Gli algoritmi di ricerca locale operano cercando a partire da uno stato iniziale e procedendo verso gli stati adiacenti senza tener traccia dei cammini, ne degli stati già raggiunti. Non sono algoritmi sistematici e quindi non sono completi.
Usano poca memoria e spesso trovano soluzioni anche in spazi degli stati grandi o infiniti per cui usare algoritmi sistematici non sarebbero praticabili.
Gli stati di un problema sono disposti in un **panorama dello spazio degli stati**. Ogni punto (ovvero uno stato), ha un'altezza definita dal valore della funzione obiettivo. Se lo scopo è trovare un massimo globale per la funzione chiamiamo il processo di ricerca Hill Climbing. Al contrario se è trovare un minimo globale, discesa del gradiente.

Hill Climbing

Tiene traccia di un solo stato corrente e a ogni iterazione passa allo stato vicino con valore più alto, cioè punta nella direzione che presenza l'ascesa più rapida, senza guardare oltre gli stati adiacenti. Viene detto anche ricerca locale greedy perché sceglie uno stato vicino senza passare a come andrà avanti.

L'algoritmo può procedere molto rapidamente verso una soluzione, perché di solito è molto semplice migliorare una situazione sfavorevole. Ma l'algoritmo rimane per i seguenti problemi:

- **Massimi locali:** un massimo locale è un picco più alto degli stati vicino ma inferiore al massimo globale. Gli algoritmi Hill Climbing che raggiungono la vicinanza ad un massimo locale saranno attirati verso il picco, ma rimarranno bloccati lì.
- **Creste:** Danno origine a una sequenza di massimi locali molto difficili da esplorare.
- **Plateau:** Un'area piatta del panorama degli stati. Può essere un massimo locale piatto da cui non è possibile fare miglioramenti o una **spalla** da cui si potrà salire ulteriormente.

L'algoritmo non riesce a raggiungere ulteriori progressi in queste situazioni.
Una soluzione può essere quello di proseguire nella ricerca una volta raggiunto un plateau , con una **mossa laterale** sperando si tratti di una spalla. Ma se così non fosse, si incomberebbe in un ciclo infinito. Una soluzione a ciò può essere fornirnre un limite al numero di mosse laterali.

- **Hill Climbing stocastico:** Sceglie a caso tra tutte le mosse in salita (magari basando le probabilità sulle "pendenza" delle mosse). Ma converge più lentamente di quello che sceglie sempre la mossa più competente ma in alcuni casi si trova la soluzione migliore.
- **Hill Climbing a prima scelta:** genera a caso i successori fino a ottenerne uno preferibile allo stato corrente. Buono con stati con molti successori.
- **Hill climbing con riavvio casuale:** esegue una serie di ricerche Hill Climbing partendo da stati iniziali casuali fino a quando raggiunge un obiettivo. E' completo con probabilità 1, perché prima o

poi dovrà generare come stato inziale un nodo obiettivo.

Simulated anealing

Simile all' Hill Climbing però invece di prendere la scelta migliore, viene scelta una mossa casuale. Se la misura migliora la situazione, viene sempre accettata; in caso contrario l'algoritmo l'accetta con probabilità inferiore a 1.
La probabilità decresce esponenzialmente con la cattiva qualità della mossa, misurata dal peggiormaneto della valutazione (ΔE). La probabilità decresce anche con la "temperatura" (T) che scende costantemente: le mosse "cattive" saranno accettate più facilmente all'inizio e diventeranno sempre meno probabili a lungo andare. Se T viene decrementato abbastanza lentamente, si raggiungerà la soluzione ottima.

Local Beam Search

Tiene traccia di k stati anziché uno. All'inizio comincia con k stati generati casualmente: a ogni passo sono generati i successori di tutti i k stati. Se uno qualsiasi di questo è un obiettivo l'algoritmo termina. Altrimenti sceglie k successori migliori e ricomincia.
Lo stato che genera i migliori successori da informazione che i suoi successori forniscono progressi maggiori e l'algoritmo sposta le sue risorse su questi abbandonando le precedenti ricerche.
Ma in questo modo la ricerca si limita ad una piccola regione dello spazio degli stati, rallentando le prestazioni. Una variante è la **beam search stocastica**, invece di scegliere i migliori k successori, si scelgono i successori con probabilità proporzionale al loro valore.

Azioni non deterministiche

Quando l'ambiente è non deterministico, l'agente non sa in quale stato si arriverà dopo un'azione.
Chiamiamo **stato-credenza** un insieme di stati fisici che l'agente ritiene possibile.
La soluzione in ambienti non determinstici la soluzione non è una sequenza di azioni ma un piano condizionale che specifica cosa fare in base alle percezioni ricevute durante l'attuazione del piano.

ES
$Risultati(1, Aspira) = \{5, 7\}.$

Un piano condizionale per questo tipo di agente è:

$[\text{ Aspira, if stato=5 then [Destra,Aspira] else [] }]$

Alberi and-or

Una soluzione per un problema di ricerca AND-OR è un sottoalbero dell'albero di ricerca completo che:

- ha un nodo obiettivo in ogni foglia.
- specifica una sola azione in ogni nodo OR.
- include ogni ramo uscente da ogni nodo AND.

Se lo stato corrente è identico a uno stato sul cammino dalla radice allora l'algoritmo termina con un fallimento (non significa che la soluzione non esiste, ma che bisogna cercare in un altro cammino).
Potrebbe presentarsi però una soluzione ciclica, dove bisognerà continuare a provare una determinata azione finchè non funzionerà.
In tal caso il piano di contingenza diventa:

$$[\text{ Aspira, while stato}{=}5 \text{ do Destra,Aspira}]$$

Quando un piano ciclico è una soluzione? Una condizione minima è che ogni foglia sia un obiettivo e che sia sempre raggiungibile.

Sensorless

Quando le osservazioni dell'agente non forniscono alcuna informazione, abbiamo un problema **conformante** o **senza sensori**.
L'agente, in questo tipo di problemi, può forzare il mondo ad entarre in un certo stato mediante le sue azioni.
La soluzione ad un problema del genere non è un piano di contingenza ma una sequenza di azioni.
La ricerca viene fatta nello spazio degli stati-credenza non lo spazio degli stati fisici. In quello spazio, il problema è completamente osservabile perché l'agente conosce sempre il proprio stato credenza.
La soluzione è una sequenza di azioni e non un piano condizionale perché le percezioni sono sempre vuote e non ci sono contingenze da considerare, ciò vale anche per ambienti non deterministici.

Trasformiamo il problema fisico in un problema di stati-credenza.
Il problema originale P ha come componenti $AZIONI_P, RISULTATI_P$, il problema stati-credenza è:

- Stati: Contiene ogni possibile sottoinsieme degli stati fisici. Se P ha N stati, allora vi sono 2^N stati-credenza.
- Stato inziale: Generalmente è lo stato-credenza costituito da tutti gli stati in P.

- Azioni: Supponiamo che l'agente si trovi nello stato-credenza $b = \{ s_1, s_2 \}$, ma $AZIONI_P(s_1) \neq AZIONI_P(s_2)$, allora l'agente non sa quali siano leggittime. Se le azioni non sono leggittime non hanno effetti sull'ambiente, allora:
  $AZIONI(b) = \bigcup_{s \in b} AZIONI_P(s)$
  altrimenti
  $AZIONI(b) = \bigcap_{s \in b} AZIONI_P(s)$

- Modello di transizione: Per azioni determinstiche, ogni nuovo stato-credenza contiene un solo risultato:
  $b' = Risultato(b, a) = \{ s' : s' = RISULTATO_P(s, a) \ \wedge \ s \in b \}$
  Nel caso non deterministico, il nuovo stato-credenza consiste di tutti i possibili risultati ottenuti applicando l'azione a uno qualsiasi degli stati nello stato-credenza corrente:
  $b' = Risultato(b, a) = \bigcup_{s \in b} RISULTATI_P(s, a)$.

- Test obiettivo: L'agente può raggiungere l'obiettivo se qualche stato s nello stato credenza soddisfa il test-obiettivo del problema sottostante.
- Costo azione: Il costo potrebbe dipendere dallo stato ma assumiamo di no.

Nella ricerca su grafo normale, i nuovi stati raggiunti vengono controllati per tracciare i cammini ridondanti. Ciò funziona anche per gli stati-credenza: se viene raggiunto uno stato-credenza **soprainsieme** di un altro $(\{ 1, 3, 5, 7 \}$ è un soprainsieme di $\{ 5, 7 \})$, questo può essere potato e dunque scartato poiché una soluzione da $\{ 1, 3, 5, 7 \}$ deve essere una soluzione per ogniuno dei singoli stati e per qualsiasi combinazione ottenuta da questi stati; non serve dunque risolvere il soprainsieme ma concentrarci unicamente sullo stato più piccolo e più facile.
Viceversa, ciò vale anche per i sottoinsiemi, se $\{ 1, 3, 5, 7 \}$ è già stato generato ed è risolvibile non ha senso considerare i suoi sottoinsiemi.

Il problema più grande è la vastitità dello spazio degli stati-credenza: con N stati fisici si hanno 2^N stati-credenza.
Un approccio è quello di evitare algoritmi di ricerca standard, che trattano gli stati-credenza come scatole nere alla pari di qualsiasi altro stato e di considerare l'interno degli stati-credenza per sviluppare algoritmi di **ricerca stato-credenza incrementale** che costruiscono la soluzione considerando uno stato fisico per volta. Per esempio $\{ 1, 2, 3, 4, 5, ..., 8 \}$ e dobbiamo trovare una sequenza di azioni che funzioni in tutti gli 8 stati. Si cerca una soluzione che funzioni per 1, si verifica che questa vada bene per 2, fino a 8, se non funzione ad esempio per 2, si cerca un'altra soluzione partendo da 1. Scopre presto i fallimenti (quando uno stato-credenza è irrisolvibile, avviene solitamente che un piccolo sottoinsieme dello stato-credenza, costituito dai primi stati esaminati, è irrisolvibile). Cerca, però, una soluzione che vada bene a tutti gli stati.

Ambienti parzialmente osservabili

Nel caso di problema parzialmente osservabile, la specifica del problema includerà una funzione $PERCEZIONE(s)$ che restituisce la percezione ricevuta dall'agente in un dato stato.
Se i sensori non sono deterministici, possiamo utilizzare una funzione percezione che restituisce un insieme di possibili percezioni.
Nello stato 1 di aspirapolvere , la funzione percezione sarà $[S, Sporco]$ anche nello stato 3. Quindi data questa percezione iniziale lo stato credenza inziale sarà $\{ 1, 3 \}$.
Il modello di transizione qui opera in 3 fasi:

- **Fase di predizione:** calcola lo stato credenza risultante dall'azione, $RISULTATO(b, a)$, come nei problemi senza sensori.
  $b^* = RISULTATO(b, a)$
- **Fase di percezioni possibili:** calcola l'insieme delle percezioni che potrebbero essere asserite nello stato-credenza prodotto:
  $PERCEZIONI - POSSIBILI(b^*) = \{ o : o = PERCEZIONE(s) \ \wedge \ s \in b^* \}$.
- **Fase di aggiornamento:** calcola per ogni possibile percezione lo stato-credenza che risulterebbe da essa:
  $b_0 = AGGIORNA(b^*, o) = \{ s : o = PERCEZIONE(s) \ \wedge \ s \in b^* \}$.

Mettendo insieme le tre fasi otteniamo i possibili stati credenza risultanti da una data azione e le conseguenti percezioni possibili:
$RISULTATI(b, a) = \{ b_0 : b_0 = AGGIORNA(PREDIZIONE(b, a), o) \ \wedge \ o \in PREDIZIONI - POSSIBILI(PREDIZIONE(b, a)) \}$

Ricerca online

Nella ricerca online un agente opera alternando computazione e azione: prima esegue un azione e poi osserva l'ambiente e determina l'azione successiva. Utile in ambienti dinamici e semidinamici e in domini non deterministici , perchè consente all'agente di concentrare le attività di calcolo nelle contingenze che si verificano sicuramente anziché su quelle che si pianificano e magari non avvengono. Comunque un agente più pianifica in anticipo, meno incomberrà in situazioni problematiche.

In ambienti ignoti l'agente non conosce stati o effetti delle sue azioni e deve sfruttare le sue azioni come esperimenti per apprendere l'ambiente.

Un problema di ricerca online viene risolto con attività di elaborazione, percezione e azione.
Supponiamo che l'ambiente sia determinstico e completamente osservabile e stabiliamo che l'agente conosce:

- Le azioni eseguibili in un certo stato (solo una volta nello stato).
- Il costo nell'eseguire un azione in un certo stato, portando l'ambiente in un altro stato (solo dopo aver eseguito l'azione però).
- Il test obiettivo.

> OSS:
> Un agente non può determinare $RISULTATO(s, a)$ se non trovandosi effettivamente in s ed eseguendo a.

L'agente potrebbe aver accesso a una funzione euristica $h(s)$ capace di stimare la distanza dallo stato corrente al nodo goal.
Generalmente lo scopo dell'agente è raggiungere uno stato obiettivo minimizzando il costo, questo è rappresentato da quello totale ovvero del cammino effettivamente percorso dall'agente, questo poi si confronta con quello del cammino che l'agente seguirebbe se conoscesse in anticipo lo spazio di ricerca e quindi il cammino ottimo (**rapporto di competitività**).

Gli esploratori online sono vulnerabili ai vicoli ciechi, stati da cui non è possibile raggiungere stati obiettivo. Molte azioni sono **irreversibili**, ovvero non permettono di tornare a uno stato precedente. In tal caso non è garantito il trovamento di una soluzione.

Gli spazi degli stati con azioni **reversibili** (labirinto,...) sono esplorabili in modo sicuro, ma anche in questi ambienti non può essere garantito un rapporto di competitività limitato.

Dopo ogni azione, un agente online in un ambiente osservabile riceve una percezione che gli comunica lo stato raggiunto in modo da integrare la mappa che, una volta aggiornata, viene usata per pianificare l'azione successiva. Rispetto agli algoritmi offline, quelli online esplorano il mondo reale (vengono scoperti i successori solo di un stato che si sta fisicamente esplorando), dunque sembra sensato (per evitare di espandere nodi troppo distanti da quello corrente) eseguire ricerche locali, come l'algoritmo in profondità, che espande sempre i sucessori del nodo appena espanso.

Anche la ricerca Hill Climbing gode della proprietà di località nelle espansioni dei suoi nodi. Ma questo si può bloccare in corrispondenza di massimi locali e il riavvio casuale non può essere fatto perché l'agente non può teletrasportarsi.
Una soluzione è l'uso di un **random walk**, che sceglie semplicemente a caso una delle azioni possibili nello stato corrente.
E' possibile prediligere quelle che non sono ancora state provate. Prima o poi troverà una soluzione, ma il processo può essere lento.
Arricchire l'algoritmo con memoria anziché "casualità" può portare maggiore efficienza. L'idea è di memorizzazione la stima $H(s)$ migliore corrente del costo per raggiungere l'obiettivo da ogni stato

visitato. $H(s)$ all'inizio non è altro che la stima euristica $h(s)$, ma viene aggiornata man mano che l'agente acquisisce esperienza nello spazio degli stati.

LRTA*

Aggiorna la stima del costo dello stato che ha appena lasciato e poi sceglie la mossa "apparentemente migliore" in base alle stime correnti dei costi e al costo effettivo dell'azione. Un aspetto importante è che si pensa sempre che le azioni che non sono ancora state provate, conducano immediatamente all'obiettivo con il minimo costo $h(s)$. Questo **ottimismo in condizioni di incertezza** incoraggia l'agente ad esplorare cammini nuovi e promettenti. Trova l'obiettivo se l'ambiente è finito ed esplorabile in modo sicuro. Ha costo $O(n^2)$. Non è ottimale almeno che non si usi un'euristica perfetta.