qualità della soluzione.

Algoritmi di Ricerca Locale

L'algoritmo Hill-Climbing

L'algoritmo Hill-Climbing è il più semplice tra gli algoritmi di ricerca locale e ha l'obiettivo di "scalare" lo spazio di ricerca per trovare un massimo/minimo globale.

```
function HILL-CLIMBING(problema) returns uno stato che è un massimo

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    loop do

        vicino <— il successore di nodo_corrente di valore più alto
        if vicino.VALORE <= nodo_corrente.VALORE then return nodo_corrente.STATO
        nodo_corrente <— vicino
```

Algoritmi di Ricerca Locale

L'algoritmo Hill-Climbing

L'algoritmo Hill-Climbing è il più semplice tra gli algoritmi di ricerca locale e ha l'obiettivo di "scalare" lo spazio di ricerca per trovare un massimo/minimo globale.

```
function HILL-CLIMBING(problema) returns uno stato che è un massimo

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    loop do

        vicino <— il successore di nodo_corrente di valore più alto
        if vicino.VALORE <= nodo_corrente.VALORE then return nodo_corrente.STATO
        nodo_corrente <— vicino
```

/* Ad ogni passo, il nodo corrente viene rimpiazzato dal vicino migliore (lo pseudocodice fa riferimento ad un problema di massimizzazione). */

Algoritmi di Ricerca Locale

L'algoritmo Hill-Climbing

L'algoritmo Hill-Climbing è il più semplice tra gli algoritmi di ricerca locale e ha l'obiettivo di "scalare" lo spazio di ricerca per trovare un massimo/minimo globale.

```
function HILL-CLIMBING(problema) returns uno stato che è un massimo

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    loop do

        vicino <— il successore di nodo_corrente di valore più alto
        if vicino.VALORE <= nodo_corrente.VALORE then return nodo_corrente.STATO
        nodo_corrente <— vicino
```

/* L'algoritmo termina quando raggiunge un picco che non ha vicini di valore più alto. */

Algoritmi di Ricerca Locale

L'algoritmo Hill-Climbing

L'algoritmo Hill-Climbing è il più semplice tra gli algoritmi di ricerca locale e ha l'obiettivo di "scalare" lo spazio di ricerca per trovare un massimo/minimo globale.

```
function HILL-CLIMBING(problema) returns uno stato che è un massimo

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    loop do

        vicino <— il successore di nodo_corrente di valore più alto
        if vicino.VALORE <= nodo_corrente.VALORE then return nodo_corrente.STATO
        nodo_corrente <— vicino
```

È da notare che l'algoritmo non mantiene un albero di ricerca, per cui la struttura dati per il nodo corrente deve solo memorizzare lo stato e il valore della sua funzione obiettivo.

L'algoritmo non guarda al di là degli stati immediatamente vicini a quello corrente: a breve vedremo le implicazioni che questa strategia ha rispetto alla ricerca di una soluzione ottima.

Algoritmi di Ricerca Locale

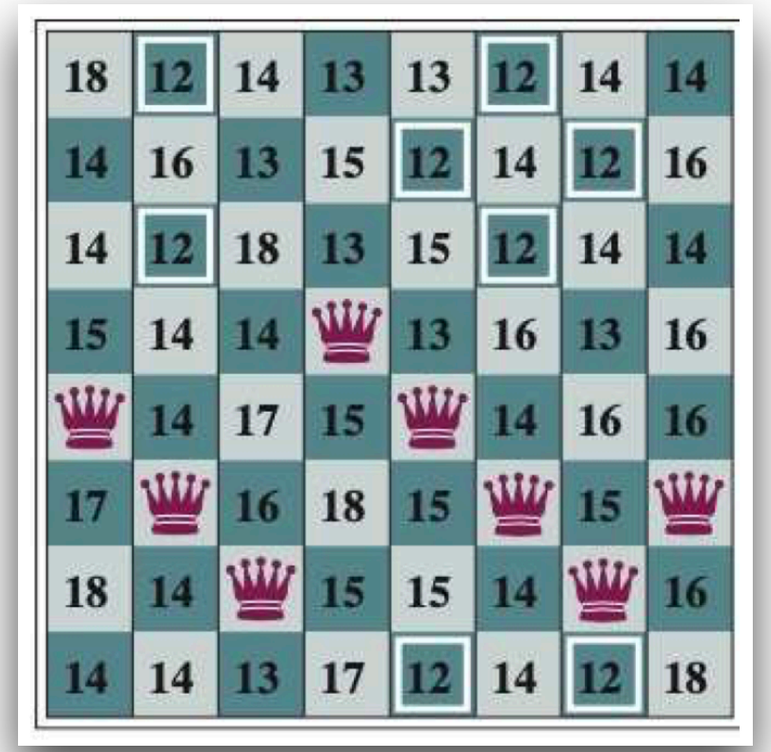
L’algoritmo Hill-Climbing, un esempio

Consideriamo il problema delle 8 regine, nella sua formulazione a stato completo - ogni stato ha 8 regine sulla scacchiera, una per colonna.

La funzione successore restituisce tutti gli stati possibili generati muovendo una singola regina in un’altra casella della stessa colonna - in questo modo, ogni stato avrà 8x7 = 56 possibili successori.

La funzione obiettivo è data dal numero di coppie di regine che si stanno attaccando a vicenda, in maniera diretta o indiretta. Vogliamo chiaramente minimizzare la funzione.

Quindi, il minimo globale è 0, che si verifica solo nel caso delle soluzioni.



La figura rappresenta uno stato del problema con h=17. Sono indicati i valori della funzione di ogni possibile successore ottenuto muovendo una regina nella sua colonna - le mosse migliori sono evidenziate.

Quando ci sono più successori con lo stesso valore di funzione obiettivo, allora l’algoritmo sceglierà casualmente la prossima mossa.

Tipicamente, l’Hill-Climbing è piuttosto rapido, poiché è abbastanza facile migliorare uno stato sfavorevole.

Algoritmi di Ricerca Locale

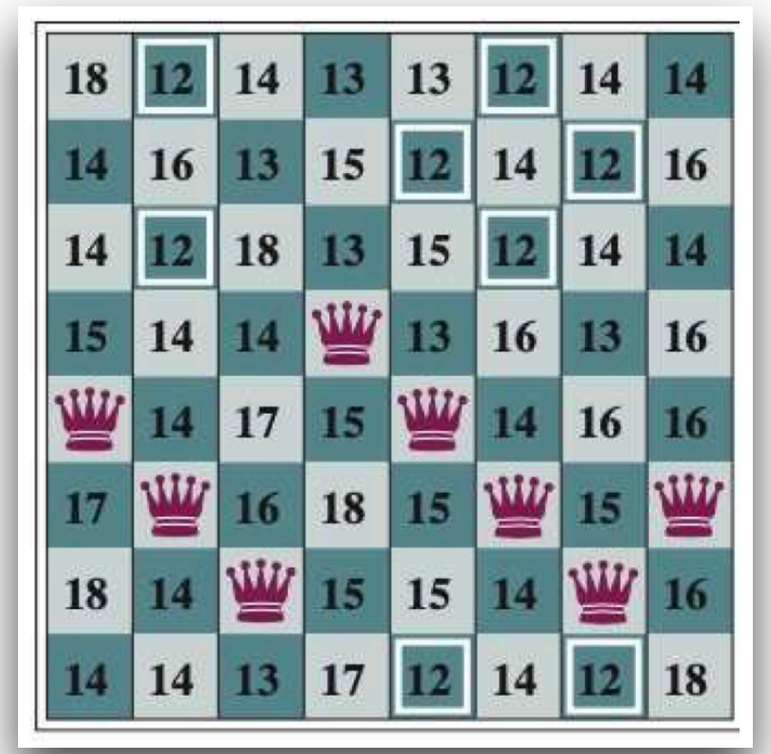
L’algoritmo Hill-Climbing, un esempio

Consideriamo il problema delle 8 regine, nella sua formulazione a stato completo - ogni stato ha 8 regine sulla scacchiera, una per colonna.

La funzione successore restituisce tutti gli stati possibili generati muovendo una singola regina in un’altra casella della stessa colonna - in questo modo, ogni stato avrà 8x7 = 56 possibili successori.

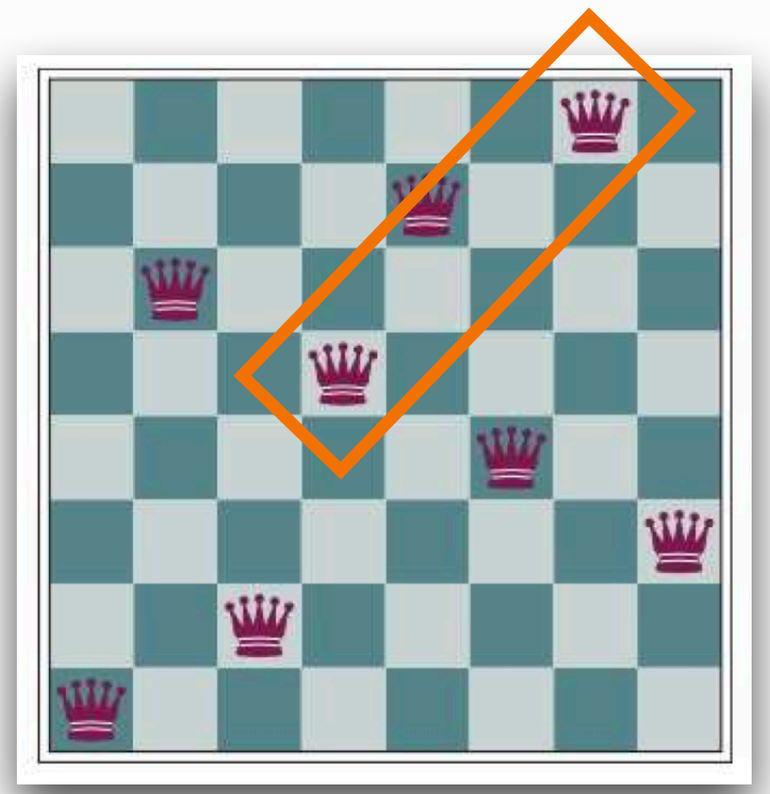
La funzione obiettivo è data dal numero di coppie di regine che si stanno attaccando a vicenda, in maniera diretta o indiretta. Vogliamo chiaramente minimizzare la funzione.

Quindi, il minimo globale è 0, che si verifica solo nel caso delle soluzioni.



In soli cinque passi, l’algoritmo può arrivare ad uno stato quasi ottimo.

La funzione h sarà uguale ad 1 e l’algoritmo si fermerà poiché tutti i successori hanno costo più alto.

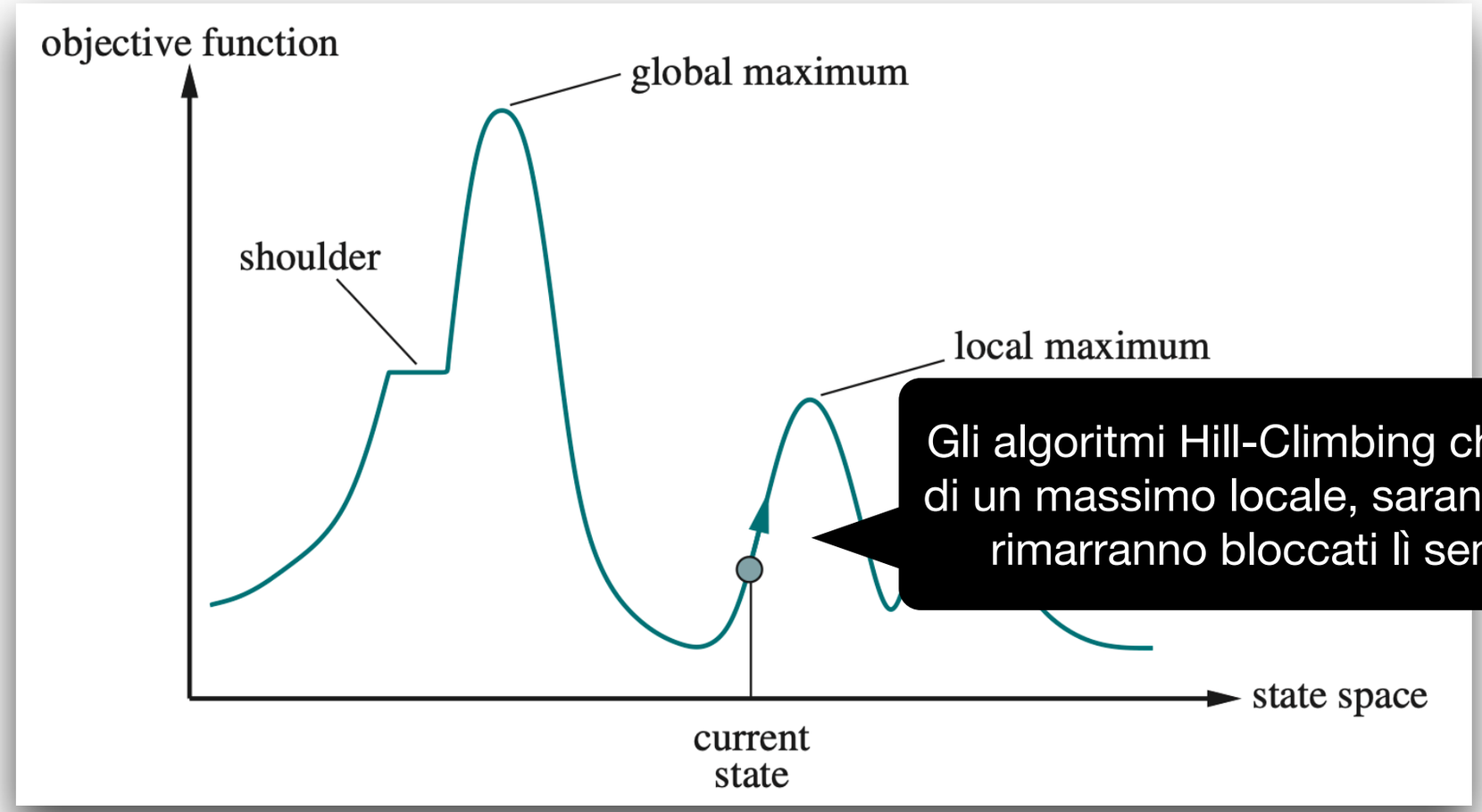


Algoritmi di Ricerca Locale

L'algoritmo Hill-Climbing, alcune considerazioni e svantaggi

L'algoritmo viene talvolta chiamato *ricerca locale greedy*, poiché segue una strategia simile agli algoritmi di ricerca best-first greedy che abbiamo visto in precedenza - sceglie uno stato "buono" senza pensare a come andrà avanti.

Sfortunatamente, l'algoritmo Hill-Climbing spesso non riesce ad identificare la soluzione migliore a livello globale per una serie di ragioni.



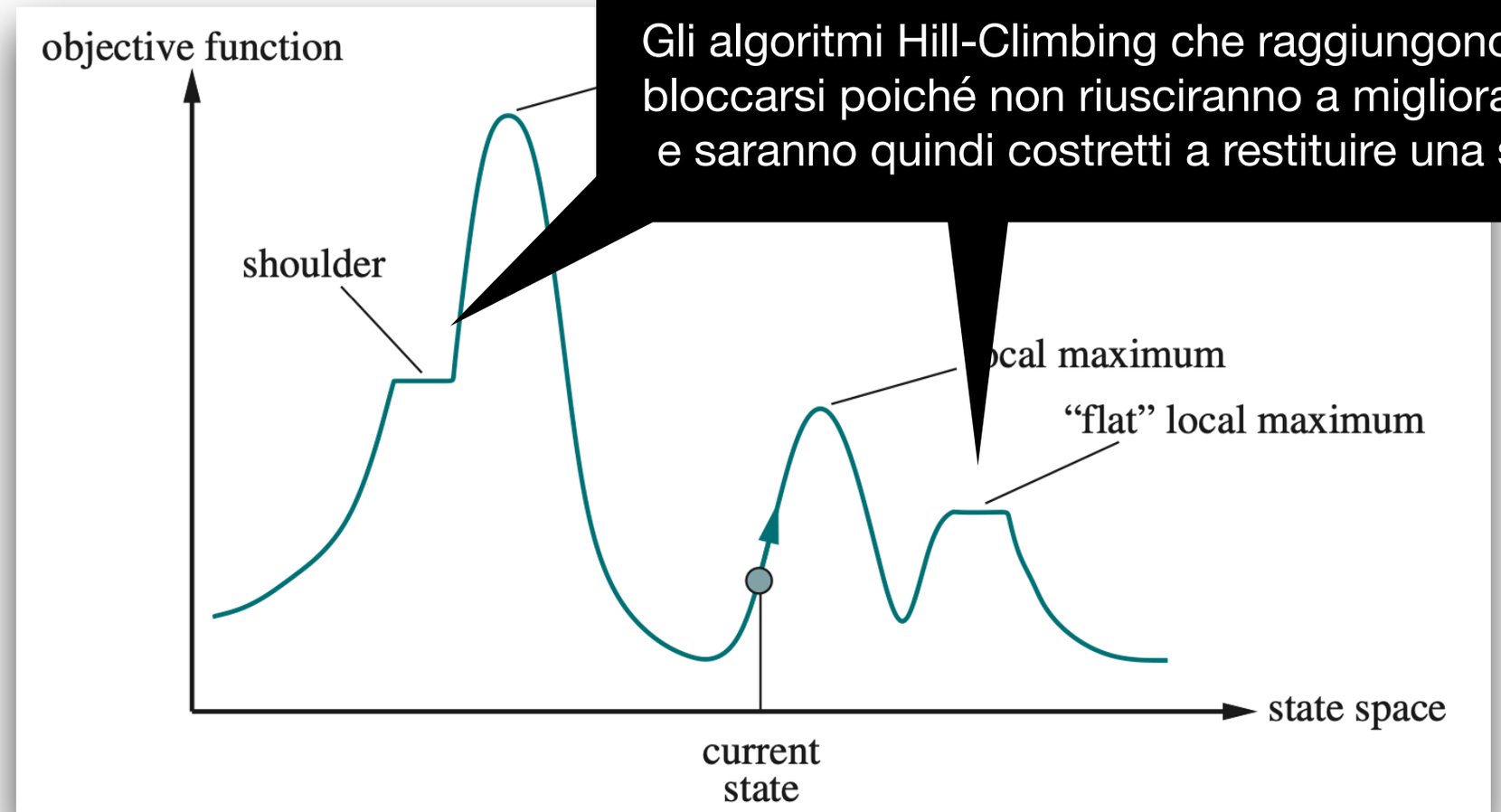
Gli algoritmi Hill-Climbing che raggiungono la vicinanza di un massimo locale, saranno attirati verso il picco, ma rimarranno bloccati lì senza poter andare altrove.

Algoritmi di Ricerca Locale

L'algoritmo Hill-Climbing, alcune considerazioni e svantaggi

L'algoritmo viene talvolta chiamato *ricerca locale greedy*, poiché segue una strategia simile agli algoritmi di ricerca best-first greedy che abbiamo visto in precedenza - sceglie uno stato "buono" senza pensare a come andrà avanti.

Sfortunatamente, l'algoritmo Hill-Climbing spesso non riesce ad identificare la soluzione migliore a livello globale per una serie di ragioni.



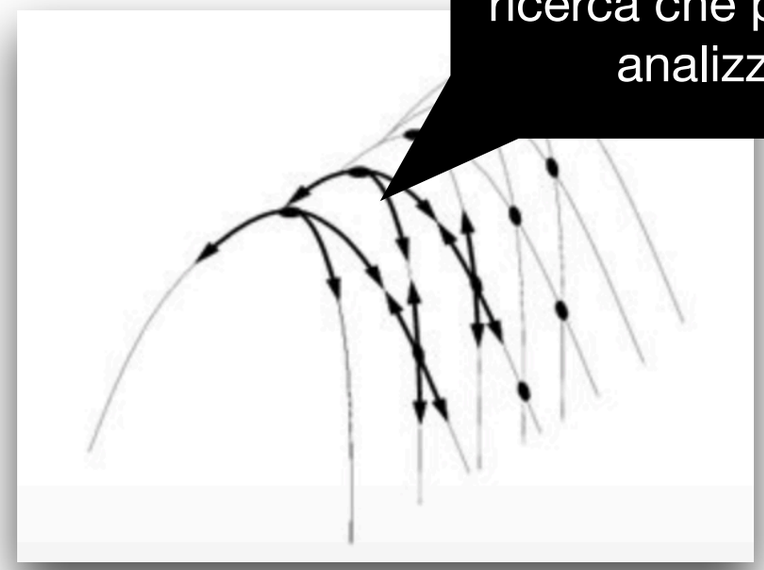
Gli algoritmi Hill-Climbing che raggiungono un plateau rischiano di bloccarsi poiché non riusciranno a migliorare la soluzione corrente e saranno quindi costretti a restituire una soluzione sub-ottimale.

Algoritmi di Ricerca Locale

L’algoritmo Hill-Climbing, alcune considerazioni e svantaggi

L’algoritmo viene talvolta chiamato *ricerca locale greedy*, poiché segue una strategia simile agli algoritmi di ricerca best-first greedy che abbiamo visto in precedenza - sceglie uno stato “buono” senza pensare a come andrà avanti.

Sfortunatamente, l’algoritmo Hill-Climbing spesso non riesce ad identificare la soluzione migliore a livello globale per una serie di ragioni.



I cosiddetti “ridge” (creste) rappresentano delle porzioni dello spazio di ricerca che presentano una brusca variazione. In questi casi, l’algoritmo analizzerà una serie di massimi locali da cui è difficile uscire.

aximum

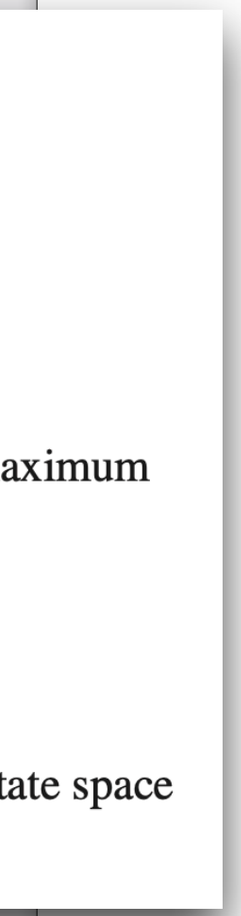
tate space

Algoritmi di Ricerca Locale

L'algoritmo Hill-Climbing, alcune considerazioni e svantaggi

L'algoritmo viene talvolta chiamato *ricerca locale greedy*, poiché segue una strategia simile agli algoritmi di ricerca best-first greedy che abbiamo visto in precedenza - sceglie uno stato "buono" senza pensare a come andrà avanti.

Sfortunatamente, l'algoritmo Hill-Climbing spesso non riesce ad identificare la soluzione migliore a livello globale per una serie di ragioni.



Quindi, massimi locali, plateau e creste possono deteriorare le prestazioni degli algoritmi Hill-Climbing. In ognuno di questi casi, infatti, questi algoritmi raggiungeranno un punto dal quale non riusciranno a fare ulteriori progressi.

Statisticamente, il problema delle 8 regine viene risolto solo nel 14% dei casi. In media, tuttavia, l'algoritmo richiederà solo 4 passi per trovare una soluzione.

È da notare il fatto che lo spazio degli stati del problema delle 8 regine è pari a 8^8, circa 17 milioni di stati. Quindi, tutto sommato, gli algoritmi Hill-Climbing non sono sempre così male.

Vale la pena cercare soluzioni per superare alcuni dei limiti che abbiamo visto.

L'algoritmo Hill-Climbing, miglioramenti

Potrebbe valere la pena di continuare la ricerca con una mossa laterale nella speranza che il plateau sia effettivamente uno spalla?

Generalmente si, ma non possiamo consentire all'algoritmo di eseguire un numero illimitato di mosse laterali, poiché potrebbe andare in ciclo infinito una volta raggiunto un massimo locale piatto che non è una spalla.

Una soluzione diffusa è quella di stabilire un limite massimo al numero di mosse laterali. Questo chiaramente aumenta il tasso di successo dell'algoritmo, impattando però la sua efficienza.

Ad esempio, impostando un limite di 100 mosse laterali consecutive nel problema delle 8 regine, il tasso di successo arriva al 94% (dal 14% dell'algoritmo di base) ma il numero medio di mosse salirà a 21 (da 4).

Esistono diverse altre varianti dell'algoritmo di base dell'Hill-Climbing, le quali vanno sostanzialmente a lavorare sulla probabilità che l'algoritmo riesce ad uscire da un massimo locale ed esplorare lo spazio di ricerca in maniera più efficace.

Anche in questi casi, però, l'efficienza degrada. La scelta dell'algoritmo da utilizzare dipende dal tipo di problema da affrontare.

Algoritmi di Ricerca Locale

L’algoritmo Hill-Climbing, varianti

L’Hill-Climbing stocastico sceglie a caso tra tutte le mosse che vanno verso l’alto; a differenza dell’algoritmo di base, non sceglie immediatamente la mossa più conveniente, aumentando le chance di navigare lo spazio di ricerca in maniera più estesa.

L’Hill-Climbing con prima scelta può generare le mosse a caso fino a trovarne una migliore dello stato corrente. Questa strategia è molto buona quando uno stato ha molti (migliaia) successori, poiché l’algoritmo riuscirà a navigare meglio lo spazio di ricerca.

L’Hill-Climbing con riavvio casuale può consentire alla ricerca di ripartire da un punto a caso dello spazio di ricerca - è da notare che generare uno stato casuale in uno spazio degli stati può essere di per sé un problema difficile.

Il riavvio casuale consente, in pratica, all’algoritmo di avviare una serie di ricerche hill-climbing. Risulta essere completo perché prima o poi dovrà generare, come stato iniziale, proprio un obiettivo.

Più in generale, il successo dell’ultimo algoritmo dipende dalla forma del panorama dello spazio degli stati. Se ci sono pochi massimi locali e plateau, allora troverà una soluzione molto velocemente. Ma purtroppo, non è sempre così.

I problemi NP-hard hanno tipicamente un numero esponenziale di massimi locali.

Algoritmi di Ricerca Locale

L’algoritmo Simulated Annealing

Un algoritmo che non scende mai a “valle” verso stati con valore inferiore sarà sicuramente incompleto, poiché potrebbe restare bloccato in un massimo/minimo locale.

Di contro, un’esplorazione completamente casuale è completa ma estremamente inefficiente - dovrà continuamente navigare lo spazio di ricerca senza una strategia.

Sembra perciò sensato combinare le due cose, ovvero proporre una ricerca hill-climbing con un’esplorazione casuale, per migliorare sua efficienza che completezza.

Questo è quello che si propone di fare il **Simulated Annealing**.

In metallurgia, l’annealing è il processo usato per indurire i metalli o il vetro riscaldandoli ad altissime temperature e raffreddandoli gradualmente, permettendo al materiale di cristallizzare in uno stato a bassa energia.

La similitudine fu introdotta per indicare che, in una fase iniziale, l’algoritmo “scuoterà” molto la ricerca, variando molto nel panorama degli stati, per poi man mano ridurre l’intensità di questo meccanismo, concentrandosi quindi su un punto specifico del panorama e cercando una soluzione ottima in quel contesto.

Algoritmi di Ricerca Locale

L'algoritmo Simulated Annealing

```
function SIMULATED-ANNEALING(problema, velocità_raffreddamento)
                                                    returns uno stato soluzione

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    for t <— 1 to ∞
       T <— velocità_raffreddamento[t]
       if T = 0 then return nodo_corrente
       successivo <— un successore scelto a caso di nodo_corrente
       ΔE <— successivo.VALORE - nodo_corrente.VALORE

       if ΔE > 0 then nodo_corrente <— successivo
       else nodo_corrente <— successivo solo con probabilità e^(ΔE/T)
```

Algoritmi di Ricerca Locale

L'algoritmo Simulated Annealing

```
function SIMULATED-ANNEALING(problema, velocità_raffreddamento)
                                                    returns uno stato soluzione

    nodo_corrente

    for t <- 1 to
        T <- veloc
        if T = 0 then return nodo_corrente
        successivo <- un successore scelto a caso di nodo_corrente
        ΔE <- successivo.VALORE - nodo_corrente.VALORE

        if ΔE > 0 then nodo_corrente <- successivo
        else nodo_corrente <- successivo solo con probabilità e^(ΔE/T)
```

/* Il parametro `velocità_raffreddamento` rappresenta una corrispondenza dal tempo alla "temperatura" */

Algoritmi di Ricerca Locale

L'algoritmo Simulated Annealing

```
function SIMULATED-ANNEALING(problema, velocità_raffreddamento)
                                                    returns uno stato soluzione

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    for t <— 1 to ∞
       T <— velocità_raffreddamento[t]
       if T = 0 then return nodo_corrente
       successivo <— un successore scelto a caso di nodo_corrente
       ΔE <— s
       if ΔE >
       else no                                          à e^(ΔE/T)
```

/* La velocità di raffreddamento impatterà la temperatura T ed indicherà quanto T venga ridotto ad ogni passo dell'algoritmo */

Algoritmi di Ricerca Locale

L'algoritmo Simulated Annealing

```
function SIMULATED-ANNEALING(problema, velocità_raffreddamento)
                                                    returns uno stato soluzione

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    for t <— 1 to ∞
       T <— velocità_raffreddamento[t]
       if T = 0 then return nodo_corrente
       successivo <— un successore scelto a caso di nodo_corrente
       ΔE <— successivo.VALORE - nodo_corrente.VALORE

       if ΔE > 0 then nodo_corrente <— successivo
       else nodo_corrente <— successivo solo con probabilità e^(ΔE/T)
```

/* Il ciclo interno è simile a quello dell'Hill-Climbing, ma questa volta il nodo successore sarà scelto a caso. */

Algoritmi di Ricerca Locale

L'algoritmo Simulated Annealing

```
function SIMULATED-ANNEALING(problema, velocità_raffreddamento)
                                                    returns uno stato soluzione

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    for t <— 1 to ∞
       T <— velocità_raffreddamento[t]
       if T = 0 then return nodo_corrente
       successivo <— un successore scelto a caso di nodo_corrente
       ΔE <— successivo.VALORE - nodo_corrente.VALORE

       if ΔE > 0 then nodo_corrente <— successivo
       else nodo_corrente <— successivo solo con probabilità e^(ΔE/T)
```

/* Se il successore ha un valore migliore, allora la mossa verrà sempre accettata e l'algoritmo procederà ad esplorare lo spazio intorno a quello stato. */

Algoritmi di Ricerca Locale

L'algoritmo Simulated Annealing

```
function SIMULATED-ANNEALING(problema, velocità_raffreddamento)
                                                    returns uno stato soluzione

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    for t <— 1 to ∞
       T <— velocità_raffreddamento[t]
       if T = 0 then return nodo_corrente
       successivo <— un successore scelto a caso di nodo_corrente
       ΔE <— successivo.VALORE - nodo_corrente.VALORE

       if ΔE > 0 then nodo_corrente <— successivo
       else nodo_corrente <— successivo solo con probabilità e^(ΔE/T)
```

/* In alternativa, il successore potrebbe essere accettato ugualmente, ma con una qualche probabilità inferiore ad 1. */

Algoritmi di Ricerca Locale

L'algoritmo Simulated Annealing

```
function SIMULATED-ANNEALING(problema, velocità_raffreddamento)
                                                    returns uno stato soluzione

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    for t <— 1 to ∞
       T <— velocità_raffreddamento[t]
       if T = 0 then return nodo_corrente
       successivo <— un successore scelto a caso di nodo_corrente
       ΔE <— successivo.VALORE - nodo_corrente.VALORE
       if ΔE > 0 then nodo corrente <— successivo
                                  on probabilità e^(ΔE/T)
```

/* Nella similitudine con la metallurgia, *ΔE* rappresenta la variazione di energia. */

Algoritmi di Ricerca Locale

L'algoritmo Simulated Annealing

```
function SIMULATED-ANNEALING(problema, velocità_raffreddamento)
                                                    returns uno stato soluzione

    nodo_corrente <— CREA-NODO(problema.STATO-INIZIALE)

    for t <— 1 to ∞
       T <— velocità_raffreddamento[t]
       if T = 0 then return nodo_corrente
       successivo <— un successore scelto a caso di nodo_corrente
       ΔE <— successivo.VALORE - nodo_corrente.VALORE

       if ΔE > 0 then nodo_corrente <— successivo
       else nodo_corrente <— successivo solo con probabilità e^(ΔE/T)
```

Vale la pena notare che la probabilità decresce esponenzialmente in base alla cattiva qualità della soluzione (ΔE). Inoltre, decresce anche con la temperatura T - questa scenderà costantemente ad ogni passo dell'algoritmo.

In altri termini, mosse "cattive" verranno accettate più facilmente all'inizio poiché T sarà alto e diventeranno sempre meno probabili a mano a mano che T si abbassa.

Se la temperatura si abbassa abbastanza lentamente, l'algoritmo troverà un ottimo globale con una probabilità tendente ad 1.

Algoritmi di Ricerca Locale

L'algoritmo Local Beam

Memorizzare un solo nodo può sembrare una reazione estrema ai problemi legati alle limitazioni di memoria. Una variante è rappresentata dalla beam search.

Algoritmi di Ricerca Informata

Migliorare l'occupazione di memoria, la Beam Search

La Beam Search è una variante della ricerca best-first che non conserva in memoria tutti i nodi generati, ma tiene ad ogni passo solo i *k* nodi più promettenti, con *k* definito come *ampiezza del raggio*.

L'ampiezza del raggio è definita in base ad una euristica (ad esempio, *h(n)* potrebbe rappresentare l'euristica); solo i nodi nel raggio vengono conservate e considerate come candidati per la soluzione.

La Beam Search è, quindi, a tutti gli effetti una soluzione *greedy* alla risoluzione dei problemi, poiché espanderà sempre il nodo che ritiene essere più utile al raggiungimento della soluzione.

Da un punto di vista pratico, ad ogni livello dell'albero di ricerca l'algoritmo genera tutti i successori degli stati a quel livello, ordinandoli in maniera decrescente in base all'euristica scelta. Conserverà poi solo i primi *k* di ogni livello, che saranno poi espansi.

Chiaramente, maggiore sarà il raggio e minore sarà il numero di nodi potati. L'algoritmo restituirà la prima soluzione identificata.

Sebbene migliori l'occupazione di memoria (in questo caso, la complessità spaziale sarà pari ad *O(kd)*), l'algoritmo *non è completo né ottimale*.

Algoritmi di Ricerca Locale

L’algoritmo Local Beam

Memorizzare un solo nodo può sembrare una reazione estrema ai problemi legati alle limitazioni di memoria. Una variante è rappresentata dalla beam search.

L’algoritmo di ricerca local bean tiene traccia di k stati anziché uno. All’inizio comincia con k stati generati casualmente: ad ogni passo, sono poi generati i successori di tutti i k stati.

Se uno di questi è un obiettivo, la ricerca termina. Altrimenti, scegli i k successori migliori dalla lista e ricomincia.

In linea teorica, potrebbe sembrare simile all’Hill-Climbing con riavvio casuale. Tuttavia, nella ricerca con riavvio casuale ogni processo di ricerca è del tutto indipendente dagli altri. Nella local beam, invece, la ricerca è “informata”, nel senso che informazione viene passata dall’uno all’altro thread di ricerca paralleli.

Questo aumenta le chance di navigare il panorama degli stati verso stati più “promettenti”, possibilmente aumentando le chance di trovare un ottimo globale.

Da un altro punto di vista, la local beam potrebbe soffrire di una carenza di diversificazione tra i k stati, concentrando quindi la ricerca troppo rapidamente in una piccola regione dello spazio.

L’algoritmo Local Beam

Un’alternativa è rappresentata dalla ricerca **local beam stocastica**, che è ispirata ai concetti già discussi con l’Hill-Climbing stocastico.

Invece di scegliere i k successori migliori tra i candidati, si scelgono k successori a caso. Tuttavia, ai migliori di questi si assegna una probabilità maggiore di scelta.

Questo approccio ricorda il processo di *selezione naturale*, in cui i “successori” (progenie) di uno “stato” (organismo) popolano la generazione successiva in base al loro “valore adattativo” (idoneità, anche detta *fitness*).

Nella prossima lezione, andremo ad analizzare una variante della local bean stocastica ispirata alla *teoria evolutiva di Darwin*, in cui gli stati successori sono generati come combinazione di due *stati padre* - invece di modificarne uno.

Gli algoritmi genetici considerano infatti il problema di ottimizzazione in analogia con la teoria evolutiva e simulano il processo di selezione naturale dove le configurazioni sono viste come individui di una popolazione e gli individui meno adatti “muoiono” senza riprodursi.

L’idea alla base, chiaramente, sarà quella di generare di volta in volta una generazione nel complesso migliore della precedente, aumentando le chance di identificare una soluzione ottima globale.

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laurea triennale in Informatica

Fondamenti di Intelligenza Artificiale

Lezione 6 - Algoritmi di ricerca locale (I)

Algoritmi di Ricerca Locale

Materiale aggiuntivo ed esercizi consigliati

Il codice sorgente che implementa un algoritmo di ricerca di tipo Simulated Annealing per la risoluzione del problema del commesso viaggiatore sarà disponibile sulla piattaforma e-learning.

A partire da questo (o anche implementandone uno nuovo), risolvere il problema del commesso viaggiatore utilizzando un algoritmo di tipo Hill-Climbing tradizionale e con riavvio casuale. Spiegare poi le differenze, in termini di prestazioni, con la soluzione implementata con Simulated Annealing.