**SPAZI DI STATI E RICERCA DI SOLUZIONI**

Nella situazione più semplice assumiamo che il sistema ovvero l’agente ragioni in assenza di incertezza quindi è possibile conoscere tutto a riguardo del mondo attraverso un modello fatto a stati. Questa rappresentazione del modello sarà una rappresentazione piatta ovvero non gerarchica attraverso di un grafo. Una soluzione sarà rappresentata da un cammino nel grafo da uno stato di partenza ad uno detto goal.

L’agente sfrutta la rappresentazione del problema tramite il modello [grafo] per ragionare sul problema e computare una soluzione che in seguito una volta trovata sarà applicherà nel mondo reale.

L’idea base: per trovare una soluzione è aggiungere stati ovvero nodi al percorso e in seguito, verificare se tale stato raggiunto tramite il percorso sia goal o meno. Questo test viene effettuato con una funzione booleana.

**SPAZIO DEGLI STATI DI RICERCA**

Uno stato contiene tutte le informazioni necessarie per predire un’azione[questo significa che una volta raggiunto uno stato rappresentato come un nodo del grafo da esso sappiamo quali siano i vicini come raggiungerli e con quali costi] che sarà compiuta e per verificare se tale stato sia goal o meno. Gli stati che rappresentano un problema assieme costituiscono lo spazio degli stati di ricerca. Ogni stato avrà associato delle azioni che da esso sarà possibile compiere le quali sono dette azioni ammissibili, queste ci permettono di definire una funzione di azione la quale prende in input uno stato e un’azione e ritorna un nuovo stato. Su tale funzione dovrà essere associato un criterio che permetta di valutare le azioni compiute e conseguentemente di valutare la bontà di una soluzione in termini di ottimalità.

**RICERCA SU GRAFO**

Utilizziamo come modello per la rappresentazione del problema di ricerca un grafo orientato, il quale ci permette di astrarre il problema. [è compito dell’ingegnere della conoscenza sviluppare lo spazio di ricerca e rappresentarlo nel modo più opportuno]. Molte volte per affrontare il problema dovremo gestire un numero elevato di stati per cui servirà un modo per generare i nodi del grafo all’occorrenza e quindi in forma implicita tramite opportune regole.

Nei problemi di ricerca è necessario riuscire a riconoscere un obiettivo o stato goal e capire come i diversi stati sono tra loro connessi ovvero qual è la relazione che sussiste tra di essi.

L’assunzione che viene fatta è quella di conoscenza perfetta dello spazio di ricerca, per cui lo stato corrente nel quale si trova l’agente è completamente osservabile.

Nel grafo il nodo di partenza è detto start e il nodo o i nodi obiettivo sono detti goal. La soluzione altro non è che un percorso dal nodo di partenza a un goal con una metrica di costo associata a trovare tale percorso.

Quando l’azione giusta da compiere non è evidente, un agente può avere la necessità di guardare avanti, cioè considerare una sequenza di azioni che formano un camino che porterà a uno stato obiettivo ovvero ad un goal. Questo tipo di agente è chiamato agente risolutore di problemi e il processo computazionale che effettua è la ricerca.

GRAFI ACICLICI ORIETATI DAG E ALBERI DI RICERCA**:** risultano essere rappresentazioni ottimali per riuscire a trovare la soluzione ai nostri problemi.

MISURE DELLA COMPLESSITA’ DEI GRAFI: si utilizzano i fattori di ramificazione dei nodi in particolare:

* Uscente (forward) o ramificazione in avanti
* Entrante (backward) o ramificazione all’indietro

Quando si affronta un problema di ricerca l’agente lo divide in 4 fasi:

* Formulazione dell’obiettivo: si definisce un obiettivo, il quale permetterà di organizzare e limitare il comportamento e le azioni dell’agente nel mondo.
* Formulazione del problema: l’agente elabora una rappresentazione astratta del mondo ovvero un modello.
* Ricerca [strategia per affrontare il problema]: l’agente simula nel modello sequenze di azioni con il fine ultimo di raggiungere il suo obiettivo, se vi riesce tale sequenza di azioni prende il nome di soluzione. Questo processo con sui costruisce tali sequenze di azioni necessita di una strategia che in questo caso chiamiamo ricerca.
* Esecuzione: trovata la soluzione con il processo di ricerca può applicarla nel mondo.

Formalmente possiamo definire il problema della ricerca come:

1. Un insieme di stati nei quali si può trovare l’ambiente, tali stati descrivono l’ambiente in un dato momento e l’insieme di tali stati è detto spazio degli stati.
2. Uno stato iniziale dal quale parte l’agente.
3. Un insieme contenente un solo o più stati obiettivo.
4. Le azioni che l’agente può compiere, ovvero una funzione azione(s) che prende in input uno stato dell’ambiente e restituisce le azioni ammissibili da quello stato o che l’agente può compiere trovandosi in tale stato.
5. Un modello di transizione il quale serve a descrivere ed analizzare ciò che fa ogni azione compiuta. Banalmente è una funzione che prende in input stato in cui si trova l’agente e l’azione che da quello stato può essere compiuta risultato(s, a) = nuovo stato.
6. Funzione di costo dell’azione cost(a) che permetta di associare un numero o peso del costo di quell’azione eseguita.

In fine una soluzione come precedentemente detto è una sequenza di azioni che risulta essere un cammino dallo stato di partenza a quello goal, per cui il problema viene rappresentato come un grafo orientato dove gli stati sono i nodi e gli archi sono le azioni con associate un costo.

**ALGORITMI DI RICERCA**

L’idea di base per affrontare la ricerca su un grafo senza seguire alcuna strategia in particolare è quella di andare a esplorare percorsi andando incrementalmente a costruire una frontiera di percorsi costituita dai percorsi fino ad un dato momento conosciuti e quindi visitati. Quando all’interno di questa frontiera ci sarà un percorso con nodo finale goal allora tramite la funzione di test lo scoprirò e l’algoritmo restituirà il percorso associato come soluzione. La frontiera quindi raccoglie soluzioni parziali, i percorsi sono espansi andando a visitare un nodo adiacente all’ultimo visitato al passo precedente. [il percorso viene prima aggiunto alla frontiera e poi a posteriori è controllato per verificare se lo stato è un goal].

STRATEGIE DI RICERCA NON INFORMATE: non prendono in considerazione la posizione dell’obiettivo tra questi abbiamo:

* BFS 🡪 Ricerca in ampiezza
* DFS 🡪 Ricerca in profondità
* IDS 🡪Iterative deepening (approfondimento iterativo)
* LCFS 🡪 Ricerca con funzioni di costo minimo degli archi

Gli algoritmi di ricerca su grafo si basano sull’albero di ricerca che prevede un nodo di partenza e poi un camino aciclico verso il nodo goal. Nella costruzione di tale albero si usano i costi per trovare il cammino di costo minimo sfruttando sempre l’albero dei cammini. L’idea è quella di partire da un nodo ed espanderlo con i possibili nodi nei quali ci si può spostare queste alternative prendono il nome di FRONTIERA DELL’ALBERO ed è un insieme di nodi che può essere espanso per cercare di arrivare ad una soluzione. Il punto cruciale è stabilire chi sia il nodo che viene scelto dalla frontiera per essere espanso. Questa scelta porta a differenti strategie algoritmiche di ricerca nonché a definire una struttura dati adatta ad implementare la frontiera per gli scopi e le caratteristiche desiderate. [le strutture dati + utilizzate risultano pila, coda e coda prioritaria].

RICERCA BEST-FIRST: In questo approccio si scegli di espandere il migliore nodo, ove a stabilire chi sia il nodo migliore è una funzione di valutazione. Tale valutazione può basarsi sul costo degli archi e quindi delle azioni e decidere di selezionare dalla frontiera il nodo che abbia un costo minimo per essere raggiunto. Nell’implementazione di un algoritmo di ricerca si gestisce la frontiera con una struttura dati di tipo coda in particolare è possibile usare:

* Coda con priorità
* Coda FIFO
* Coda LIFO detta Stack o ”pila”

Gli algoritmi di ricerca a seconda che tengano traccia e quindi memoria e questo gli permette di controllare se vi sono cammini ridondanti ovvero cammini che portano allo stesso stato ma con costi diversi, si dividono in ricerca su grafo che prende in considerazione cammini ridondanti (in tal caso vedremo le strategie di pruning) e ricerca su albero ove non c’è tale controllo.

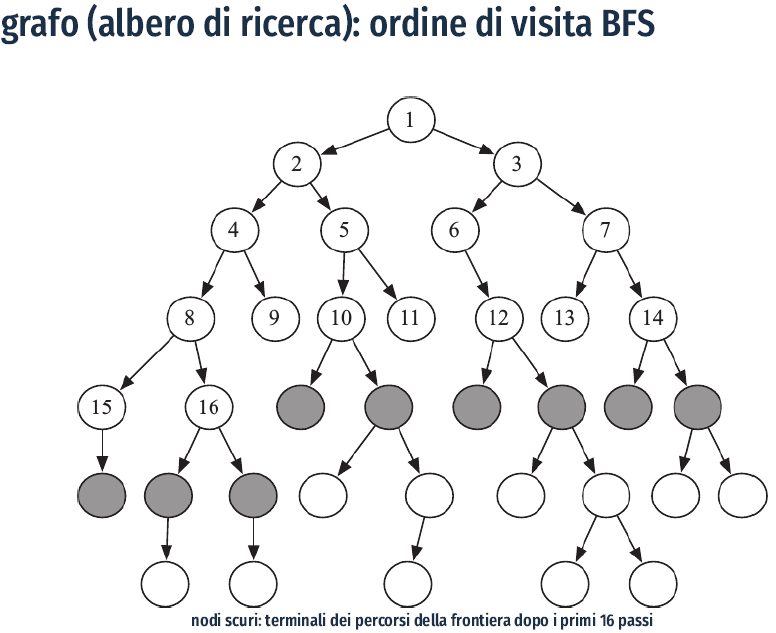
Gli algoritmi di ricerca vengono ad essere valutati tenendo in considerazione i seguenti parametri:

* Completezza: vi è garanzia di trovare la soluzione se essa esiste.
* Ottimalità rispetto al costo: trova la soluzione di costo minimo.
* Complessità temporale: valuta il tempo impiegato per trovare la soluzione
* Complessità spaziale: valuta la memoria utilizzata per trovare la soluzione.

La complessità di un algoritmo che lavora su un grafo può essere misurata in vari modi in forma esplicita dal numero di nodi e archi mentre in forma implicita con la profondità [legata alla soluzione trovata], il numero max di azioni permesse in un cammino e il branching factor ovvero il fattore di ramificazione.

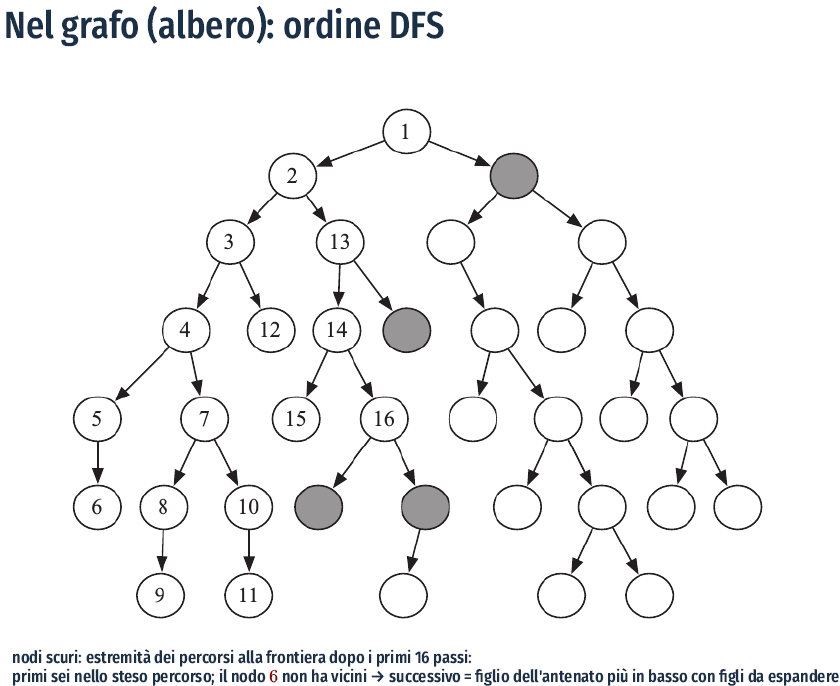
La ricerca può essere non informata oppure informata nel primo caso l’agente non ha idea raggiunto uno stato se esso sia + o – vicino allo stato goal(non ha nessuna info su dove possa essere il goal), nel secondo caso ovvero quando è informato allora l’agente sa se essendo in uno stato si sia avvicinato al goal oppure allontanato da esso.

**BFS 🡪 RICERCA IN AMPIEZZA:** detta breadth-first search BFS questo algoritmo utilizza una coda gestita in modalità FIFO per mantenere la frontiera dei percorsi; quindi, i primi ad essere estesi saranno i percorsi ad essere entrati per primi nella frontiera .Quando tutte le azioni hanno lo stesso costo la strategia appropriata è la ricerca in ampiezza che parte da un nodo ed inizia ad espandere i suoi successori e poi passa ai successori dei successori procedendo di generazione in generazione. Questo tipo di ricerca trova sempre una soluzione con numero minimo di azioni(quindi soluzioni ottime) perché quando genera i nodi di profondità d ha già generato e controllato tutti i nodi e di profondità d-1 e chiaramente avrà controllato i cammini di tale profondità. Questo fa sì che tale algoritmo sia ottimale e quindi fornisca una soluzione ottima nel qual caso tutte le azioni possibili hanno lo stesso costo. La complessità spaziale e temporale è pari a O(bd) ove b è il branching factor e d è la profondità. La complessità per cui è esponenziale. È utile usare la BFS quando il grafo non è complicato e il fattore di ramificazione non è elevato, inoltre ci garantisce il ritrovamento della soluzione ove essa esista, per cui risulta essere un algoritmo completo. Non risulta utile nel momento in cui le soluzioni sono associate a percorsi lunghi oppure è disponibile una conoscenza euristica aggiuntiva oppure quando il grafo viene generato dinamicamente poiché richiede un elevata complessità spaziale.



**RICERCA A COSTO UNIFORME “ALGORITMO DI DIJIKSTRA”:** si utilizza quando il costo delle azioni è differente, l’idea alla base è di usare un best-first con funzione di valutazione di costo minimo dalla radice al nodo corrente.

**DFS 🡪 RICERCA IN PROFONDITA’:** conosciuta come Depth-first search DFS, in questo algoritmo si utilizza una coda che viene gestita in modalità pila quindi LIFO quindi abbiamo una pila o stack; quindi, si aggiungono alla frontiera nuovi percorsi che saranno al top della pila e continuerà ad espandere tale percorso fino a quando o troverà la soluzione oppure resterà bloccato e dovrà tornare indietro per scegliere un altro percorso ovvero effettuando backtracking. La frontiera gestita come stack permette di focalizzare la ricerca su rami dell’albero di ricerca per cui il percorso in cima allo stack che sarà scelto ogni volta per essere espanso è sempre quello a profondità maggiore presente nella frontiera. È implementata come una ricerca su albero e permette di raggiungere immediatamente la profondità massima alla quale non vi sono più successori e di lì tornare in dietro alla scoperta della soluzione nel caso non si sia trovata lungo il percorso precedente. Tale tipologia di ricerca non è ottimale, infatti, tale algoritmo fornisce la prima soluzione incontrata la quale potrebbe non essere ottimale. Con questo algoritmo possono verificarsi cicli infiniti e per ovviare a tale problema si effettuano controlli per verificare la presenza di cicli. Un altro problema che ha tale algoritmo è il fatto di essere incompleto nel caso di spazi di stati infiniti, quindi potrebbe non ritornare una soluzione. Nonostante questi problemi una ricerca in profondità viene ad essere preferita ad una in ampiezza o ad una best-first poiché utilizza poca memoria. Nel caso di percorsi infiniti l’algoritmo diverge e quindi non restituisce una soluzione. La complessità nel caso pessimo può raggiungere quella esponenziale O(bd), mentre nei casi ottimi la complessità è lineare. Risulta utile in caso di limitazioni di spazio o nel caso in cui ci siano zone a densità di soluzioni nel grafo, ovvero un numero elevato di soluzioni concentrate per zone o equamente distribuite. Diventa inefficiente quando nel grafo ci sono percorsi infiniti o cicli.



**IDS 🡪 RICERCA A PROFONDITA’ LIMITATA E AD APPROFONDIMENTO ITERATIVO:** è conosciuto come iterative deepening e nasce dall’ esigenza di voler combinare i punti di forza dei precedenti algoritmi ovvero l’efficienza in spazio del DFS e l’ottimalità del BFS, l’idea per fare questo è utilizzare un BFS che ad ogni iterata lavori come un DFS sino ad una profondità limitata che ad ogni passo viene incrementata. L’algoritmo, quindi, lavora fondamentalmente come un DFS ma ad ogni passo l’algoritmo aumenta il suo spettro di ricerca con un idea simile al BFS il quale va in ampiezza e quindi andando ad approfondire incrementalmente di 1(# di archi e quindi livello di profondità) l’algoritmo e come se procedesse come un BFS. Per evitare che l’algoritmo di ricerca in profondità si perda in un cammino infinito, possiamo modificare l’algoritmo fornendo un limite massimo di profondità al quale scendere tale limite è il bound. Tale algoritmo prende il nome di ricerca a profondità limitata. Una volta fissata la profondità massima se i nodi a questa profondità avessero dei successori tali successori non sono presi in considerazione. La ricerca ad approfondimento iterativo parte da una profondità pari a 0 e va ad incrementare questa profondità di 1 ad ogni passo, fino a trovare una soluzione, si ricorda che l’incremento è considerato come # di archi. Tale algoritmo risulta ottimale quando le azioni hanno lo stesso costo(ovvero il costo degli archi è unitario per cui la metrica di costo diventa il # di archi) e non vi è presenza di cicli oppure essi sono controllati previamente. Iterative deepening può fallire in 2 modi differenti in modo innaturale quando la profondità limite non era sufficiente per trovare la soluzione e quindi, l’algoritmo fallirà momentaneamente, mentre poi il secondo modo è il fallimento naturale quando non vi è soluzione e in tal caso restituisce il bottom.

Nota: la ricerca ad approfondimento iterativo è il metodo preferito di ricerca non informata quando lo spazio degli stati in cui cercare è troppo grande per essere mantenuto in memoria e la profondità della soluzione non è nota. Inoltre, risulta essere il miglior algoritmo di ricerca non informata in termini di completezza e complessità oltre la quale non ci si abbassa senza informazioni aggiuntive.

**LCFS 🡪 RICERCA A COSTO MINIMO [lowest-Cost-First Search]:** in questo tipo di algoritmo si parte dall’informazione che gli archi abbiano costi differenti e che quindi le azioni non siano tutte eque. La BFS minimizza il numero di archi quando essi hanno costo unitario, andando a cambiare il criterio di scelta del BFS basandoci sul costo degli archi realizzeremo un lowest-Cost-First Search ove la differenza sta nella struttura dati utilizzata per la frontiera ovvero nel BFS era una semplice coda FIFO mentre qui adesso avremo una coda con priorità dove la priorità sarà decisa dal costo del percorso.

RICERCA EURISTICA E RICERCA INFORMATA

Un euristica è una funzione che associa un numero reale non negativo che aiuti nella scelta del percorso da espandere presente nella frontiera, essa costituisce una stima sul costo che servirà per arrivare dall’ultimo nodo raggiunto ad uno stato goal. La funzione euristica deve essere AMMISSIBILE ovvero deve sempre sottostimare il costo reale.

Le strategie di ricerca informata o anche detta ricerca euristica si basano su informazione aggiuntiva specifica del dominio del problema che permette di utilizzare una funzione euristica ovvero una funzione che dia un costo approssimato o stimato per raggiungere un nodo dallo stato ove l’agente si trova in un dato momento e questo costo è sempre una sottostima del reale costo tra i due nodi.

**DFS EURISTICA:** Il miglioramento portato dalla funzione euristica lo si potrà cogliere nella gestione della pila che per scegliere quale nodo posizionare in cima allo stack userà la funzione di valutazione euristica. Questo comporta che la funzione euristica permetterà di ordinare gli inserimenti nella pila dal percorso meno promettente al più promettente e in un dato momento l’algoritmo espanderà sempre il percorso più promettente che si trova sulla cima dello stack.

**GBFS 🡪 RICERCA BEST-FIRST GREEDY O GOLOSA:**  detta GBFS. In questo approccio viene ad essere espanso il nodo della frontiera che abbia il valore minimo dato dalla funzione euristica, e viene ad essere scelto tale nodo poiché risulta essere quello con maggiore probabilità di arrivare prima ad una soluzione. In tal caso la frontiera è gestita come una coda prioritaria ordinata dalla funzione euristica.

**RICERCA A\*:** Questo algoritmo cerca di combinare un lowest-cost-first con un greedy best-first search. È un tipo ricerca best-first che utilizza una funzione di costo che aggrega il costo reale fatto sino ad un dato stato raggiunto più quello stimato dal nodo raggiunto ad un possibile stato o nodo obiettivo.

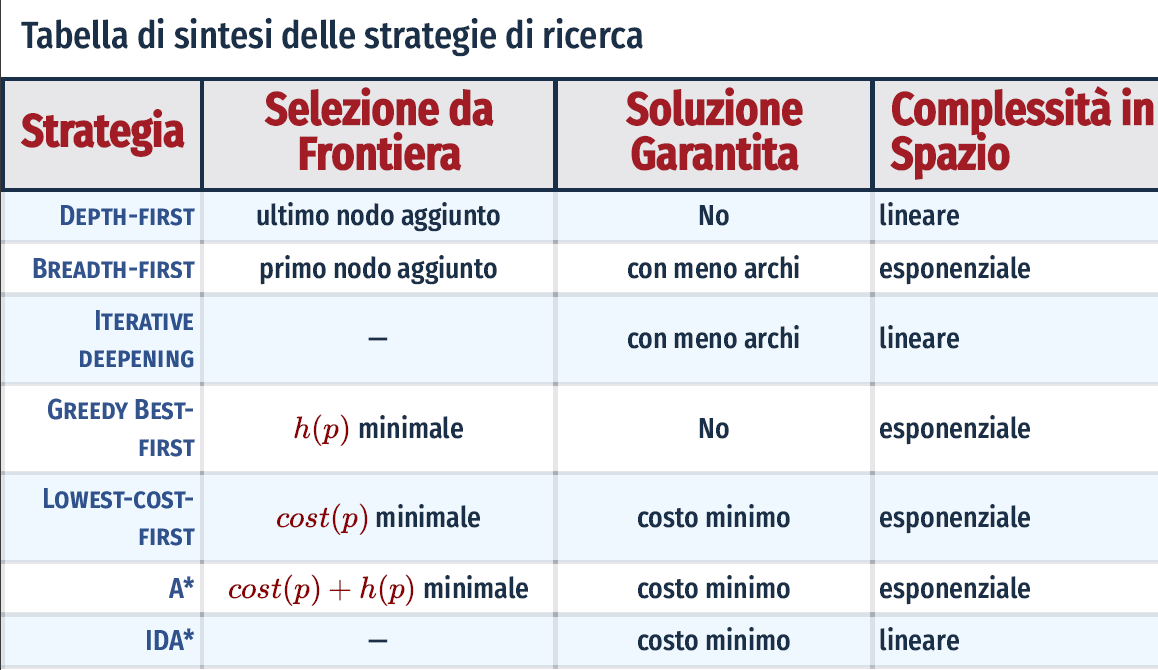
f(n) = g(n) + h(n)

Ove g(n) è il costo reale dal nodo di partenza al nodo n, mentre h(n) è la funzione euristica che stima il costo ipotetico e sottostimato da n ad un goal. La ricerca A\* è completa e risulta ottimale se è soddisfatta la proprietà di ammissibilità dell’euristica ove è ammissibile se non sovrastima mai il percorso e quindi risulta sempre e comunque inferiore o al più uguale al costo reale. Poiché sarà sempre scelta la funzione f(n) con costo minimo la prima soluzione fornita risulterà essere anche quella ottimale ove la soluzione esiste. L’algoritmo di A\* può implementare la potatura dell’albero e questo gli permette di non prendere in considerazione alcuni cammini che non risultano promettenti. La frontiera è gestita come una coda prioritaria. A\* è un algoritmo ammissibile ovvero se esistono soluzioni al problema ne troverà sempre una ottimale a patto che il fattore di ramificazione sia finito, i costi degli archi siano maggiori di un certo epsilon > 0 e la funzione euristica h() deve essere ammissibile. L’efficienza di A\* dipende dalla complessità della funzione euristica.

RICERCA IDA\*: questo algoritmo va a combinare l’iterative deepening con l’A\*, l’idea è quella di usare il DFS in modo iterativo, ma anziché utilizzare come limite la profondità che viene ad essere incrementata si utilizza un limite fissato per la funzione f() dell’A\*. l’IDA\* va quindi a lavorare sugli stessi nodi sui quali lavora l’A\*, ma li ricalcola con la DFS invece di andarli a memorizzare. Ottimizzando la complessità spaziale ovvero il consumo di memoria.

Potatura dei cicli CP detta loop pruning effettua un controllo sui cammini presenti nella frontiera in particolare il controllo verifica se nel cammino ove si va ad aggiungere un nuovo nodo tale nodo aggiunto non sia già presente, se così fosse il cammino viene scartato. In alcuni casi più di un percorso porta allo stesso nodo e quindi genera un ciclo in questo caso serve una strategia di eliminazione dei cicli che prende il nome di MPP multiple-path pruning. Questa strategia viene implementata tramite una lista di nodi che prende il nome di closed list o explored set essa inizialmente è vuota e viene popolata ogni qualvolta che prendendo un percorso il nodo che si vuole aggiungere per espanderlo non sia presente se non lo è allora tale nodo oltre ad essere aggiunto al percorso della frontiera viene anche ad essere aggiunto alla closed list.

MPP non è indicata con l’IDA\*, poiché ne peggiora la complessità dovendo memorizzare gli stati visitati cosa che l’IDA\* non esegue per migliorare l’efficienza spaziale.



[Per l’esame le domande comprenderanno funzionamento generale dell’algoritmo di ricerca, struttura dati utilizzata per gestire la frontiera e criterio di selezione del prossimo cammino della frontiera che si andrà ad espandere, la complessità in spazio e se l’algoritmo garantisce o meno la soluzione]

STRATEGIE DI RICERCA SOFISTICATE

DF-B&B 🡪 RICERCA DEPTH-FIRST BRANCH-AND-BOUND: sfrutta l’euristica e permette di risparmiare spazio. Sfrutta le strategie in profondità, l’idea è di memorizzare il costo minimo trovato verso un goal fino ad un dato momento ed impostare tale costo come limite per le successive ricerche nel caso si trovi un nuovo cammino a costo inferiore questo viene sostituito al precedente. Questo algoritmo sfrutta la conoscenza del precedente cammino minimo trovato e nei successivi passi prova partendo in maniera differente a costruire un nuovo cammino che abbia un costo inferiore a quello che ha impostato come bound. Questo algoritmo genera una sequenza di soluzioni che via via vanno a diminuire il bound. L’idea è quella di schiacciare la soluzione tra due limiti l’upper bound che viene fissato a più infinito (ma in alcuni casi ove si possiede una euristica o altre conoscenze è possibile abbassare tale bound di modo che risulti sopra la soluzione ma non di molto) e viene diminuito ogni qualvolta viene trovata una nuova soluzione fissando un nuovo upper bound e il lower bound che è dato dalla costruzione del percorso che sarà una soluzione e viene aumentato ad ogni passo iterativo di ricerca fatta in tale modo i due limiti andranno a convergere proprio sulla soluzione ottimale.

**RICERCA BIDIREZIONALE:** in questo tipo di ricerca si parte sia dallo stato iniziale cercando di arrivare verso uno stato goal e allo stesso tempo si procede da uno stato goal verso lo stato di partenza nella speranza che le 2 ricerche si incontrino nel mezzo. Il problema in questa strategia è gestire le due frontiera in modo tale che i cammini scelti dalle due frontiere si incontrino il più possibile a metà strada. L’intersezione delle 2 frontiere risulta un problema abbastanza complesso.

RICERCA BASATA SU ISOLE: permette di scomporre il problema in più livelli di astrazione e una volta risolti il problema al livello di astrazione più basilare si fa con la conoscenza guadagnata dall’aver risolto i livelli più bassi ad attaccare il problema più complesso. Le isole risultano stati sui quali si hanno delle informazioni precise come ad esempio che la soluzione dovrà passare per uno di questi stati isole, questi stati risultano un ottima euristica per far in modo che nella ricerca bidirezionale i due algoritmi di ricerca forward e quello backward si incontrino a metà strada intersecando e unendo le frontiere. Ad ogni modo anche quando la strategia di ricerca non dovesse essere bidirezionale le isole permettono di ridurre drasticamente la complessità del problema affrontando più problemi di difficoltà e complessità basica

RICERCA CON PROGRAMMAZIONE DINAMICA: l’idea alla base è quella di risolvere il problema tramite soluzioni parziali.