

Artificial Intelligence

Agenti intelligenti

- Un'agente è un'entità che percepisce l'ambiente tramite **sensori** ed agisce su di esso mediante **attuatori**
 - Funzione agente: $f : P^* \rightarrow A$ dove P^* è una sequenza percettiva, A le possibili azioni
 - Sequenza percettiva: insieme di percezioni osservate fino a quel momento
- Definizione informale di Agente razionale: cerca di raggiungere il più possibile i suoi obiettivi data l'informazione di cui dispone o che può acquisire con le sue azioni
 - Limitazioni computazionali nella ricerca dello spazio delle azioni
- Misura di prestazione: valuta una sequenza di stati dell'ambiente, non di stati dell'agente
 - Definizione formale di Agente razionale: sceglie un'azione che massimizza il valore atteso della misura di prestazione, data la sequenza di percezioni ottenuta fino all'istante corrente.
 - Razionalità !=
 - onniscienza: conoscere il risultato effettivo delle sue azioni
 - chiarezza: sequenza percettiva fino al momento corrente
 - successo atteso e non quello reale
 - Razionalità =>
 - esplorazione: intraprendere azioni mirate a modificare le percezioni future, **information gathering**
 - apprendimento: modificare la conoscenza pregressa sulla base delle proprie percezioni (scarabeo e vespa)
 - autonomia: apprendere il più possibile per compensare la presenza di conoscenza parziale o erronea (un aspirapolvere che apprende come prevedere dove apparirà lo sporco)
- PEAS (Performance Environment Actuators Sensors)
 - PEAS di un taxi automatizzato
- Tipi di ambiente
 - Completamente/parzialmente osservabile: capacità dell'agente di misurare tutti gli aspetti dell'ambiente che sono rilevanti per la scelta dell'azione
 - Deterministico/stocastico: lo stato successivo dell'ambiente è completamente determinato dallo stato corrente e dall'azione eseguita
 - Episodico/sequenziale: in ogni episodio l'agente riceve una percezione e poi esegue un'azione. Un episodio non dipende dalle azioni intraprese in quelle precedenti e ogni decisione non influenza quelle successive. (Identificare pezzi difettosi vs scacchi)
 - Statico/dinamico: l'ambiente può cambiare mentre un agente sta pensando, per cui non deve continuare ad osservare il mondo oppure non si deve preoccupare del passaggio del tempo.
 - Semidinamico se non l'ambiente non cambia ma la valutazione della prestazione sì, come negli scacchi con orologio
 - Discreto/continuo: si riferisce allo stato e all'insieme di percezioni/azioni. Gli scacchi sono discreti, la guida autonoma ha stato e tempo continui
 - Agente singolo/multiagente
 - Competitivo negli scacchi
 - Cooperativo nella guida autonoma, salvo parcheggio
 - Noto/ignoto: se l'agente conosce i risultati per tutte le azioni (gioco di carte vs videogioco)
- Tipi di agente
 - Agente con tabella sequenza percezioni x azioni: irrealizzabile per complessità spaziale
 - **Reattivo semplice**: scelgono l'azione in base solo alla percezione corrente
 - Riduce la tabella delle azioni da 4^T con T =vita a 4.
 - Regola condizione-azione (luci di frenata rosse o riflessi automatici)
 - Richiede ambiente completamente osservabile
 - **Reattivi basati su modello**: tiene traccia dell'ambiente in uno stato interno che dipende dalla storia delle percezioni e quindi riflette parte degli aspetti non osservabili dello stato corrente.
 - Richiede conoscenza sull'evoluzione del mondo indipendentemente dalle sue azioni e informazioni sull'effetto che hanno sull'ambiente le sue azioni. Questa conoscenza è implementata tramite un **modello del mondo**.
 - Poco flessibile avendo il comportamento hard-coded in regole
 - **Basati su goal**: unione dell'obiettivo al modello per la scelta dell'azione
 - Spesso richiede ricerca e pianificazione per identificare le sequenze, anche lunghe, di azioni.
 - Richiede di prendere in considerazione il futuro
 - Meno efficiente ma più flessibile, può adattare le sue azioni in base a variazioni dell'ambiente, come la pioggia, senza riscrivere le regole
 - Può anche essere modificato cambiando semplicemente l'obiettivo senza riscrivere le regole

- **Basati su una misura di utilità:** internalizzazione della misura di prestazione.
 - Gestisce casi di obiettivi in conflitto oppure non raggiungibili con certezza, in ambienti parzialmente osservabili e stocastici.
 - Calcola utilità attesa dei risultati, date le probabilità ed utilità di ciascun risultato.
- Agenti che apprendono
 - Permette di operare in ambienti inizialmente sconosciuti, migliorando la conoscenza iniziale
 - L'apprendimento può essere definito come il processo che modifica ogni suo componente affinché si accordi meglio con l'informazione di feedback disponibile, migliorando così le prestazioni globali dell'agente
- 1. **Elemento di apprendimento:** determina se e come modificare l'elemento esecutivo affinché in futuro si comporti meglio, in base all'elemento critico
- 2. **Elemento esecutivo:** prende in input le percezioni e decide le azioni
- 3. **Elemento critico:** dice a quello di apprendimento come si sta comportando rispetto a uno standard di prestazione prefissato. (Linguaggio scurrile degli altri guidatori). Può essere anche un feedback esterno come ricompensa/penalità.
- 4. **Generatore di problemi:** suggerire azioni che portino ad esperienze nuove e significative (provare freni su nuove superfici stradali)

Risoluzione di problemi

- La **formulazione dell'obiettivo**, basato sullo stato corrente e sulla misura di prestazione, è il primo passo
- Un **obiettivo** è composto dall'insieme di tutti e soli gli stati del mondo in cui l'obiettivo è soddisfatto. Il compito dell'agente è come agire, ora e nel futuro, per raggiungere uno stato obiettivo.
- La **formulazione del problema** è il processo di decidere, dato un obiettivo, *quali azioni e stati considerare*. Un agente può decidere cosa fare esaminando le azioni future che porteranno a stati di valore conosciuto.
- **Ricerca:** il processo che cerca una sequenza di azioni che raggiunge l'obiettivo.
- Un problema è composto da:
 - Lo **stato iniziale** s
 - L'insieme delle azioni che possono essere eseguite in s .
 - Un modello di transizione $RISULTATO(s, a)$, che restituisce lo stato risultante dall'esecuzione dell'azione a nello stato s .
 - Il **test-obiettivo** che determina se un particolare stato è uno stato obiettivo
 - La funzione **costo di cammino**. La soluzione ottima è quello che ha costo minore di tutte.
- Spazio degli stati: insieme di tutti gli stati raggiungibili a partire da quello iniziale mediante qualsiasi sequenza di azioni, rappresentato come grafo o **albero di ricerca**.
 - Può essere infinito come nel problema di Knuth di rappresentare un intero come operazioni di radice e fattoriale di 4
 - **RICERCA – ALBERO:** processo di espandere i nodi sulla frontiera continua finché si trova una soluzione o finché non vi sono più nodi da espandere
 - Frontiera: insieme di tutti i nodi foglia che possono essere espansi in un dato punto
 - Separa stati esplorati da quelli non esplorati.
 - Implementato come coda FIFO o LIFO
 - I cammini ciclici sono un caso particolare dei cammini ridondanti, che esistono ogni volta che esistono più modi per passare da uno stato a un altro.
 - In alcuni casi è possibile ridefinire un problema per eliminare cammini ridondanti (problema della 8 regine poste nella colonna vuota più a sinistra).
 - Non è invece possibile qualora le azioni siano reversibili.
 - Si può usare un *insieme esplorato* per ricordare i nodi espansi/visitati. Si parla di **RICERCA – GRAFO**
 - Un nodo è implementato con 4 componenti:
 - stato
 - padre
 - azione
 - costo di cammino
- Criteri di valutazione di un algoritmo di ricerca
 - **Completezza:** garantisce di trovare una soluzione quando esiste
 - **Ottimalità:** trova la soluzione ottima con costo di cammino minore
 - **Complessità temporale**
 - **Complessità spaziale**
 - Nell'informatica in genere la complessità si misura in base a $|V| + |E|$, nell'IA l'insieme di stati ed azioni è spesso infinito.
 - b *branching factor* o numero massimo di successori di un nodo
 - d *depth* del nodo obiettivo più vicino allo stato iniziale
 - m , *maximum length* dei cammini nello spazio degli stati

Ricerca non informata

- Ricerca in ampiezza (BFS)
 - Tutti i nodi a profondità minore devono essere espansi prima che si possa espondere uno dei nodi al livello successivo
 - Usa coda FIFO per la frontiera
 - Il test obiettivo è applicato alla generazione del nodo e non alla sua selezione per limitare la complessità
 - Trova sempre il cammino meno profondo
 - Completo ma non ottimo, salvo che il costo di un cammino sia una funzione monotona crescente della profondità del nodo.
 - Complessità temporale e spaziale $O(b^d)$, espande b nodi ad ogni livello e li memorizza nell'insieme esplorato
 - Ricerca a costo uniforme
 - Espande il nodo n con minimo costo di cammino $g(n)$
 - Usa coda ordinata secondo costo di cammino
 - Il test obiettivo è applicato alla selezione per espansione invece che generazione per garantire di ottimalità
 - Test aggiuntivo nel caso in cui sia trovato un cammino migliore per raggiungere un nodo sulla frontiera
 - Ottimalità:
 1. Ogni volta che un nodo è espanso, il cammino ottimale verso tale nodo è stato trovato. Altrimenti andrebbe in contraddizione con la decisione di scegliere il nodo con minore costo $g(n)$
 2. I costi non sono negativi per cui non si possono ottenere cammini migliori aggiungendo nodi
 - Il primo nodo obiettivo selezionato per l'espansione deve essere la soluzione ottima.
 - Completezza: garantita se $g(n) \geq \epsilon, \epsilon \geq 0$
 - Complessità $O(b^{1+\lceil C^*/\epsilon \rceil})$, che può essere $\gg O(b^d)$. L'intuizione è che esplora grandi alberi fatti di piccoli passi prima di considerare i cammini che prevedono passi molti grandi e forse più utili.
- Ricerca in profondità
 - Espande sempre il nodo più profondo nella frontiera
 - Utilizza una coda LIFO per la frontiera
 - Si può usare funzione ricorsiva invece di *RICERCA – GRAFO*
 - Completa con ricerca su grafo in spazi finiti, altrimenti non completa con ricerca con albero o in spazi infiniti
 - Analoghi motivi per l'ottimalità
 - Complessità temporale $O(b^m)$ con $m \gg d$ ed infinito per alberi illimitati
 - Complessità spaziale $O(b * m)$:
 - Deve memorizzare un solo cammino dalla radice a un nodo foglia a profondità m , insieme ai rimanenti $O(b)$ nodi fratelli non espansi. Un nodo espanso può essere rimosso con tutti i suoi discendenti se esplorato completamente.
 - Variante **ricerca con backtracking**: genera solo un successore quindi complessità spaziale $O(m)$, o addirittura $O(1)$ se si può generare lo stato successore tornando indietro sui propri passi e modificando direttamente la descrizione dello stato.
- Ricerca a profondità limitata l : i nodi alla profondità l sono trattati come se fossero senza successore
 - l può essere basato sulla conoscenza a priori del problema, ad esempio *diametro dello spazio degli stati*
 - Mitiga l'incompletezza della ricerca in profondità in spazi degli stati infiniti
 - Incompleto comunque se $l < d$ e non ottimo se $l > d$
 - Complessità temporale $O(b^l)$ e spaziale $O(bl)$
- Ricerca ad approfondimento iterativo
 - Il limite l viene incrementato progressivamente finché non raggiunge d
 - Complessità temporale $O(b^d)$, spaziale $O(bd)$
 - Completa con b finito
 - Ottima con funzione costo di cammino monotona crescente della profondità di nodo
 - I nodi figli diretti del nodo radice sono generati d volte, quelli più profondi meno volte
- Ricerca bidirezionale
 - Eseguire due ricerche BFS in parallelo, una in avanti dallo stato iniziale e l'altra indietro dall'obiettivo. Si spera si incontrino a metà strada. $O(b^{d/2}) + O(b^{d/2}) \ll O(b^d)$
 - Si usa un test di intersezione delle due frontiere invece che test obiettivo. Il controllo può essere in tempo costante con una tabella hash.
 - Ottimale assicurandosi che non esistano altre scorciatoie che colmino il gap
 - Complessità temporale e spaziale $O(b^{d/2})$
 - Infatti almeno una delle due frontiere deve essere mantenuta in memoria per il controllo di intersezione
 - Difficile da utilizzare con obiettivi astratti come 8 regine. Se invece ci sono più stati obiettivi espliciti, si può costruire uno stato obiettivo fittizio i cui predecessori siano tutti gli stati obiettivo.

criterio	in ampiezza	a costo uniforme	in profondità	a profondità limitata	ad approfondimento iterativo	bidirezionale
----------	----------------	---------------------	------------------	--------------------------	---------------------------------	---------------


criterio	in ampiezza	a costo uniforme	in profondità	a profondità limitata	ad approfondimento iterativo	bidirezionale
completezza	Sì	Sì	No	No	Sì	Sì
ottimalità	Sì	Sì	No	No	Sì	Sì
tempo	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
spazio	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

Ricerca informata

- Il nodo da espandere viene scelto in base ad una funzione di valutazione $f(n)$, come stima di costo
- Identico alla ricerca a costo uniforme sostituendo f con g per ordinare la coda di priorità
- f include in genere una **funzione euristica** $h(n)$, come costo stimato del cammino più conveniente dal nodo n ad uno stato obiettivo
 - Si basa solo sullo stato del nodo
- Ricerca best-first greedy
 - Espande il nodo più vicino all'obiettivo: $f(n) = h(n)$
 - Non ottimale
 - Non completa: può finire in vicoli ciechi o cicli infiniti, salvo controllo di ripetizione di stati.
 - Complessità temporale e spaziale $O(b^m)$, sebbene con una buona euristica la complessità possa essere ridotta notevolmente.
- Ricerca A*
 - $f(n) = g(n) + h(n)$, costo stimato della soluzione più conveniente che passa per n
 - Completa ed ottima se $h(n)$ è consistente
 - Ammissibilità: $h(n)$ non sbaglia mai per eccesso la stima del costo per arrivare all'obiettivo, quindi è una stima *ottimista* per natura
 - Consistenza:**
 - $h(n) \leq c(n, a, n') + h(n')$, n' successore di n tramite azione a (disuguaglianza triangolare)
 - condizione più forte dell'ammissibilità.
 - Ottimalità per ricerca ad albero: se un nodo obiettivo non ottimale G_2 viene generato e si ha n nodo sul cammino minimo verso il nodo obiettivo ottimale G , A* sceglierà n : $f(G_2) = g(G_2) \geq f(G) \geq f(n)$
 - Non vale per ricerca a grafo, rischia di scartare un nodo già visitato che si trova però sul cammino ottimo
 - Ottimalità per ricerca a grafo:
 - $f(n)$ è crescente lungo ogni cammino: $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$
 - Il cammino trovato per il nodo n selezionato per l'espansione è ottimo.
 - Altrimenti esisterebbe un altro n' lungo il cammino verso n , che avrebbe però avuto un $f(n') \leq f(n)$ e quindi sarebbe stato selezionato prima
 - Il primo nodo obiettivo selezionato deve quindi essere ottimo, perché ho trovato il cammino minimo verso quel nodo e tutti i nodi obiettivo successivi avranno un costo \geq
 - A* espande gradualmente il contour in ordine di f : contiene tutti i nodi che hanno $f_i \leq f_{i+1}$
 - Con ricerca a costo uniforme $h(n) = 0$ le bande sono circolari, con l'euristica $h(n)$ sono allungate verso lo stato obiettivo e si stringono intorno al cammino ottimo.
 - A* espande tutti i nodi con $f(n) < C^*$ ed alcuni con $f(n) = C^*$ con C^* costo del cammino ottimale
 - La completezza richiede che esista un numero finito di nodi $f(n) \leq C^*$, vera se $f(n) \geq \epsilon$ e b è finito
 - A* non espande alcun nodo $f(n) > C^*$, i relativi alberi sono *pruned*
 - A* è ottimamente efficiente, non è possibile espandere meno nodi
 - La complessità temporale è $O(b^{ed})$ ove $\epsilon = (h^* - h)/h^*$ (errore relativo) dove h^* è il costo effettivo del cammino dalla radice all'obiettivo
 - La complessità spaziale è $O(b^{ed})$, non applicabile per problemi di grandi dimensioni
- Iterative Deepening A* (IDA*)
 - Invece di tagliare a profondità d , si taglia a costo f , ove il limite è il f -costo minimo tra quelli di tutti i nodi che hanno superato il limite nell'iterazione precedente
- Recursive Best-First Search (RBFS):
 - come ricerca ricorsiva in profondità, con spazio lineare
 - Tiene traccia di f_limite , come f -costo del miglior cammino alternativo di uno degli antenati del nodo corrente
 - Quando il nodo corrente supera il limite, torna indietro a quello alternativo e sostituisce l' f -valore di ogni nodo lungo il ritorno con un valore di backup, il migliore dei suoi nodi figli.

- Eccessiva rigenerazione dei nodi: in uno spazio di ricerca grande e vicini all'obiettivo, il cammino alternativo tende a diventare quello migliore in assoluto, ma poi la ricerca deve tornare di nuovo indietro sul cammino ottimale per rieseguirlo. Ogni "ripensamento" corrisponde ad un'iterazione di IDA* e ne può richiedere diversi per estendere il cammino migliore di un nodo.
- Ottima se $h(n)$ è ammissibile
- Complessità spaziale $O(bd)$, temporale difficile da definire ma esponenziale nel caso pessimo
- IDA* e RBFS usano troppa poca memoria, dimenticando ciò che hanno fatto riesplorano nuovamente gli stessi stati più volte.
- Memory-bounded A* (MA*) o Simplified MA* (SMA*)
 - Espande la foglia migliore finché la memoria non è piena. A quel punto scarta la foglia peggiore, con f -valore più alto, memorizzando nel nodo padre il valore del nodo scartato.
 - Rigenera il sottoalbero dimenticato solo quando *tutti gli altri cammini* sono peggiori.
 - Se f -valore è uguale, espande la foglia più recente e rimuove quella più vecchia. Se c'è una sola foglia la rimuove, perché in ogni caso non ci sarebbe abbastanza memoria.
 - Completo se il cammino fino a d è raggiungibile con la memoria disponibile
 - Ottima se c'è la soluzione tra quelle raggiungibili, altrimenti restituisce la migliore tra quelle disponibili
 - In problemi molto difficili, SMA* passa continuamente da un cammino candidato all'altro e potrà tenere in memoria solo un piccolo sottoinsieme. Dovrà rigenerare ripetutamente i nodi.
 - Le limitazioni di memoria possono rendere temporalmente intrattabile un problema altrimenti risolvibile in A*
- Euristiche
 - 8-puzzle
 - $h_1(n)$ = numero di tasselli fuori posto
 - $h_2(n)$ = somma della distanza Manhattan, distanza in orizzontale e verticale per ogni tassello dalla posizione obiettivo
 - TSP: MST come euristica, limite inferiore al percorso aperto più breve
 - La qualità dell'euristica si misura con b^* , fattore di branching effettivo indicato come $N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$
 - Il valore ideale di b è vicino a 1, in modo che i tempi siano ragionevoli anche per problemi di dimensioni grandi. Già 2 significa esplorare tutti i nodi di un albero binario.
 - Se un'euristica h_2 domina su h_1 si traduce in maggior efficienza perché ci saranno meno nodi espansi: $f(n) < C^* \Leftrightarrow h(n) < C^* - g(n)$
 - Il costo di una soluzione ottima di un problema rilassato è un'euristica ammissibile per il problema originale
 - Un problema rilassato ha meno vincoli sulle azioni possibili
 - Il grafico del problema rilassato corrisponde ad un *supergrafo* dell'originale, con più archi quindi potenziali scorciatoie verso la soluzione ottima
 - Se invece il problema rilassato è ancora difficile da risolvere, i valori dell'euristica saranno difficili da ottenere
 - Altrimenti potremmo sempre usare la ricerca in ampiezza come euristica "perfetta"
 - Si può derivare euristiche ammissibili dal costo della soluzione di un sottoproblema (i.e. 4 tasselli)
 - Database di pattern: memorizzare i costi delle soluzioni dei sottoproblemi (programmazione dinamica)

Algoritmi di miglioramento iterativo / Ricerca locale

- Il cammino non è rilevante, conta trovare una configurazione ottima nello spazio degli stati dato dall'insieme delle configurazioni a stato completo (es 8 regine)
-  Si mantiene un singolo stato corrente e si tenta di migliorarlo iterativamente
- Complessità spaziale costante, quindi adatto per la ricerca online, e possono trovare soluzioni ragionevoli in spazi degli stati grandi o infiniti
- Il panorama dello spazio degli stati mostra una posizione, definita dallo stato, e un'altezza che corrisponde al valore della funzione costo o di obiettivo. Rispettivamente si punta a trovare il minimo e massimo globali.
- Completo se trova sempre un obiettivo, ottimale se trova il minimo/massimo globale
- **Hill climbing**
 - Ciclo che si muove continuamente verso l'alto, nella direzione dei valori crescenti e termina quando raggiunge un picco che non ha valore vicino più alto
 - I.e. 8 regine: stati con 8 regine e i possibili stati muovono una singola regina. h è il numero di coppie di regine che si stanno attaccando a vicenda
 - $p = 14\%$ delle volte soluzione ottima
 - Procede molto rapidamente verso una soluzione, ma spesso rimane bloccato:
 - **massimo locali**: picco più alto degli stati vicini, ma inferiore al massimo globale
 - **creste**: sequenza di massimi locali molto difficili da esplorare
 - **plateau**: area piatta nello spazio degli stati, può essere un massimo locale piatto o una *spalla* da cui si potrà salire ulteriormente

- Utilizzo di una *mossa laterale* per spostarsi nel plateau. Bisogna porre limite massimo di mosse consecutive per evitare ciclo infinito.
 - $p = 94\%$ delle volte soluzione ottima
- Hill climbing stocastico: sceglie a caso tra le mosse che vanno verso l'alto
- Hill climbing con riavvio casuale: serie di ricerche hill climbing con stati iniziali generati casualmente
- Se la probabilità di ricerca è p , sono richiesti $1/p$ riavvi in media

$$x_i = \begin{cases} 0 & \text{se la } i\text{-esima ricerca non trova la soluzione ottima} \\ 1 & \text{se la } i\text{-esima ricerca trova la soluzione ottima} \end{cases}$$

$$\forall i, P(x_i = 1) = p \text{ e } P(x_i = 0) = 1 - p$$

La probabilità che la k -esima ricerca sia la prima con soluzione ottima è quindi: $P(\sum_{j=1}^{k-1} x_j = 0 \wedge (x_k = 1)) = (1 - p)^{k-1} p$

Il valore atteso quindi è: $\sum_{k=1}^{\infty} k(1 - p)^{k-1} p = \frac{1}{p}$
- 8 regine:
 - Con $p = .14$ sono richieste in media $1/.14 = 7$ ricerche
 - Con mosse laterali sono richieste in media $1/.94 = 1.06$ ricerca per la soluzione ottima
- La bontà della ricerca dipende dal panorama dello spazio degli stati, soprattutto dalla presenza di massimi locali o plateau. Spesso purtroppo gli spazi degli stati hanno forma a "porcospino".
- Simulated annealing
 - Combinazione del hill climbing con esplorazione casuale partendo da mosse "cattive"
 - Prende il nome dalla tecnica di raffreddamento del metallo, che viene portato ad alta temperatura e poi raffreddato lentamente ottenendo lo stato con minor configurazione energetica
 - Si usa discesa di gradiente, quindi minimizzazione di costo
 - Scelta casuale della mossa che migliora, con probabilità pesata in base al grado di miglioramento ΔE e alla temperatura T . Quest'ultima decresce gradualmente, quindi all'inizio ci sono maggiori probabilità di scegliere mosse "cattive"
 - Se la temperatura decresce abbastanza lentamente, trova ottimo globale con probabilità tendente a 1
 - $p(x) = e^{\frac{E(x)}{kT}}$, k è la costante che regola la temperatura
 - Usato per configurazione VLSI

Ricerca online

- Ricerca offline: la soluzione completa viene calcolata e poi eseguita
- Ricerca online: l'agente opera alternando computazione ed azione, prima esegue un'azione, poi osserva l'ambiente e determina l'azione successiva
 - Si presta a domini dinamici e in ambienti ignoti, stati ed effetti delle azioni sono sconosciuti all'agente
 - Adatta a problemi di esplorazione
- Problema di ricerca online
 - L'agente conosce solo $AZIONI(s)$, $c(s, a, s')$ e $TEST - OBIETTIVO(s)$
 - L'agente non può determinare $RISULTATO(s, a)$ se non trovandosi effettivamente in s ed eseguendo a
 - La funzione di costo $c(s, a, s')$ non può essere usata finché l'agente non appura che s' è il risultato dell'azione
 - Rapporto di competitività rispetto ad un algoritmo che conosce lo spazio di ricerca
 - Può essere infinito nel caso di azioni irreversibili e **vicoli ciechi**
 - Può essere non limitato anche nel caso di spazio degli stati *esplorabile in modo sicuro* e con azioni reversibili, immaginando un avversario che blocca il cammino migliore con un lungo muro sottile
- **AGENTE – ONLINE – RIP** (ricerca in profondità)
 - Un agente online può espandere solo nodi fisicamente successori di quello che sta occupando, quindi per evitare di spostarsi continuamente è meglio espandere secondo un ordine locale: ricerca in profondità.
 - L'agente mantiene in memoria gli stati risultanti delle azioni $RISULTATO(s, a)$ e prova ogni azione non ancora esplorata
 - Se tutte le azioni sono state provate in uno stato, torna indietro con backtracking (richiede quindi azioni reversibili)

- Nel caso peggiore attraversa lo spazio degli stati due volte
 - Ottimo per l'esplorazione
 - Non ottimo, per il raggiungimento del goal, esplorare lunghe distanze quando si è vicini all'obiettivo
 - Variante ad approfondimento iterativo
- Ricerca hill climbing
 - È già una ricerca online in quanto mantiene solo lo stato corrente
 - Non è utile perché si può bloccare a massimo locale e non si può usare riavvio casuale perché l'agente non può trasferirsi in un altro stato magicamente
 - **Random walk**: sceglie a caso una delle possibili azioni, prediligendo quelle non ancora provate
 - Converge *prima o poi* all'obiettivo ma può essere molto lento, soprattutto se ci sono alte probabilità di tornare indietro
 - Trick LRTA*: arricchire con memoria, memorizzando la miglior stima corrente $H(s)$ ad ogni stato s , che inizialmente è l'euristica $h(s)$. Man mano con l'esperienza nello spazio degli stati, il valore viene aggiornato per "sfuggire" ad esempio da minimi locali "appiattendoli"
 - Mantiene memoria dei risultati delle azioni come *AGENTE – ONLINE – RIP*
 - Aggiorna la stima del costo $H(s)$ dello stato appena lasciato, $H(s) = c(s, a, s') + H(s')$
 - Ottimismo in condizioni di incertezza: si privilegiano le azioni non ancora provate in uno stato, poiché si pensa conducano all'obiettivo con il minimo costo possibile $h(s)$.
 - Completo in spazi finiti, non completo in spazi degli stati infiniti poiché può lasciarsi sviare senza possibilità di ritorno
 - Complessità temporale $O(n^2)$ con n stati, nel caso peggiore

Ricerca con avversari

- Ambienti multiagente competitivi, bisogna considerare le azioni degli altri e gli obiettivi degli agenti sono in conflitto
- **Giochi a somma zero con informazione perfetta**: ambienti deterministici e completamente osservabili, in cui due agenti agiscono alternandosi e i cui valori di unità, a fine partita, sono uguali ma di segno opposto (a somma zero).
- I giochi sono facili da rappresentare in astratto ma difficili da risolvere
 - Scacchi hanno fattore di ramificazione 35
 - Richiedono abilità di prendere una decisione quando calcolare quella ottima non è realizzabile per inefficienza
 - L'obiettivo è sfruttare al meglio il tempo disponibile
- Un gioco è composto dalle seguenti componenti:
 - S_0 stato iniziale
 - $GIOCATORE(s)$: a chi tocca tra MIN e MAX
 - $AZIONI(s)$
 - $RISULTATO(s, a)$
 - $TEST - TERMINAZIONE(s)$
 - $UTILITÀ(s, p)$ o funzione di payoff: attribuisce il valore numerico finale nello stato s al giocatore p . Un gioco a somma zero ha payoff totale 0 ed uguale per tutte le istanze del gioco
- Stato iniziale, le azioni ed i possibili risultati compongono l'**albero di gioco**
 - Costrutto teorico che non possiamo realizzare nel mondo fisico perché troppo grande con giochi non banali
 - Si usa **albero di ricerca**, porzione dell'albero di gioco esaminato per consentire di determinare quale mossa fare
- A differenza della ricerca normale, in un gioco con avversari MIN può dire la sua: MAX deve elaborare una strategia che valuti la propria mossa, gli stati possibili dalle mosse di MIN , le proprie mosse risultati e così via
- La strategia ottima esamina il **valore minimax** di ogni nodo: miglior utilità per MAX contro un avversario che gioca in modo ottimo
 - \$

$$\text{MINIMAX}(n) = \begin{cases} \text{UTILITÀ}(s, MAX) & \text{se } TEST-TERMINALE(s) \\ \max_{a \in AZIONI} \{ \text{VALORE-MINIMAX-RISULTATO}(s, a) \} & \text{se } GIOCATORE(s) = MAX \\ \min_{a \in AZIONI} \{ \text{VALORE-MINIMAX-RISULTATO}(s, a) \} & \text{se } GIOCATORE(s) = MIN \end{cases}$$
 \$
- Algoritmo minimax
 - Calcola ricorsivamente il valore minimax di ogni stato successore fino alle foglie, di cui ottiene $UTILITÀ$ poiché terminali
 - I valori minimax sono portati su nella fase di ritorno: prende il minimo nei livelli in cui tocca a MIN , il massimo quando tocca a MAX
 - **Esegue esplorazione completa in profondità dell'albero di gioco**
 - Complessità temporale $O(b^m)$ e spaziale $O(bm)$
 - Completo con alberi di gioco finiti
 - Ottimo contro avversari ottimali, altrimenti ancora meglio perché il minimax è il massimo del minimo vantaggio ottenibile

- Multiplayer A, B, C: ogni nodo ha un vettore $\langle v_A, v_B, v_C \rangle$
 - Stati terminali: il vettore è riempito con **UTILITÀ**
 - Stati intermedi: il vettore di utilità passato in alto più a favore di chi sta scegliendo la mossa
 - Alleanze: conseguenza naturale delle strategie ottime di ogni giocatore
- **Potatura alfa-beta**: pota i rami che non possono influenzare la decisione finale
 - Ogni nodo ha un intervallo dei possibili valori minimax dei successori
 - α il valore migliore, più alto, per *MAX*
 - β il valore migliore, più basso, per *MIN*
 - $\text{MINIMAX}(\text{radice}) = \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) = 3$ a prescindere da x e y che quindi possono essere potati
 - Possiamo potare un nodo appena abbiamo raccolto abbastanza informazioni per concludere che esiste una scelta sicuramente migliore in un nodo precedente
 - L'efficacia della potatura dipende fortemente dall'ordine con cui si esaminano gli stati
 - Per l'ordine perfetto serve poter:
 - conoscere il valore esatto di uno stato:
 - conoscere il valore esatto di utilità per uno figlio
 - conoscere un bound sulla utilità di tutti gli altri figli
 - $E(d+1) = E(d) + (b-1)B(d)$
 - conoscere un bound sulla utilità di uno stato:
 - conoscere il valore esatto di utilità per uno stato figlio
 - $B(d+1) = E(d)$
 - $E(d) / B(d)$ numero minimo di stati da considerare per conoscere il (valore esatto) / bound di utilità di uno stato a profondità d dagli stati terminali
 - $E(m) \leq (\sqrt{2}b)^m = (\sqrt{2})^m b^{m/2}$ upper-bound
 - Complessità temporale $O(b^{m/2})$ se sono esaminati prima i successori più promettenti tramite ordinamento perfetto
 - Il fattore di ramificazione effettivo diventa $b^* = \sqrt{b}$
 - Scacchi con ordinamento semplice che cerca prima di catturare i pezzi, poi di minacciarli, poi mosse in avanti e infine quelle indietro: $\sqrt{35} = 6$
 - Complessità $O(b^{3m/4})$ con ordine casuale
 - Si può usare ricerca ad approfondimento iterativo per avere informazioni sull'ordinamento delle mosse
 - Utile usare **tabella delle trasposizioni**: hashmap che memorizza le valutazioni delle trasposizioni, permutazioni diverse della stessa sequenza di mosse che portano alla stessa configurazione
 - Negli scacchi raddoppia la profondità raggiungibile
- **Funzioni di valutazione**
 - Limiti alle risorse di tempo per calcolare una mossa
 - Tagliare la ricerca minimax o alfa-beta prima che raggiunga una foglia, usando una funzione di valutazione euristica **EVAL** che stima l'utilità della posizione raggiunta
 - **TEST-TAGLIO** decide quando applicare **EVAL**
 - **EVAL** fornisce una stima del guadagno atteso in uno stato, analogo all'euristica distanza dall'obiettivo
 - È quello che fanno i giocatori di scacchi
 - Il comportamento corretto è preservato per trasformazioni monotone: conta mantenere l'ordine.
 1. Deve ordinare gli stati terminali nello stesso modo della funzione **UTILITÀ**, per evitare che si sbagli quando l'agente è capace di "vedere" fino alla fine della partita
 2. Deve essere efficiente
 3. Per gli stati non terminali deve avere una forte correlazione con la probabilità di vincere
 - In genere è in base alle **caratteristiche** dello stato, che prese insieme definiscono *categorie o classi di equivalenza*: gli stati di una categoria hanno lo stesso valore per le caratteristiche (i.e. "due pedoni vs uno")
 - Il valore di ogni categoria può essere stimato dal valore atteso pesando le % di vittoria, pareggio e sconfitta con i rispettivi valori di utilità. Nella pratica richiede troppe categorie.
 - Si preferisce invece l'insieme di valori delle caratteristiche sommato in maniera pesata come **funzione lineare pesata**: $\text{EVAL}(s) = \sum_i^n w_i f_i(s)$
 - Si possono usare anche combinazioni non lineari per includere dipendenze tra caratteristiche (i.e. alfiere con *numero di mosse* alto)
 - I pesi w_i possono essere stimati con Machine Learning
 - **TEST-TAGLIO**(s, d)
 - Può usare un limite di profondità, ma va scelto affinché la scelta della mossa avvenga nel tempo allocato
 - Più robusto usare ricerca ad approfondimento iterativo
 - Restituisce la mossa calcolata con la più profonda ricerca completata in tempo
 - Aiuta anche l'ordinamento delle mosse
 - Rischio di errore causa approssimazione

- Meglio tagliare e valutare in posizioni quiescenti tramite **ricerca di quiescenza**, ad esempio posizioni senza catture
- Potatura in avanti: in un dato nodo alcune mosse saranno potate immediatamente senza altra considerazione
 - Taglio della ricerca e la potatura alfa-beta non influenzano il risultato finale, quella in avanti potrebbe.
 - PROBCUT: utilizza statistiche da esperienza precedente per potare mosse *probabilmente* fuori dall'intervallo alfa-beta.
 1. Esegue ricerca poco profonda per calcolare il valore minimax v portato su di un nodo
 2. Stima la probabilità che il valore v a profondità d sia fuori da (α, β)
- Giochi stocastici
 - Backgammon: vengono tirati due dadi per determinate possibili mosse, 21 casi distinti
 - L'albero di gioco deve includere **nodi di casualità** i cui archi uscenti rappresentano i diversi esiti e relativa probabilità
 - **valore expectiminimax**: i nodi casualità hanno valore atteso minimax, come somma pesata dei minimax in base alla probabilità della mossa $\sum_r P(r) \cdot \text{EXPECTIMINIMAX}(\text{RISULTATO}(s, r))$ con r risultato del tiro di dadi
 - La funzione di valutazione deve essere una **trasformazione lineare positiva** della probabilità di vincere, altrimenti valutazioni che preservano l'ordinamento comunque modificano la scelta della mossa migliore
 - Complessità temporale $O(b^m n^m)$ con n numero di tiri di dado distinti
 - Quando entra in gioco l'incertezza le possibilità si moltiplicano enormemente e diventa inutile formulare piani dettagliati
 - Se limitiamo i valori della funzione **UTILITÀ** possiamo derivare dei limiti anche per la media dei nodi di casualità e di conseguenza potarne alcuni
- Giochi ad informazione parziale e stocastici (i.e. carte)
 - Approccio naive: come se tutti i dadi fossero stati tirati all'inizio della partita
 - Considerare tutte le possibili distribuzioni di carte nascoste e risolverle una per una come fosse un gioco completamente osservabile. Scegliere poi il miglior risultato pesato sulle probabilità $P(s)$ delle distribuzioni. $\text{argmax}_a \sum_s P(s) \text{MINIMAX}(\text{RISULTATO}(s, a))$ o H-MINIMAX se non è possibile
 - $\binom{26}{13}$ possibili distribuzioni, impossibile risolvere per tutte
 - Approssimazione Monte carlo: prendiamo casualmente N distribuzioni, ognuna con probabilità di essere pescata proporzionale a $P(s)$: $\text{argmax}_a \frac{1}{N} \sum_i \text{MINIMAX}(\text{RISULTATO}(s_i, a))$
 - Con $N = [100, 1000]$ fornisce una buona approssimazione
 - La strategia viene chiamata *media sulla chiaroveggenza*: ipotizza che il gioco diventerà osservabile per tutti dopo la prima mossa. Assunzione falsa
 - Racconto della strada A con montagna d'oro e strada B con bivio, montagna d'oro più grande o autobus
 - Con informazione non completa, il valore di un'azione dipende dallo stato di credenza in cui si trova l'agente
 - Non sceglie mai azioni che ottengano informazioni o le nascondano all'avversario, assume anzi che si sappiano già.

Constraint Satisfaction Problem

- Finora lo stato era atomico, scatola nera priva di struttura atomica
 - In CSP si usa una rappresentazione fattorizzata, una serie di variabili con vincoli
 - L'idea è eliminare ampie porzioni dello spazio di ricerca, individuando combinazioni di variabili e valori che violano i vincoli
 - Più efficiente di una ricerca classica sull'intero spazio degli stati
- Un CSP è composto da:
 - X insieme di variabili $\{X_1, \dots, X_n\}$
 - D insieme di domini delle variabili $\{D_1, \dots, D_n\}$ ove ogni dominio contiene i valori ammessi $\{v_1, \dots, v_n\}$
 - Le variabili possono avere domini discreti e finiti o infiniti, oppure domini continui (programmazione lineare). Per domini discreti e infiniti serve linguaggio di vincoli $J_1 + d \leq J_2$
 - C è un insieme di vincoli, ogni vincolo è costituito da $\langle \text{ambito}, \text{relazione} \rangle$, ambito è una tupla di variabili che partecipano al vincolo e relazione definisce i valori che possono assumere.
 - I vincoli possono essere lineari (risolvibili) o non lineari (non risolvibili da alcun algoritmo)
 - I vincoli possono essere unari, binari, di ordine superiore (Sudoku) o globali a seconda del numero di variabili interessate
 - Ogni relazione supporta due operazioni:
 1. Appartenenza alla relazione
 2. Enumerazione dei membri stessi
- Assegnamento: dare valori alle variabili
 - Completo/parziale se tutte le variabili hanno un valore o meno
 - Consistente se ogni assegnamento non viola alcun vincolo
 - **Soluzione se completo e consistente**
- Si può visualizzare un CSP come **grafo di vincoli**, i nodi corrispondono alle variabili del problema e un arco connette ogni coppia di variabili che partecipano ad un vincolo
 - Ogni vincolo a dominio finito può essere ridotto ad un insieme di vincoli binari usando variabili ausiliarie. Si ottiene il grafo duale.

- **Propagazione dei vincoli:** utilizzare i vincoli per ridurre il numero di valori legali per una variabile, che a sua volta può ridurre i valori legali per un'altra variabile e così via.
 - Il concetto è forzare la **consistenza locale**, eliminando i valori inconsistenti dal grafo
 - Consistenza di nodo: tutti i valori del dominio soddisfano i vincoli unari del nodo (nodo-consistente)
 - Consistenza d'arco: ogni valore del dominio di un nodo soddisfa i vincoli binari in cui partecipa (arco-consistente)
 - X_i è arco-consistente rispetto a X_j se $\forall v \in D_i, \exists w \in D_j$ che soddisfa il vincolo binario sull'arco (X_i, X_j)
 - Algoritmo AC-3:
 1. Mantiene un insieme degli archi
 2. Estrae un arco (X_i, X_j) e rende X_i arco-consistente rispetto a X_j
 3. Se il dominio D_i è rimasto invariato si passa all'arco successivo, altrimenti si aggiungono alla coda tutti gli archi (X_k, X_i) con X_k vicino di X_j
 4. Se D_i è vuoto allora il CSP non ha soluzione e AC-3 può ritornare fallimento
 5. Otteniamo altrimenti un CSP equivalente all'originale ma con future ricerche più rapide perché le variabili hanno domini più piccoli
 - Complessità $O(cd^3)$, c numero di vincoli binari, d dimensione massima di un dominio. Ogni vincolo richiede al più $O(d^2)$ controlli e può essere raggiunto al più d volte
 - A volte inutile perché ogni variabile è già arco-consistente
 - Consistenza di cammino: restringe i vincoli binari usando vincoli *impliciti* inferiti considerando triplette di variabili
 - Due variabili $\{X_i, X_j\}$ sono cammino-consistenti rispetto ad una terza variabile X_m se per ogni assegnamento $\{X_i = a, X_j = b\}$ consistente con i vincoli $\{X_i, X_j\}$ esiste un assegnamento di X_m che soddisfa i vincoli su $\{X_i, X_m\}$ e $\{X_m, X_j\}$.
 - Si considera essenzialmente un cammino da X_i a X_j con X_m nel mezzo
 - Algoritmo PC-2
 - k -consistenza: per ogni insieme di $k - 1$ variabili e loro assegnamento consistente, è sempre possibile assegnare un valore consistente a ogni k -esima variabile
 - Fortemente k -consistente se è $k - 1$ consistente, $k - 2$ consistente etc. fino a 1-consistente
 - Vincoli globali
 - *Tuttediverse*: se il numero di variabili m è $> n$ numero di possibili valori distinti, allora il vincolo non può essere soddisfatto
 - Semplice algoritmo: per ogni variabile del vincolo con dominio con un solo valore, rimuoviamo il valore dai domini delle altre variabili. Se un dominio rimane vuoto o $m > n$ allora c'è un'inconsistenza
 - Vincolo delle risorse (*atmost*(10, P_1, P_2, P_3, P_4)): controllare se la somma dei valori minimi dei domini > 10
 - Propagazione degli estremi per ottenere variabili con estremi consistenti:
 - $D_1 = [0, 165], D_2 = [0, 385]$
 - $F_1 + F_2 = 420$
 - $D_1 = [35, 165], D_2 = [255, 385]$
 - Sudoku: CSP con 81 variabili, 27 vincoli *Tuttediverse*
 - Per gli schemi più facile si può applicare algoritmo AC-3
 - Naked triples: trovare tre caselle in un'unità riga, colonna o riquadro, che abbiano lo stesso dominio. Possiamo rimuovere allora il dominio dalle altre caselle
- Ricerca con backtracking: essenzialmente ricerca depth-first per CSP con assegnamento di singole variabili
 - Applicare ricerca a profondità limitata avrebbe complessità temporale $O(n!d^n)$ con n variabili con al massimo d valori nel dominio
 - I CSP sono **commutativi**: qualsiasi ordine di assegnamento alle variabili produce lo stesso assegnamento parziale indipendentemente
 - Basta considerare una sola variabile in ogni livello dell'albero di ricerca
 - Il numero di foglie si restringe a d^n
 - La ricerca con backtracking assegna valori ad una variabile per volta e torna indietro quando non ci sono più valori legali da assegnare
 1. Sceglie man mano una variabile non assegnata
 2. Prova uno ad uno tutti i valori del suo dominio
 - Se viene rilevata un'inconsistenza si ritorna alla chiamata precedente che prova un altro valore
 - Miglioramenti:
 - Ordinamento delle variabili
 - Euristiche MRV (Minimum Remaining Values): scegliere la variabili con il minor numero di valori legali.
 - Chiamata anche variabile più vincolata o fail-first: con maggior probabilità di arrivare ad un fallimento, potendo così l'albero di ricerca
 - Ad esempio se una variabile non ha più valori sarà scelta subito, rilevando il fallimento
 - Miglioramento di fattore 1000

- Euristiche di grado: cerca di contenere il fattore di ramificazione scegliendo la variabile coinvolta nel maggior numero di vincoli con le altre variabili non assegnate
 - Da usare in grado di pareggio dell'euristica MRV
 - Euristiche valore meno vincolante: predilige il valore che lascia più flessibilità alle variabili adiacenti sul grafo dei vincoli
- Alternanza di ricerca e inferenza
 - Invece che applicare AC-3 prima di iniziare la ricerca, possiamo farlo ogni volta che scegliamo un valore per una variabile
 - Verifica in avanti (**forward checking**)
 1. Ogni volta che una variabile X è assegnata, si stabilisce la consistenza d'arco per ogni variabile non assegnata Y collegata a X da un vincolo
 2. Si cancella dal dominio di Y ogni valore non consistente con quello scelto per X
 - Si può usare in combinazione con l'euristica MRV come modo efficiente per calcolarne le informazioni
 - Problema: non guarda avanti per rendere arco-consistenti le variabili non collegate a X
 - MAC (Maintaining Arc Consistency): richiama AC-3 per solo gli archi (X_i, X_j) per le X_j non assegnate adiacenti a X_i e **si propaga in avanti** ricorsivamente quando si apportano modifiche ai domini delle variabili
- **Backjumping**: backtracking ad una delle variabili che ha causato il fallimento
 - Invece di fare backtracking cronologico al punto decisionale più recente
 - L'insieme dei conflitti per una variabile X è l'insieme delle variabili precedentemente assegnate che sono collegate a X da vincoli.
 - Il forward checking può fornire l'insieme dei conflitti senza lavoro aggiuntivo
 - Ogni volta che si cancella un valore dal dominio di Y a causa di un assegnamento in X , si aggiunge X all'insieme di conflitti di Y
 - Se viene cancellato l'ultimo valore dal dominio di Y , tutti gli assegnamenti dell'insieme di conflitto di Y vanno aggiunti a quello di X
 - Allo stesso tempo ridondante con forward checking: ogni ramo dell'albero potato dal backjumping è parimenti potato dalla verifica in avanti
 - Backjumping guidato dai conflitti: ci indica fino a dove risalire in modo da non perdere tempo a modificare variabili che non risolveranno il problema
 - Si applica laddove il backjumping fallisce in quanto questi rileva un fallimento solo quando il dominio di una variabile diventa vuoto, ma in molti casi un ramo è condannato molto prima che questo si verifichi
 1. Sia X_j la variabile corrente e $conf(X_j)$ l'insieme dei suoi conflitti
 2. Se ogni possibile valore di X_j fallisce, si salta indietro alla variabile più recente X_i in $conf(X_j)$ e si assegna $conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_j\}$
 - Insieme **no-good**: insieme minimo di variabili dell'insieme dei conflitti, con i relativi valori, che è la responsabile dell'inconsistenza.
 - Si aggiunge ad una cache separata, per evitare di imbattersi nello stesso problema
- Ricerca locale:
 - Si usa uno stato completo e si modifica il valore di una variabile per volta (i.e. 8 regine)
 - Generalmente l'ipotesi iniziale viola diversi vincoli, che si punta ad eliminare
 - L'euristica **min-conflicts**: scegliere il valore che risulta nel numero minimo di conflitti con le altre variabili
 - Nelle 8 regine, il tempo di esecuzione è quasi indipendente dalle dimensioni del problema! In media 50 passi
 - Funziona bene per le 8 regine perché è un problema facile: le soluzioni sono distribuite densamente nello spazio degli stati
 - Purtroppo presenta solitamente un plateau: potrebbero esserci milioni di assegnamenti che distano di un conflitto dalla soluzione
 - Si può indirizzare la ricerca nel plateau tramite **tabu search**, mantenendo un elenco degli stati visitati di recente per impedire all'algoritmo di ritornarvi
 - **Constraint weighting**: si aggiunge una topografia ai plateau
 1. A ogni vincolo è assegnato un peso numerico $W_i = 1$ per tutti
 2. L'euristica è modificare una variabile col valore che porterà il minimo peso totale di tutti i vincoli violati
 3. I pesi sono aggiustati incrementando il peso di ciascun vincolo violati
 - Molto usato in ambiente online, ad esempio nelle linee aeree per ottenere schedule alternativo con numero minimo di cambiamenti
 - Una ricerca con backtracking potrebbe prendere molto più tempo e trovare una soluzione molto diversa dall'originario
- Sfruttare la struttura del grafo dei vincoli
 - Sottoproblemi indipendenti, trovabili cercando componenti connessi
 - Se l'assegnamento S_i è una soluzione di CSP_i , allora $\cup_i S_i$ è una soluzione di $\cup_i CSP_i$
 - Complessità $O(d^n/c)$ se ci sono n/c sottoproblemi risolvibili in d^c , invece che $O(d^n)$ (lineare in n)
 - Alberi: un grafo dei vincoli è un albero quando variabili qualsiasi sono collegate da un solo cammino
 - Può essere risolto in tempo lineare al numero di variabili
 - Directed Arc Consistency (DAC): un CSP è arco orientato consistente con ordinamento di variabili $X_1, \dots, X_n \Leftrightarrow$ ogni X_i è arco-consistente con ogni $X_j, j > i$

1. Si ordina topologicamente le variabili scelta la radice dell'albero
2. Si rende l'albero arco-consistente in $O(nd^2)$
3. Si percorre la lista di variabili e si sceglie qualsiasi valore rimanente
 1. Poiché ogni collegamento è arco-consistente, si può percorrere la lista delle variabili e scegliere qualsiasi valore rimanente sapendo che ci sarà un valore valido nel figlio
 - Non occorre backtracking, $O(nd^2)$ sul numero di variabili
- Riduzione di grafo dei vincoli ad albero
 - Rimozione dei nodi
 1. Scegliere sottoinsieme di S , chiamato **cycle cutset**, delle variabili del CSP
 - Trovare il più piccolo insieme di taglio dei ciclo è NP-hard, ma ci sono algoritmi approssimati efficienti
 2. Per ogni assegnamento delle variabili in S che soddisfi i vincoli in S
 - Rimuovere dal dominio delle variabili rimanenti tutti i valori non consistenti con gli assegnamenti in S
 - Restituire l'eventuale soluzione dell'albero insieme all'assegnamento per S
 - Complessità $O(d^c \cdot (n - c)d^2)$ se l'insieme di taglio dei cicli ha dimensioni c
 - Funziona bene se il grafo è "quasi-albero", quindi c basso
 - Fusione dei nodi: si scompone in un albero di sottoproblemi collegati.
 - Una scomposizione deve soddisfare:
 1. Ogni variabile del problema originale deve comparire in almeno uno dei sottoproblemi
 2. Se due variabili sono collegate da un vincolo devono comparire insieme con il vincolo in almeno uno dei sottoproblemi
 3. Se una variabile compare in due sottoproblemi sull'albero, deve essere presente in tutti i sottoproblemi che compongono il cammino che li collega
 - Ogni variabile deve avere lo stesso valore in ognuno dei sottoproblemi in cui appare
 - Ogni sottoproblema viene risolto in maniera indipendente
 - Se un sottoproblema non ha soluzione, non ce l'ha nemmeno il problema originale
 - Altrimenti ogni sottoproblema viene vista come una variabile gigante il cui dominio è l'insieme delle soluzioni
 - Usando l'algoritmo degli alberi, si possono risolvere i vincoli sui sottoproblemi
 - La regola 3 impone alle rispettive soluzioni di concordare sui valori delle variabili comuni
 - La dimensione del sottoproblema più grande -1 si chiama **larghezza d'albero**. Quella di un grafo è la minima tra le sue scomposizioni ad albero.
 - Complessità temporale $O(nd^{w+1})$
 - I CSP i cui grafi dei vincoli hanno una larghezza d'albero limitata sono risolvibili in tempo polinomiale.
 - Trovare la scomposizione con larghezza d'albero minima è NP-hard, ma ci sono metodi euristici
- Struttura dei valori: per ogni soluzione consistente esiste un insieme $n!$ di soluzioni permutando i nomi dei colori (**simmetria di valore**)
- Si può ridurre quindi di $n!$ lo spazio di ricerca introducendo un vincolo di rottura della simmetria, ad esempio un ponendo un vincolo di ordinamento dei colori

Agenti logici

- Base di conoscenza (KB knowledge base): insieme di formule, espresse mediante un linguaggio di rappresentazione della conoscenza
 - Ogni formula rappresenta un'asserzione sul mondo
 - Deve prevedere meccanismi per aggiungere nuove formule e per interrogazioni tramite *TELL* e *ASK*
 - Richiedono **inferenza**: derivazione di nuove formule a partire da quelle conosciute
 - La risposta deve essere una conseguenza di quello che è stato detto *TELL* in precedenza
 - Può avere una base di conoscenza iniziale (*background knowledge*)
- Un agente basato sulla conoscenza fa tre cose:
 1. Comunica le percezioni alla base di conoscenza attraverso *TELL*
 2. Chiede *ASK* quale azione eseguire
 3. Registra l'azione nella base di conoscenza con *TELL* prima di eseguirla
- Un agente basato sulla conoscenza può essere costruito semplicemente dicendogli *TELL* ciò che deve sapere. Questo è un approccio dichiarativo, invece di codificare direttamente i comportamenti desiderati con un approccio procedurale.
- Mondo di Wumpus
 - PEAS

- Misura di performance: +1000 se si esce dalla caverna, -1000 se si muore, -1 per ogni azione, -10 uso freccia
 - Ambiente: griglia 4x4, inizio in [1, 1]. Tutti i riquadri hanno probabilità 0.2 di contenere un pozzo
 - Discreto, statico, monoagente, sequenziale, parzialmente osservabile
 - Attuatori: Avanti, GiraSinistra, GiraDestra, Afferra, Scocca
 - Sensori: Fetore, Brezza, Scintillio, Urto, Ululato
 - La principale difficoltà è l'ignoranza della configurazione dell'ambiente, bisogna usare un ragionamento logico
 - KB = regole del mondo dei wumpus + percezioni
- Modelli
 - m è un modello di una sentenza α se α è vera in m
 - $M(\alpha)$ è l'insieme di tutti i modelli di α
 - $KB \models \alpha \Leftrightarrow M(KB) \subseteq M(\alpha)$
 - Il model checking è un algoritmo di inferenza logica che enumera depth-first tutti i possibili modelli per verificare che $KB \models \alpha$
 - I.e. KB = "nulla in [1, 1] e brezza in [2, 1]" e α = "Non c'è nulla in [1, 2]"
 - Completo se lo spazio dei modelli di KB è finito
 - Corretto, poiché applica solo la definizione di conseguenza logica
 - Complessità temporale $O(2^n)$ e spaziale $O(n)$ (l'enumerazione viene fatta in profondità)
 - La conseguenza logica proposizionale è co.NP-completa, quindi nel caso peggiore ogni algoritmo di inferenza è esponenziale
 - Proprietà algoritmi di inferenza
 - Correttezza: preserva la verità
 - Completezza: può derivare ogni formula che è conseguenza logica
 - Logica proposizionale
 - Formule atomiche: consistono di un singolo simbolo. Sono vere o false.
 - Formule complesse formate da formule più semplici usando connettivi logici: not, and, or, implicazione, sse.
 - Due formule sono logicamente equivalenti $\alpha \equiv \beta$ se sono vere nello stesso insieme di modelli
 - Una formula è valida se è vera in tutti i modelli: tautologia, cioè equivalente a True
 - Date due formule qualsiasi α e β , $\alpha \models \beta$ sse la formula $(\alpha \Rightarrow \beta)$ è valida, cioè equivalente a True
 - Una formula è soddisfacibile se è vera in qualche modello. Determinare la soddisfacibilità delle formule della logica proposizionale, problema SAT, è NP-completo.
 - Dimostrazione per refutazione o per contraddizione: $\alpha \models \beta$ sse la formula $(\alpha \wedge \neg \beta)$ è insoddisfacibile
 - Monotonicità: conoscenze aggiuntive possono solo aiutare a trarre nuove conclusioni, mai invalidare conclusioni già dedotte. Se $KB \models \alpha$ allora $KB \wedge \beta \models \alpha$
 - Oppure anche: le regole di inferenza possono essere applicate non appena si trovano nella base di conoscenze le premesse necessarie, indipendentemente dal resto delle formule nella KB
 - Regola di inferenza: applicabile per derivare una dimostrazione, conclusioni che portano all'obiettivo desiderato
 - **Modus Ponens:** $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$
 - Dimostrazione di teoremi: applicando regole di inferenza direttamente alle formule della KB per costruire una dimostrazione della formula desiderata senza consultare i modelli
 - Possiamo applicare uno degli algoritmi di ricerca per trovare una sequenza di passi che costituisca una dimostrazione
 - Stato iniziale: KB
 - Azioni: insieme delle regole di inferenza applicate a tutte le formule che corrispondono alla metà superiore della regola di inferenza
 - Risultato: aggiunta della formula nella metà inferiore della regola di inferenza
 - Obiettivo: Uno stato che contiene la formula che vogliamo dimostrare
 - Trovare una dimostrazione può essere molto efficiente perché si possono ignorare le proposizioni irrilevanti
 - Singola regola di inferenza, la **risoluzione**: unita a qualsiasi algoritmo di ricerca completo dà luogo ad un algoritmo di inferenza completo

$$\frac{l_1 \vee \dots \vee l_k, m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{i-1} \vee m_{i+1} \vee \dots \vee m_n}$$

1. Prende due clausole, disgiunzioni di letterali, con l_i e m_i letterali complementari e ne produce una nuova che contiene tutti i letterali delle due clausole tranne i due complementari
2. La clausola risultante contiene solo una copia di ogni letterale (**fattorizzazione**)
- Ogni formula della logica proposizionale è equivalente ad una congiunzione di clausole: **conjunctive normal form (CNF)**
 1. Eliminare i sse $\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
 2. Eliminare le implicazioni $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$
 3. Richiede che \neg si applichi solo ai letterali applicando De Morgan e la doppia negazione
 - $\neg(\neg\alpha) \equiv \alpha$
 - $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
 - $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
 4. Applicare la legge distributiva (\vee su \wedge) per portare tutto su un livello $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
- Le procedure di inferenza basate sulla risoluzione sfruttano il principio di dimostrazione per assurdo. Algoritmo

CP-RISOLUZIONE:

1. Si converte $KB \wedge \neg\alpha$ in CNF
2. Viene applicata la risoluzione ad ogni coppia di clausole
 1. Ogni coppia che contiene letterali complementari è risolta e la nuova clausola viene aggiunta all'insieme se non presente
3. Il processo continua finché:
 1. Non è più possibile aggiungere nuove clausole. $KB \not\models \alpha$
 2. La risoluzione tra due clausole dà la clausola vuota. $KB \models \neg\alpha$ è False, quindi $KB \models \alpha$
 - La clausola vuota è equivalente a False perché è una disgiunzione senza alcun disgiunto ed una disgiunzione è vera solo se è vero almeno uno dei disgiunti
- L'algoritmo è completo
 - **Chiusura della risoluzione** $RC(S)$ di un insieme di clausole S : è l'insieme di tutte le clausole derivabili dall'applicazione ripetuta della regola di risoluzione alle clausole in S o a quelle da loro derivate
 - $RC(S)$ è finito in quanto è finito il numero di clausole distinte costruite con i simboli di S grazie alla fattorizzazione \Rightarrow CP-RISOLUZIONE termina sempre l'esecuzione
 - **Teorema di risoluzione ground**: se un insieme di clausole è insoddisfacibile, la sua chiusura della risoluzione contiene la clausola vuota.
 1. Dimostrazione per contrapposizione: se la chiusura $RC(S)$ non contiene la clausola vuota, S è soddisfacibile
 2. Possiamo costruire un modello di S assegnando adeguati valori di verità a P_1, \dots, P_k
 3. L'assegnamento dato è un modello di S . Dimostriamo ipotizzando l'opposto
 4. In qualche fase i c'è stata un'assegnazione che rende falsa una clausola C
 - Devono esserci due letterali complementari P_i e $\neg P_i$
 - i non può essere 1 altrimenti $RC(S)$ conterrebbe una clausola vuota
 - Dato che $RC(S)$ è chiuso rispetto alla risoluzione, contiene anche il risolvente, che però avrà già tutti i letterali falsi per P_1, \dots, P_{i-1}
 - Viola l'ipotesi che la prima clausola falsa appaia nella fase i
- Clausole di Horn e clausole definite
 - Se le formule della KB hanno certe restrizioni, si possono applicare algoritmi di inferenza più ristretti ed efficienti della risoluzione
 - **Clausola definita**: disgiunzione di letterali di cui *esattamente uno* è positivo. $\neg L_{1,1} \vee \neg Brezza \vee B_{1,1}$
 - **Clausola di Horn**: disgiunzione di letterali in cui *al massimo uno* dei letterali è positivo
 - Le clausole definite sono un caso particolare
 - Le clausole senza letterale positivo si chiamano *clausole obiettivo*
 - Sono interessanti perché:
 1. Possono essere riscritte come implicazione $L_{1,1} \wedge Brezza \Rightarrow B_{1,1}$. La premessa si chiama corpo, la conclusione testa. I letterali positivi nel corpo sono fatti.
 2. L'inferenza sulle clausole di Horn può essere fatta con concatenazione in avanti e all'indietro
 3. Si può determinare la conseguenza logica in tempo lineare rispetto alla dimensione della KB
 - Forma di Horn: congiunzione di clausole di Horn
 - Concatenazione in avanti: determina se un singolo simbolo q è conseguenza logica dei *fatti* nella KB
 1. Se tutte le premesse di un'implicazione sono verificate, la conclusione è aggiunta ai fatti noti
 2. Continua finché non viene aggiunta la query q oppure non sono possibili ulteriori inferenze
 - Corretto: applica il Modus Ponens
 - Completo: ogni sentenza atomica che è conseguenza della KB sarà derivata
 1. L'algoritmo raggiunge un punto fisso dove nessuna nuova sentenza atomica è derivata
 2. Si può considerare lo stato finale come un modello m che assegna True ai simboli inferiti, False agli altri

3. Ogni clausola nella KB originale è vera in m , altrimenti avremmo qualche clausola $a_1 \wedge \dots \wedge a_k \Rightarrow b$ falsa nel modello
 - Allora $a_1 \wedge \dots \wedge a_k$ è True in m e b è False in m quindi non avremmo raggiunto un punto fisso
4. m è quindi un modello per KB
5. Se $KB \models q$ allora in ogni modello di KB q è vera, compreso m
 - Ragionamento guidato dai dati: l'attenzione parte dai fatti conosciuti
- o Concatenazione all'indietro: parte dalla query e lavora a ritroso
 1. Trova tutte le implicazioni nella KB che hanno q come conclusione
 2. Se tutte le premesse di una delle implicazioni possono essere dimostrate vere, allora q è vera
 3. Si ripete il procedimento quindi per le premesse, viste come query, fino a raggiungere un insieme di fatti noti che formano la base della dimostrazione
 - Per evitare cicli controlla se un nuovo sottogoal è già presente nella lista dei goal o se è già fallito
 - Ragionamento basato sugli obiettivi
 - Complessità spesso meno che lineare sulla dimensione della KB, in quanto coinvolge solo i fatti rilevanti

Logica di primo ordine

- Ciò che manca nei linguaggi di programmazione è un meccanismo che consenta di derivare fatti da altri fatti
- La logica proposizionale separa invece la conoscenza dall'inferenza e quest'ultima rimane totalmente indipendente dal dominio
- La logica proposizionale è un linguaggio troppo poco potente per rappresentare la conoscenza di ambiti complessi in modo compatto
 - o Siamo stati costretti a scrivere per ogni stanza $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
 - o I modelli collegano simboli a valori di verità
- I modelli della logica di primo ordine contengono oggetti
 - o Il **dominio** di un modello è l'**insieme di oggetti** che contiene (non solo True e False) e non deve essere vuoto
 - {Riccardo Cuor di Leone, suo fratello minore, una corona, il malvagio Re Giovanni, le gambe sinistre di Riccardo}
 - o Una **relazione** è un insieme di tuple di oggetti collegati.
 - Relazione fratello: $\{ \langle \text{Riccardo Cuor di Leone}, \text{Re Giovanni}, \langle \text{Re Giovanni}, \text{Riccardo Cuor di Leone} \rangle \rangle \}$
 - o **Funzioni** relazioni particolari in cui ogni dato oggetto può essere collegato ad esattamente un altro oggetto
 - Funzione unaria "Gamba sinistra":
 - $\langle \text{Riccardo Cuor di Leone} \rangle \rightarrow \text{la gamba sinistra di Riccardo}$
 - $\langle \text{Re Giovanni} \rangle \rightarrow \text{la gamba sinistra di Giovanni}$
 - o Un **termine** è un'espressione logica che si riferisce ad un oggetto: $\text{GambaSinistra}(\text{Giovanni})$
 - o Una formula atomica è composta da un simbolo di predicato seguito da una lista di termini tra parentesi $\text{Fratello}(\text{Riccardo}, \text{Giovanni})$
 - Una formula atomica è vera in un dato modello se la relazione a cui fa riferimento il simbolo di predicato è verificata tra gli oggetti a cui fanno riferimento gli argomenti (nel mondo reale). I.e. $\text{Arma}(M_1)$ se M_1 è effettivamente un'arma
 - o Formule complesse:
 - Utilizzo dei connettivi logici della logica proposizionale
 - Quantificatori: per esprimere caratteristiche di intere collezioni di oggetti senza doverli enumerare
 - Universale $\forall x \text{ Re}(x) \Rightarrow \text{Persona}(x)$
 - $\forall x P$ è vera in un dato modello se P è vera in tutte le possibili interpretazioni estese costruite a partire dall'interpretazione fornita nel modello
 - Interpretazione estesa: specifica un elemento del dominio a cui x fa riferimento
 - Tipicamente \Rightarrow è il connettivo principale usato con \forall
 - Esistenziale $\exists x \text{ Corona}(x) \wedge \text{SullaTesta}(x, \text{Giovanni})$: per formulare enunciati circa alcuni oggetti del dominio senza citarli per nome
 - $\exists x P$ è vera in un modello se P è vera in almeno una interpretazione estesa
 - Tipicamente \wedge è il connettivo principale usato con \exists
 - I due quantificatori sono strettamente correlati attraverso la negazione.
 - \forall è una congiunzione degli oggetti, \exists è una disgiunzione
 - Leggi di De Morgan:
 - $\forall x \neg P \equiv \neg \exists x P$
 - $\forall x P \equiv \neg \exists x \neg P$
 - $\exists x \neg P \equiv \neg \forall x P$
 - $\exists x P \equiv \neg \forall x \neg P$
 - o Semantica dei database:
 - "Riccardo ha due fratelli, Giovanni e Goffredo": $\text{Fratello}(\text{Giovanni}, \text{Riccardo}) \wedge \text{Fratello}(\text{Goffredo}, \text{Riccardo})$

- Ipotesi dei nomi unici: ogni simbolo di costante fa riferimento ad un oggetto distinto
- Ipotesi del mondo chiuso: le formule atomiche non conosciute come vere sono considerate false
- Chiusura del dominio: ogni modello contiene un numero di elementi del dominio non superiore a quello degli elementi denominati dai simboli di costante
- Inferenza proposizionale e inferenza del primo ordine
 - Si converte la KB in logica proposizionale per poi usare l'inferenza proposizionale
 - Regola di **istanziamento universale** (UI): possiamo inferire delle le formule ottenute dal quantificatore universale sostituendo un termine *ground* (senza variabili) alla variabile
 - Da non confondersi con l'interpretazione estesa che mette in corrispondenza variabili con oggetti del dominio
 - **Sostituzione**: per ogni variabile v e termine *ground* g

$$\frac{\forall v \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

- UI si può applicare più volte per aggiungere nuove sentenze
 - Regola di **istanziamento esistenziale** (EI): la variabile è sostituita da un unico nuovo simbolo di costante che non compare da nessun'altra parte nella KB (costante di Skolem)
- $$\frac{\exists v \alpha}{\text{SUBST}(\{v/k\}, \alpha)}$$
- Si può applicare EI solo una volta per rimpiazzare la sentenza esistenziale
 - **Proposizionalizzazione**: si applica UI in tutti i possibili modi, EI a tutte le istanze esistenziali e si considerano le formule atomiche *ground* alla stregua di simboli proposizionali
 - Il metodo della proposizionalizzazione è completo: ogni formula che segue logicamente dalla KB originale può essere dimostrata nella nuova KB. tuttavia quando non lo è è impossibile determinare il contrario
 - Problema per i simboli di funzione con infinito numero di termini *ground*
 - Teorema di Herbrand: se una sentenza α è conseguenza logica di una KB in FOL, esse è conseguenza logica di un sottoinsieme *finito* della KB proposizionale
 - Applicare una sorta di approfondimento iterativo con profondità crescente della KB proposizionale, ma funziona solo se α è conseguenza logica
 - Il problema della conseguenza logica per la logica del primo ordine è **semi-decidibile**: esistono algoritmo che rispondono affermativamente per ogni formula che p conseguenza logica, ma nessun algoritmo potrà rispondere negativamente per ogni formula che non è conseguenza logica.
 - La proposizionalizzazione inoltre è piuttosto inefficiente, genera formule inutili dall'istanziamento.
 - Genera pn^k istanziazioni!, k arità massima, p predicati, n simboli distinti di costante
 - **Modus Ponens generalizzato**: troviamo una sostituzione sia per le variabili nell'implicazione che per quelle nelle formule della KB.
 - Per le formule atomiche p_i, p'_i, q dove ci sia una sostituzione θ tale che $\text{SUBST}(\theta, p'_i) = \text{SUBST}(\theta, p_i)$ per tutti gli i

$$\frac{p'_1, \dots, p'_n, (p_1 \wedge \dots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

- È più generale del Modus Ponens in quanto i fatti noti e le premesse dell'implicazione devono corrispondere soltanto tramite una sostituzione. Allo stesso tempo il Modus Ponens però permette come premessa qualsiasi formula α e non soltanto una congiunzione di formule atomiche
- Il Modus Ponens generalizzato è ottenuto da quello proposizionale attraverso il processo di **lifting**, "sollevando" il Modus Ponens dalla logica proposizionale, *ground* senza variabili, a quella del primo ordine
 - Si effettuano solo le sostituzioni effettivamente necessarie per portare avanti le inferenze
- **Unificazione**: trovare sostituzioni che rendono identiche espressioni logiche diverse
 - $\text{UNIFY}(p, q) = \theta$ ove $\text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$
 - $\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(y, \text{Guglielmo})) = \{x/\text{Guglielmo}, y/\text{Giovanni}\}$
 - Standardizzazione separata: rinomina delle variabili di una delle formule per evitare collisioni
 - $\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(z, \text{Elisabetta})) = \{x/\text{Elisabetta}, z/\text{Giovanni}\}$
 - MGU: per ogni coppia di espressioni unificabili, esiste un singolo unificatore più generale distinto da tutti gli altri qualora esista più di un unificatore.
 - Questo calcolo spesso include un controllo di occorrenza, $S(x)$ non si può unificare con $S(S(x))$, rende quadratica la complessità di UNIFY, per cui viene omissso in alcuni sistemi.
- Clausole definite del primo ordine: una formula atomica oppure un'implicazione in cui il corpo è una congiunzione di letterali positivi e la conseguenza è un singolo letterale positivo
 - Possono includere variabili, considerate come quantificate universalmente
- La legge americana afferma che per un cittadino è un crimine vendere armi a una nazione ostile. Lo stato di Nono, un nemico dell'America, possiede dei missili, e gli sono stati venduti tutti dal Colonnello West, un americano."

- La relativa base di conoscenza non contiene simboli di funzione: Datalog. L'assenza di funzioni rende l'inferenza molto più facile. Termina in tempo polinomiale $O(pn^k)$
 - Concatenazione in avanti
 - Partendo dai fatti noti, si fanno scattare tutte le regole le cui premesse sono soddisfatte, aggiungendo le relative conclusioni ai fatti noti
 - Il processo si ripete finché si trova una risposta oppure non è più possibile aggiungere nuovi fatti (punto fisso)
 - Un fatto non è nuovo se consiste solo nella rinominazione di uno noto
 - Corretto: ogni inferenza è applicazione del Modus Ponens generalizzato
 - Completo per le query che sono conseguenza logica di una KB composta di sole clausole definite
 - Complessità $O(pn^k)$ con p predicati, n simboli di costante, k arità massima dei predicati
 - Se le clausole definite includono simboli di funzione si possono generare un numero infinito di fatti. In tal caso se la query q è una conseguenza logica, si può ricorrere al teorema di Herbrand per asserire che terminerà con successo in un numero finito di passi. Altrimenti l'algoritmo potrebbe non terminare mai.
 - 3 inefficienze:
 1. Pattern matching: trovare tutti gli unificatori tali che la premessa di una regola possa unificare con un insieme adeguato di fatti presenti nella KB
 - $\text{Missile}(x) \wedge \text{Possiede}(\text{Nono}, x) \Rightarrow \text{Arma}(x)$: in una KB indicizzata trovare i fatti che unificano può essere fatto in tempo costante
 - Nel caso di congiunti multipli, conviene ordinarli in modo tale che il costo sia minimizzato
 - Trovare l'ordinamento ottimo è NP-hard ma si possono usare euristiche come MRV
 - In generale cercare il matching tra una clausola definita e un insieme di fatti è un problema NP-hard
 - La maggior parte delle regole però sono piccole e semplici
 - Si possono eliminare tentativi di matching ridondanti
 2. Ricontrollare tutte le regole ad ogni iterazione, anche se la KB è cambiata poco
 - Si possono evitare matching ridondanti stabilendo che ogni nuovo fatto inferito durante l'iterazione deve venire da almeno un fatto nuovo inferito nell'iterazione precedente
 - Ad ogni iterazione saranno controllate solo le regole le cui premesse includono un congiunto p_i che unifica con un fatto p'_i inferito nell'iterazione precedente
 - È utile mantenere in memoria le corrispondenze parziali delle premesse delle regole, completandole gradualmente man mano che giungono nuovi fatti (algoritmo rete)
 3. Generare molti fatti che sono irrilevanti per l'obiettivo
 - Usare la concatenazione all'indietro
 - Riscrivere l'insieme di regole utilizzando informazione dall'obiettivo, in modo tale che solo i legami rilevanti appartenenti al *magic set* siano presi in considerazione durante l'inferenza in avanti
 - Idea proveniente dai database deduttivi, che usano l'inferenza anziché le query SQL
- Concatenazione all'indietro
 - Una query è dimostrata se la KB contiene una regola della forma $\text{lhs} \Rightarrow \text{obiettivo}$ con lhs una lista di congiunti
 - Può essere visto come un caso particolare di ricerca *AND/OR*
 - *OR* perché la query può essere dimostrata da qualsiasi regola della KB
 - *AND* perché tutti i congiunti della lhs devono essere dimostrati
 - 1. L'algoritmo cerca tutte le clausole che potrebbero unificare con l'obiettivo
 - 2. Per ogni clausola dimostra ogni congiunto uno per uno, tenendo traccia della sostituzione accumulata
 - È una ricerca in profondità: requisiti spaziali lineari con le dimensioni della dimostrazione
 - Incompleta a causa di cicli infiniti, bisogna controllare il goal rispetto ai goals sulla pila della ricerca in profondità
 - Inefficiente a causa di goal ripetuti, bisogna usare una cache che contiene i risultati già calcolati
 - Programmazione logica (Prolog)
 - Esegue concatenazione all'indietro con clausole di Horn
 - Insieme di clausole separate dalla virgola criminale(X) :- americano(X), arma(Y), vende(X, Y, Z), ostile(Z)
 - Può descrivere relazioni tra diversi argomenti
 - Utilizza la semantica dei database
- Risoluzione
 - Le formule devono essere in CNF e possono contenere variabili, considerate come universalmente quantificate
 - Ogni formula della logica del primo ordine può essere convertita in una formula CNF inferenzialmente equivalente.
 - La formula CNF non sarà soddisfacibile esattamente nei casi in cui non lo era quella originale
 - I passi sono i seguenti:
 1. Eliminazione delle implicazioni
 2. Spostamento all'interno delle negazioni
 3. Standardizzazione delle variabili
 4. Skolemizzazione: rimozione dei quantificatori esistenziali per eliminazione
 - Nel caso più semplice si tratta di applicare l'istanziamento esistenziale

- In generale le entità di Skolem devono dipendere da x : $\forall x[Animale(F(x)) \wedge \neg Ama(x, F(x))] \vee Ama(G(x), x)$
 - F e G sono dette funzioni di Skolem
5. Omissione dei quantificatori universali
6. Distribuzione di \vee su \wedge
- La regola di risoluzione è semplicemente una versione "sollevata" tramite lifting di quella proposizionale.

$$\frac{l_1 \vee \dots \vee l_k, m_1 \vee \dots \vee m_n}{SUBST(\theta, l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

- Due clausole, che non hanno variabili in comune grazie alle standardizzazione, possono essere risolte se contengono letterali complementari, ovvero se uno *unifica con la negazione* dell'altro.
- Risoluzione binaria $UNIFY(l_i, \neg m_i) = \theta$
- Fattorizzazione: rimozione di letterali ridondanti, ovvero se due letterali sono unificabili
- La combinazione di risoluzione binaria e fattorizzazione è completa
- La risoluzione dimostra che $KB \models \alpha$ provando che $KB \wedge \neg \alpha$ non è soddisfacibile, ovvero deriva la clausola vuota
- La concatenazione all'indietro può essere vista come un caso speciale di risoluzione che adotta una particolare strategia di controllo per decidere l'ordine di esecuzione delle risoluzioni
 - Si sceglie di risolvere con una clausola il cui letterale positivo si unifica con quello più a sinistra della clausola corrente sulla linea principale della dimostrazione
- Completezza
 - Completa per refutazione: se un insieme di formule S è insoddisfacibile, la risoluzione sarà sempre in grado di derivare una contraddizione in un numero finito di passi.
 - 1. Teorema di Herbrand: se S non è soddisfacibile, esiste un particolare insieme di istanze ground anch'esso insoddisfacibile
 - 2. Teorema di risoluzione ground: la risoluzione proposizionale è completa per le formule ground
 - 3. Lemma di lifting: per ogni dimostrazione con risoluzione proposizionale che usa un insieme di formule ground, esiste una corrispondente dimostrazione che usa le formule del primo ordine da cui sono state ricavate le formule ground originarie
- Miglioramenti:
 - Preferenze per clausole unitarie a cui applicare la risoluzione, in quanto stiamo cercando di produrre una clausola vuota. Si preferiscono quindi inferenze che producono clausole più corte
 - Nel caso particolare la risoluzione unitaria, in cui ogni passo deve obbligatoriamente coinvolgere una clausola unitaria, è completa per le clausole di Horn. Assomiglia alla concatenazione in avanti
 - Insieme di supporto: imporre che ogni passo di risoluzione coinvolga almeno un elemento dell'insieme, in modo da diminuire notevolmente lo spazio di ricerca. Si può ad esempio usare la query negata come insieme di supporto.
 - Risoluzione di input: ogni risoluzione usa solo risoluzioni di formule input (della KB o query). Il Modus Ponens ne è un esempio ed è completo per le KB in forma di Horn. Genera una tipica struttura a spina di pesce
 - Risoluzione lineare: come risoluzione di input ma ammette anche la combinazione del risolvente con i suoi antenati nell'albero di dimostrazione
 - Sussunzione: elimina tutte le formule più specifiche di una esistente nella KB. $P(A)$ è più specifica di $P(x)$

Trattamento dell'incertezza

- Gli agenti devono gestire l'incertezza in ambienti parzialmente osservabili e/o stocastici
- Lo stato di credenza (belief state) è una rappresentazione di tutti i possibili stati in cui può trovarsi l'agente
 - Svantaggi
 - Nella valutazione delle percezioni, sono considerati tutti i possibili casi nonostante quanto siano improbabili. Comporta grandi e complessi rappresentazioni degli stati di credenza
 - Un piano di contingenza che deve gestire tutti i possibili casi diventa molto grande
 - Spesso bisogna agire per raggiungere l'obiettivo anche in assenza di un piano che ne garantisca il successo
- La decisione razionale, laddove non vi sia garanzia di successi, dipende dall'importanza degli obiettivi e dalla probabilità di raggiungerli
- In campi come quello medico l'uso della logica fallisce per 3 motivi:
 - Pigrizia: troppo lavoro per elencare tutte le possibili cause o sintomi
 - Ignoranza teorica: la scienza non ha ancora la completa informazione del dominio
 - Ignoranza pratica: pur sapendo tutte le cause teoriche, non sappiamo tutto del paziente
- Teoria delle probabilità: fornisce un modo per quantificare l'incertezza causata da pigrizia ed ignoranza
- Teoria dell'utilità: per prendere una decisione tra diverse azioni, l'agente stabilisce una preferenza in base al grado di utilità dello stato risultante
- Teoria delle decisioni: combinazione delle probabilità e delle utilità degli stati

- **Maximum expected utility:** Un agente razionale sceglie l'azione che risulta nella massima utilità attesa, pesata per la relativa probabilità
- Nella teoria delle decisioni lo stato credenza non rappresenta solo i possibili stati, ma anche la loro probabilità.
 - $0 \leq P(w) \leq 1$ e $\sum_w P(w) = 1$
- Probabilità a priori/incondizionata $P(cavity)$: probabilità in assenza di ogni altra informazione/evidenza
- Probabilità condizionale o a posteriori $P(cavity|toothache)$ data da $P(a|b) = \frac{P(a \wedge b)}{P(b)}$
 - **Product rule:** $P(a \wedge b) = P(a|b)P(b)$
 - La costante $1/P(toothache)$ può essere vista come **costante di normalizzazione** per assicurarsi che la somma sia 1.
 - Può essere calcolata alla fine come somma sulle probabilità congiunte
 - Quando deve prendere una decisione, un agente deve condizionare su tutte le percezioni osservate
- **Distribuzione di probabilità:** fornisce le probabilità per tutti i possibili assegnamenti delle variabile discreta. I.e. $P(Weather)$
 - Per una variabile continua $P(x)$ è la probabilità che x cada in un intervallo arbitrariamente piccolo intorno a x
- **Distribuzione di probabilità congiunta:** per un insieme di variabili aleatorie, fornisce la probabilità di ogni evento atomico su tali variabili come una matrice. I.e. $P(Weather, Cavity)$
- Inferenza probabilistica: calcolo delle probabilità a posteriori per le proposizioni query data l'evidenza osservata
 - Probabilità di una proposizione $P(\phi) = \sum_{w:w \models \phi} P(w)$ si sommano gli eventi atomici dove essa è vera
 - Marginalization/summing out: sommiamo le probabilità di ogni possibili valore delle altre variabili, togliendole dalla equazione
 - In genere siamo interessati alla distribuzione congiunta a posteriori delle variabili di query X , dati e specifici valori delle variabili di evidenza E , con Y variabili nascoste
 - $P(X|e) = \alpha P(X, E = e) = \alpha \sum_y P(X, E = e, Y = y)$
 - X, E, Y danno tutte le variabili aleatorie del dominio
 - Data la distribuzione di probabilità congiunta, possiamo calcolare la probabilità di qualsiasi query di variabili discrete
 - Complessità spaziale e temporale $O(d^n)$ con d arietà massima delle variabili aleatorie
- Indipendenza
 - Due variabili X, Y sono indipendenti se $P(X|Y) = P(X)$ o $P(Y|X) = P(Y)$ o $P(X, Y) = P(X)P(Y)$
 - L'indipendenza tra due variabili si può stabilisce sulla base della conoscenza del dominio
 - L'indipendenza riduce drasticamente la dimensione della tabella delle probabilità congiunte
 - Se tutte le variabili sono indipendenti, la distribuzione congiunta può essere fattorizzata nelle distribuzioni di probabilità delle singole variabili
 -
 - Regola di Bayes: $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$
 - In campo medico: $P(causa|effetto) = \frac{P(effetto|causa)P(causa)}{P(effetto)}$
 - $P(effetto|causa)$ quantifica la relazione in senso causale
 - $P(causa|effetto)$ quantifica la relazione in senso diagnostico
 - Con la normalizzazione: $P(Y|X) = \alpha P(X|Y)P(Y)$
 - Di per sé non aiuta molto rispetto ad usare la probabilità congiunta, in quanto $P(X|Y)$ non scala con tante variabili aleatorie in X
 - Indipendenza condizionale tra due variabili X, Y data una terza variabile Z : $P(X, Y|Z) = P(X|Z)P(Y|Z)$
 - $P(Toothache, Catch|Cavity) = P(Toothache|Cavity)P(Catch|Cavity)$
 - La dimensione delle tabelle di probabilità cresce lineare invece che esponenziale, oltre al fatto che è molto più facile avere a disposizione la semplice probabilità condizionale tra 2 variabili, che quella congiunta di tutte variabili
 - **Naive Bayes:** $P(Causa|Effetto_1, ..., Effetto_n) = P(Causa) \prod_i P(Effetto_i|Causa)$
 - Assume che gli effetti siano tra loro condizionalmente indipendenti data la causa
- Mondo di Wumpus con modello probabilistico
 - Incertezza causata dalla parziale osservabilità dell'ambiente
 - Un agente logico non saprebbe concludere alcuna informazione utile sapendolo solo della possibilità di $P_{1,3}, P_{2,2}, P_{3,1}$
 - Un agente probabilistico può prendere una decisione migliore
- Reti Bayesiane
 - Rappresentano qualsiasi distribuzione di probabilità congiunta in maniera concisa
 - Una rete bayesiana è un grafo diretto aciclico in cui:
 - Ogni nodo corrisponde ad una variabile aleatoria
 - Un arco da Y a X indica che X è condizionalmente dipendente da Y , che ne è il genitore
 - Ogni nodo ha le probabilità condizionali $P(X_i|Parents(X_i))$ specificate in una conditional probability table (CPT)
 - La topologia della rete specifica le indipendenze condizionali del dominio
 - In genere è meglio creare collegamenti da cause ad effetti invece che il contrario, in modo da sfruttare l'indipendenza condizionale ed avere valori di probabilità più facili da reperire
 - L'insieme della topologia e delle probabilità condizionate è sufficiente per specificare la distribuzione congiunta per tutte le variabili

- $P(X_1, \dots, X_n) = \prod_i^n P(X_i | \text{Parents}(X_i))$
- Compattezza: la rete Bayesiana è un sistema **localmente strutturato** (sparse) in cui ogni nodo interagisce direttamente con solo un numero limitato di altri nodi, a prescindere dal numero totale di nodi.
 - La complessità cresce in maniera lineare invece che esponenziale
 - La complessità spaziale è $O(n2^k)$ per n variabili booleane con al massimo k costante genitori
- Inferenza esatta: avendo implicitamente tutte le probabilità congiunte è possibile calcolare $P(X|E = e)$
 - Reti singolarmente connesse (polytree) in cui c'è al più un cammino epr ogni coppia di connessi
 - Complessità dell'inferenza lineare sulla dimensione della rete definita come numero di righe CPT. Se ogni nodo ha un limite k costante di genitori, allora è lineare anche sul numero di nodi $O(nd^k)$
 - Reti multi-connesse: l'eliminazione di variabile ha complessità spaziale e temporale esponenziale
 - Possibile ridurre £SAT all'inferenza esatta, per cui la complessità è NP-hard
- Inferenza tramite simulazione stocastica con metodo Monte Carlo
 1. Estrarre N campioni da una distribuzione di campionamento S
 - Direct sampling: si generare eventi da una rete vuota in ordine topologico
 - Rejection sampling: si rigettano i campioni in disaccordo con l'evidenza. $P(X = x|E = e)$ è stimato contando quanto spesso $X = x$ nei rimanenti campioni
 - Likelihood weighting: genera solo eventi consistenti con l'evidenza e ed ogni evento è pesato dalla verosomiglianza (likelihood) dell'evidenza
 - Markov Chain Monte Carlo: ogni campione è generato tramite modifiche random al campione precedente
 2. Calcolare la probabilità a posteriori approssimata \hat{P}
 3. Mostrare che converge alla vera probabilità P

Apprendimento automatico

- Quando usarlo
 - Difficoltà del formalizzare il problema, ma ampia disponibilità di esempi
 - Gli esempi sono rappresentati in genere come vettore di caratteristiche
 - Presenza di rumore
- I tre elementi fondamentali di un algoritmo di apprendimento sono
 - La task
 - Classificazione, regressione, traduzione automatica, stima di densità etc.
 - La misura di performance
 - Accuracy della classificazione ad esempio oppure 0-1 loss
 - Dipende dal task, MSE per regressione ad esempio
 - L'esperienza
 - Caratteristiche a valori discreti o reali
 - Dati ottenuti una volta per tutte (batch learning) o incrementalmente agendo con l'ambiente (online learning)
- Apprendimento supervisionato: dato un insieme di esempi con output $Tr = \{(x^i, f(x^i))\}$, apprendere una funzione che stimi la reale funzione f
 - Tale stimatore viene usato per generalizzare, ovvero viene applicato per esempi non ancora visti
 - Si assume che un esperto fornisca la supervisione, ovvero fornisca i valori della f
 - I dati sono divisi in **training set** e **test set**
 - Training set a sua volta diviso in ulteriore training set e validation set
 - Il validation set serve per scegliere l'ipotesi $h \in H$ migliore tra quelle **consistenti** con il training set, ovvero senza errori di classificazione
- Apprendimento non supervisionato: dato un insieme di esempi senza output

KaTeX parse error: Undefined control sequence: \x at position 8: Tr = \{\x^i\} estrarre delle regolarità o pattern

 - Non esiste un esperto che fornisca aiuto, anche se ovviamente una conoscenza del campo aiuta a indirizzare la ricerca dei pattern
 - Clustering
- Si assume che la funzione da apprendere f possa essere rappresentata, o perlomeno approssimata, da una ipotesi $h \in H$
 - Dobbiamo ovviamente restringere lo spazio delle ipotesi H , per cui si parla di **bias induttivo**
 - L'errore ideale è la probabilità che h classifichi erroneamente un input non ancora visto
 - L'errore empirico è il numero di esempi di training set che h classifica erroneamente
 - Il validation set serve per evitare overfit, ovvero di preferire ipotesi h che abbiano errore empirico migliore di un'altra ipotesi, ma maggior errore ideale
 - VC-dimension e Shattering: la VC-dimension di un iperpiano in R^n è $n + 1$
 - $error_D(h(x)) \leq error_{Tr}(h(x)) + \epsilon(N_{Tr}, VC(H), \delta)$

- errore ideale \leq errore empirico + VC-confidence
- La VC-confidence è inversamente proporzionale alla dimensione del training set N , all'intervallo di confidenza δ e direttamente proporzionale alla VC-dimension $VC(H)$
- Structural Risk Minimization (SRM)
 - All'aumentare della VC-dimensione diminuisce l'errore empirico ma aumenta la VC-confidence
 - L'approccio SRM tenta di trovare un compromesso tra i due termini

Apprendimento con rinforzo

- Problema a decisione sequenziale: l'utilità di un agente dipende dalla sequenza di decisioni
- **Markov decision process** (MDP): problema a decisione sequenziale con ambiente osservabile ma stocastico, modello di transizione Markoviano e ricompense additive
 - Transition model: descrive il risultato stocastico di ogni azione in ogni stato $P(s'|s, a)$.
 - Assunzione di Markov: $s_{t+1} = P(s_t, a_t)$ e $r_t = r(s_t, a_t)$
 - Lo stato successivo e la ricompensa dipendono solo dallo stato corrente
 - Le funzioni P, r possono essere **deterministiche o non deterministiche** oltre che conosciute o non conosciute
 - La funzione di utilità dipende dalla sequenza di stati piuttosto che dal singolo stato
 - In ogni stato s l'agente riceve una ricompensa $R(s)$ che può essere positiva o negativa, ma deve essere limitata...
 - L'utilità di una sequenza di stati è la somma delle ricompense ricevute
 - L'equilibrio tra rischio e ricompensa cambia in base al valore di $R(s)$ per gli stati non terminali
 - Può avere **orizzonte finito o infinito**:
 - Orizzonte finito: dopo un tempo fisso N la soluzione non ha più valore, per cui influenza la decisione dell'azione ottimale nel tempo (non-stazionario)
 - Orizzonte infinito: non c'è alcun motivo per comportarsi diversamente nello stesso stato in tempi diversi (stazionario)
 - Le ricompense possono essere additive o scontate
 - **Ricompense additive**: l'utilità di una sequenza di stati è $U([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$ (algoritmi euristici)
 - **Ricompense scontate**: l'utilità di una sequenza di stati è $U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$ con $\gamma < 1$
 - Descrive la preferenza dell'agente verso ricompense immediate rispetto a quelle future
 - Un buon modello delle preferenze degli umani e degli animali nel tempo
 - L'utilità di una sequenza di stati infinita è finita: se $\gamma < 1$ e le ricompense sono bounded $\pm R_{max}$ si ha $U([s_0, s_1, s_2, \dots]) = \sum_t R(s_t) \leq \sum_t \gamma^t R_{max} = R_{max} / (1 - \gamma)$
 - Una sequenza fissa di azioni non può risolvere il problema, per cui c'è bisogno di una strategia **policy** π che specifichi cosa l'agente deve fare in ogni stato che raggiunge $\pi(s)$.
 - La strategia ottimale π^* è quella che massimizza l'utilità attesa $U^\pi(s) = E[\sum_t \gamma^t R(s_t)]$
 - Sceglie l'azione che massimizza l'utilità attesa dello stato successivo $\pi^s(s) = \operatorname{argmax}_{a \in A(s)} \sum P(s'|s, a) U(s')$
 - Funziona solo se si conosce il modello di transizione $P(s'|s, a)$
 - **Equazione di Bellman**: $U(s) = R(s) + \gamma \max_{a \in A(s)} \sum P(s'|s, a) U(s')$
 - Con n possibili stati, ci sono n equazioni di Bellman, una per ogni possibile stato. La risoluzione avviene con approccio iterativo, aggiornando man mano i valori delle utilità di ogni stato fino a raggiungere un punto fisso.
 - Funzione di update: $U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum P(s'|s, a) U_i(s')$
 - Convergenza dell'iterazione
 - La funzione di update è una contrazione, ovvero presi due input produce due valori di output che sono più vicini di un fattore costante rispetto agli input
 - Una funzione contrazione ha un solo punto fisso
 - L'applicazione ripetuta della funzione contrazione raggiunge sempre il punto fisso nel limite
 - Nel caso di Bellman si può vedere che l'update è una contrazione di fattore γ se $\gamma < 1$
 - Apprendimento con rinforzo
 - Gioco di scacchi: la ricompensa (il reinforcement) è ricevuta solo al termine della partita e viene visto come parte della percezione
 - A differenza della strategia ottimale π^* , in questo problema di decisione di Markov non si conosce il vero modello di transizione o la funzione di ricompensa
 - L'agente percepisce le ricompense solo quando si muove in uno stato
 - Si può immaginare di giocare ad un gioco di cui non si conoscono le regole e dopo un centinaio di mosse, l'avversario annuncia che abbiamo perso. Questo è l'apprendimento con rinforzo
 - L'agente non sa come l'ambiente stocastico funzioni o il risultato delle sue azioni (**unknown Markov Decision Problem**)
 - **Agente Q-learning** sa confrontare le utilità attese delle azioni senza conoscere i loro risultati o il modello di transizione

- Invece di apprendere la funzione di utilità $U(s)$ e il modello di transizione $P(s'|s, a)$, apprende una rappresentazione del valore dell'azione a nello stato s , denotato con $Q(s, a)$ ed osserva le ricompense immediate r
 - $U(s) = \max_a Q(s, a)$
- L'agente calcola iterativamente l'equazione $\hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a')$, dove \hat{Q} è la funzione corrente appresa che approssima Q
- Deve saper bilanciare il tradeoff tra **exploitation**, ovvero eseguire azioni che massimizzano le ricompense secondo il modello stimato corrente, e l'**esplorazione** che massimizza il benessere a lungo termine
- Passi:
 1. Per ogni stato s , inizializza la entry della tabella $Q(s, a) = 0$
 2. Osserva lo stato corrente s
 3. loop
 1. Seleziona un'azione a ed eseguila
 - Strategia random che favorisce l'esplorazione o $\max_a \hat{Q}(s, a)$ per sfruttamento del modello corrente
 2. Ricevi la ricompensa immediata r
 3. Osserva il nuovo stato s'
 4. Aggiorna la entry $\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$
 5. $s \leftarrow s'$
- Convergenza (caso deterministico): ogni (s, a) è visitato un numero infinito di volte
 1. Definiamo un intervallo pieno un intervallo durante il quale ogni $s(a, a)$ è visitato. Durante ogni intervallo pieno l'errore più grande nella tabella \hat{Q} (calcolato come **norma max**) è ridotto di un fattore γ .
 - Sia \hat{Q}_n la tabella dopo n aggiornamenti e Δ_n l'errore massimo in \hat{Q}_n , ovvero $\Delta_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$
 2. Ad ogni iterazione $n + 1$, l'errore nella stima rivista $\hat{Q}_{n+1}(s, a)$ è

$$\begin{aligned}
 |\hat{Q}_{n+1}(s, a) - Q(s, a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s', a')) - (r + \gamma \max_{a'} Q(s', a'))| \\
 &= \gamma |\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')| \\
 &\leq \gamma \max_{a'} |\hat{Q}_n(s', a') - Q(s', a')| \\
 &\leq \gamma \max_{s'', a'} |\hat{Q}_n(s'', a') - Q(s'', a')| \\
 |\hat{Q}_{n+1}(s, a) - Q(s, a)| &\leq \gamma \Delta_n
 \end{aligned}$$

Sapendo che $|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$

- Caso non deterministico, la regola di aggiornamento è $\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \alpha_n [r + \gamma \hat{Q}_{n-1}(s', a')]$
 - $\alpha_n = \frac{1}{1 + \text{visite}_n(s, a)}$ è il learning rate che decresce man mano

Natural Language Processing

- Obiettivo: un agente informatico in grado di comprendere il linguaggio naturale, acquisendo la conoscenza, e di comunicare con l'uomo
- 4 compiti fondamentali nell'elaborazione o comprensione del linguaggio naturale
 - Named entity recognition: trovare menzioni ad entità con nomi nel testo ed etichettarle per tipo di classe. I possibili tipi di entità dipendono dal dominio: luoghi, persone, organizzazione, proteine etc.
 - Entity mention detection: trovare le menzioni nel documento, ovvero riferimenti alla stessa entità
 - Relation extraction: trovare e classificare le relazioni semantiche tra le entità: figlio-di, part-di, relazioni geospaziali
 - Coreference resolution: collegare entità in insiemi corrispondenti ad entità del mondo reale: *United Airlines* e *United* o pronomi. Si basa su entity mention detection.
- NLP in termini di ricerca di informazioni:
 - Classificazione del testo
 - Information retrieval
 - Information extraction: processo di acquisizione della conoscenza dai documenti alla ricerca di occorrenze di una particolare classe, come ad esempio estrazioni di indirizzi da pagine web
- N-gram: stima la probabilità dell'ultima parola di una sequenza di N parole, come 2-gram (bigram) o 3-gram (trigram), date le precedenti parole
 - Oppure stima la probabilità dell'intera sequenza
 - Utilizzi:

- Conoscere le probabilità delle probabilità è essenziale in task che devono identificare le parole in presenza di rumore, come nel caso di speech recognition, per sapere quali sono le parole più probabili da riconoscere
- Identificazione della lingua: $\operatorname{argmax}_I P(I|w_1^n) = \operatorname{argmax}_I P(w_1^n|I)P(I)$
 - $P(w_1^n|I)$ è dato contando i trigram in uno corpus della lingua
 - $P(I)$ è data ad esempio dalla frequenza della lingua nelle pagine web
- Correzione ortografica
- Classificazione del testo
- Named-entity recognition
- Essenziale anche per correzione grammaticale e machine translations
- Modelli che assegnano probabilità a sequenze di parole si chiamano **language models**
- Computare $P(w|h)$, probabilità di una parola dato il testo precedente
 - **Chain rule of probabilità:** $P(w_1, w_2, \dots, w_m) = P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2) \dots P(w_n|w_1^{n-1})$
 - Possiamo stimare le probabilità contando $P(\text{caldo}|\text{Oggi è}) = \frac{\text{Count}(\text{Oggi è caldo})}{\text{Count}(\text{Oggi è})}$
 - Sarebbe impossibile in quanto il web stesso non è abbastanza grande per dare buone stime di tante possibili combinazioni
- Possiamo invece approssimare usando solo le ultime n parole che precedono
- Bi-gram approssima $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$
 - Maximum Likelihood Estimate: $P(w_n|w_{n-1}) = \frac{\text{Count}(w_n|w_{n-1})}{\text{Count}(w_{n-1})}$
 - Assunzione di Markov: la probabilità della parola dipende non dal passato, ma solo dalla parola precedente
 - Date le probabilità bigram, possiamo quindi approssimare la probabilità dell'intera sequenza $P(w_1^n) = \prod_k P(w_k|w_{k-1})$
 - Le probabilità del bigram sono ottenute contando le occorrenze da un corpus molto grande e normalizzando i risultati affinché la somma delle probabilità sia 1
- Nella pratica si usano modelli trigram o 4-gram/5-gram laddove ci siano sufficienti dati
 - Usiamo i logaritmi delle probabilità, altrimenti il prodotto delle probabilità tenderebbe a zero velocemente
- Possiamo misurare la qualità del modello n-gram usando un corpus di test: maggiore è la probabilità assegnata dal modello per il test maggiore è l'accuratezza del modello. Tuttavia la probabilità di un corpus grande sarà sempre molto piccola, rischiando quindi di incappare in floating-point underflow.
 - **perplexity** PP : è la probabilità inversa del test set, normalizzata dal numero di parole

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

- Più è piccola la probabilità, più grande è la perplexity
- Problema: combinazioni di parole non viste nel training corpus o con poche occorrenze
 - Smoothing: processo di regolarizzazione delle probabilità dei conteggi a bassa frequenza su una probabilità piccola, diversa da zero
 - Smoothing di Laplace: assegniamo probabilità $\frac{1}{(n+2)}$, ovvero assumiamo con con altre 2 prove troveremo una osservazione
 - **Backoff**: stimiamo la probabilità di un N-gram usando il (N-1)-gram di ordine inferiore
 - A volte usare meno contesto aiuta a generalizzare
 - **Linear Interpolation**: combiniamo le probabilità stimate da tutti gli n-gram, pesando e combinando trigram, bigram e unigram in base alle occorrenze
 - $\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda P(w_n)$ con $\sum_i \lambda_i = 1$
 - Possiamo valutare il modello migliore tramite cross-validation
- Parole sconosciute: si introduce la parola artificiale $\langle UNK \rangle$ e si contano le occorrenze nel corpus ogni volta che si incontra per la prima volta una parola.
- Esempio: Berkeley Restaurant Project con circa 9200 frasi che rispondeva a domande riguardo un database di ristoranti in Berkeley
 - Le probabilità bigram sono in grado di codificare fatti come preferenze di cibo, relazioni sintattiche (nome dopo aggettivo) o errori grammaticali
- Text classification: dato un testo decidere a quale classe appartiene
 - Esempi: identificazione della lingua, classificazione del genere, sentiment analysis e rilevamento di spam
 - Diversi approcci:
 1. Si può classificare usando la regola di Bayes: $\operatorname{argmax}_{c \in \{spam, ham\}} P(c|message) = \operatorname{argmax}_{c \in \{spam, ham\}} P(message|c)P(c)$
 - Possiamo definire un linguaggio del modello usando gli n-gram per $P(message|spam)$
 - Naive Bayes: $c_{MAP} = \operatorname{argmax}_{c \in C} P(c|d) = \operatorname{argmax}_{c \in C} \frac{P(d|c)P(c)}{P(d)} 0 \operatorname{argmax}_{c \in C} P(d|c)P(c)$
 - $P(c)$ può essere calcolata nel corpus

- $P(d|c) = P(x_1, x_2, \dots, x_n|c)$ come probabilità congiunta sarebbe difficile da calcolare e richiederebbe un dataset molto grande, ma possiamo usare l'assunzione Naive Bayes di indipendenza condizionale data la classe c
- $P(d|c) = \prod_i^n P(x_i|c)$

2. Possiamo altrimenti usare l'apprendimento automatico

1. Si rappresentano le parole del messaggio come un insieme di caratteristiche discrete

- **Bag of words:** si può usare un contatore per ogni volta che viene contato la parola, ma risulta in un vettore molto grande a causa del numero di parole nel vocabolario ma sparse perché solo alcune saranno usate nel messaggio
 - Perde nozione di ordine delle parole e di contesto, assume indipendenza di ordine del testo
- one-hot encoder: ancora più semplice, segna solo 1 se la parola compare
 - Perde completamente nozione di similarità
 - Si può usare tassonomie come Wordnet per ottenere insieme di sinonimi o altre relazioni
 - Assenza di sfumature: "proficient" è sinonimo di "good" ma solo in alcuni contesti
 - Difficile da tenere aggiornato
 - Soggettivo e richiede lavoro umano
 - Difficile quantificare la somiglianza esatta delle parole

- Word2vec embedding

2. Si apprende un classificatore sulle caratteristiche

3. Pensare alla classificazione come un problema di **data compression**

- Un algoritmo di compressione lossless prende una sequenza di simboli, rileva pattern ripetuti e scrive una descrizione della sequenza che è più compatta dell'originale (LZW algorithm)
- In classificazione, si comprimono tutti i messaggi di spam nel training set compure una singola unità. Analogo per ham. Dato quindi un nuovo messaggio da classificare, aggiungiamo all'unità degli spam e degli ham e comprimiamo il risultato.
- La classe che comprime meglio, ovvero aggiunge minor numero di bytes per il nuovo messaggio, è la classe predetta
 - L'idea è che un messaggio di spam tenderà a condividere le parole del dizionario di altri messaggi spam e quindi sarà compresso più efficacemente

4. Usare regole codificate a mano, come black-list-address o regex. Può avere alta accuracy ma è molto costoso, non sempre possibile e difficile da mantenere aggiornato

• Word2vec: vettori corti (50-1000 valori) e densi

- Vettori densi funzionano meglio rispetto a quelli sparsi come bag-of-words o one-hot encoder per i task NLP
 - Catturano meglio la similarità tra parole: due vettore delle parole hanno meno distanza se occorrono negli stessi contesti linguistici
- Invece di contare la frequenza di occorrenza di ogni parola w accanto ad un'altra, apprendiamo un classificatore binario. Non ci importa del classificatore, ma del vettore di pesi appreso da usare come word embedding
 - Word embedding: rappresentazione delle parole che permette di memorizzare le informazioni siano semantiche che sintattiche delle parole partendo da un corpus non annotato.
 - L'idea è nata dalle reti neurali che predicono la parola successiva date le precedenti
 - Il classificatore è più semplice come problema e ritorna $P(+|t, c)$, la probabilità che c sia una parola del contesto di t
 - Il vettore embedding è inizializzato con valori random e iterativamente modificato, usando la discesa di gradiente, affinché gli embeddings di parole vicine nel testo siano simili tra loro e diversi dagli embeddings di parole che non appaiono insieme
- Una caratteristica molto interessante degli embeddings è la loro abilità nel catturare relazioni tra i significati. La distanza tra due vettori racchiude qualche relazione di analogia tra le parole.
 - I.e. $\text{vector}(\text{king}) - \text{vector}(\text{man}) + \text{vector}(\text{woman}) = \text{vector}(\text{queen})$
 - I.e. $\text{vector}(\text{Paris}) - \text{vector}(\text{France}) + \text{vector}(\text{Italy}) = \text{vector}(\text{Roma})$

• Part-of-Speech Tagging (POS)

- Fin dal tempo dei greci, sono state notate 8 parti del discorso: nome, verbo, pronome, preposizione, avverbio, congiunzione, participio e articolo
- Le parti del discorso (o categorie sintattiche) forniscono informazioni sulle parole vicine, i.e. i nomi sono preceduti da articoli o aggettivi, i verbi dai nomi
 - Sono utili anche per la struttura sintattica, i.e. i nomi sono parte di una frase nominale
 - Addirittura utili per speech recognition o sintesi, i.e. pronuncia diversa in parte alla parte del discorso
- Le parti del discorso sono tradizionalmente definite in base alla sintassi e alla morfologia, raggruppando parole che sono accompagnate da altre parole simili tra loro
- Classi chiuse o aperte

- Classi chiuse: numero di termini abbastanza fissi, come le preposizioni. In genere sono parole corte ed hanno funzione strutturale nella grammatica. Differiscono anche di più da una lingua all'altra rispetto alle classi aperte
 - Classi aperte: termini nuovi conati spesso come nomi, verbi, aggettivi e avverbi
 - Treebank tagset: insieme di 45 parti di discorso usate per etichettare diverse corpora (plurale di corpus)
 - Brown corpus: 1 milione di parole da 500 testi scritti di diverso genere negli USA nel 1971
 - WSJ: milioni di parole pubblicate dal WSJ nel 1989
 - Part-of-speech tagging: il processo di assegnazione di un marcatore di parte di discorso ad ogni parola del testo di input. L'input è una sequenza di token e l'output è una sequenza di tags
 - La difficoltà è causata dall'ambiguità del tagging, una parola può avere diversi tag. I.e. *book*
 - In corpus Brown, 15% delle parole è ambiguo
 - Most Frequent Class: assegna ad ogni parola il tag più probabile, ovvero più frequente nel training corpus
 - Accuracy 92.34%
 - Viene usato come baseline per confrontare altri algoritmi più accurati. Lo stato dell'arte ha accuracy 97%
 - Rule based tagging
 - Dato un dizionario, assegna tutti i tag possibili alle parole del dizionario
 - Utilizza poi regole scritte a mano per rimuovere selettivamente i tag, cercando di lasciare quello corretto per ogni parola
 - Approccio statistico alternativo: 95-95% accuracy
 - Bayes $\arg\max P(W|T)P(T)$ o con assunzione di Markov $P(T) = \prod_k P(t_k|t_{k-1})$
 - Classificazione con sliding window: una finestra a dimensione fissa, ie 3, assegna il tag più probabile durante lo sliding in avanti usando la categoria del token precedente o successiva
 - Corregge ambiguità andando indietro
- Constituency Parsing
 - Context-free-grammar (CFG)**: consiste di un insieme di regole o produzioni, ognuna esprime i modi in cui i simboli del linguaggio possono essere raggruppati e ordinati insieme
 - Una grammatica specifica quali stringhe sono legali
 - Le regole possono essere combinate tra di loro
 - I simboli si dividono in **terminali**, che corrispondono alle parole del linguaggio tramite part-of-speech, o **non-terminali**, che esprimono astrazioni sui terminali
 - In una regola, come $NP \rightarrow Det Nominal$, a sinistra c'è un simbolo non-terminale e a destra 1+ simboli terminali o non terminali.
 - Una CFG può essere usata per generare frasi o per assegnare una struttura ad una frase, tramite un parse tree
 - Permettono di esprimere relazioni sofisticate tra le parole di un linguaggio, pur mantenendo la trattabilità computazione con efficienti algoritmi di parsing di frasi
 - In linguistica: l'uso di linguaggi formali per modellare linguaggi naturali si chiama **generative grammar**, in quanto il linguaggio è definito come l'insieme delle frasi "generate" dalla grammatica
 - Una CFG è in Chomsky normal form (CNF) se è privata del simbolo ϵ e le produzioni hanno nella parte destra solo 2 simboli non terminali o un simbolo terminale.
 - I parse tree risultanti sono binari
 - Constituency: astrazione di gruppi di parole che si comportano come single unità, costituenti
 - Frase nominali (noun phrases): sequenze di parole che circondano almeno un nome. I.e. "Harry the corpse", "Three parties from Brooklyn"
 - Compaiono anche in struttura sintattiche simili, ad esempio prima di un verbo, ovvero hanno simile distribuzione. Oppure possono sostituirsi a vicenda o essere uno l'espansione di un altro.
 - Penn Treebank: collezioni di testi di cui sono stati generati i parse tree, poi corretti a mano
 - Syntactic parsing è il task che assegna una struttura sintattica ad una frase basata su una CFG
 - Utile in controllo grammaticale: una frase che non può essere parserizzata probabilmente è errata grammaticalmente
 - Utile soprattutto come step intermedio per l'analisi semantica e l'information extraction: per rispondere alla domanda "What books were written by British women before 1800?" serve sapere chi è il soggetto e che "by British women" è un complemento d'agente
 - Anche il parsing sintattico soffre di ambiguità, come per il POS tagging, che in questo caso è **ambiguità strutturale**
 - L'ambiguità strutturale si presenta quando una grammatica può assegnare più di un parse tree ad una frase
 - Algoritmo CKY
 - Conversione della grammatica in CNF
 - Ogni non nodo terminale avrà quindi nel parse tree esattamente 2 figli
 - Viene usata una matrice per codificare la struttura del parse-tree, di dimensione $(n + 1) \times (n + 1)$ per una frase di n parole. Si usa la triangolare superiore.
 - Ogni cella $[i, j]$ contiene l'insieme di simboli non terminali che rappresenta tutti i costituenti che vanno dalla posizione i alla j dell'input. Quindi ad esempio la cella $[0, n]$ rappresenta l'intero input.

- La diagonale principale nella matrice contiene le parti del discorso per ogni parola dell'input
 - Le successive diagonali sopra alla diagonale contengono i costituenti che via via rappresentano porzioni di lunghezza crescente dell'input.
4. L'algoritmo CKY riempie la matrice da sinistra a destra e dal basso verso l'alto. Quando si riempie la cella $[i, j]$ le celle contenenti le parti che contribuiscono ad essa sono già riempite.
5. Per ogni coppia di cella controlla se i contenuti possono essere combinati secondo una produzione della grammatica
- Per generare tutti i possibili parse tree si aggiungono due modifiche all'algoritmo
 1. I simboli non terminali hanno puntatori che indicano da quali celle derivano
 2. È possibile avere più versioni dello stesso simbolo non terminale nella tabella
 - CKY permette di rappresentare le ambiguità ma non di risolverle
 - Complessità $O(n^3 |G|)$ con n lunghezza della stringa e $|G|$ dimensione della CFG
 - Complessità media migliore per versioni più avanzate dell'algoritmo
- Probabilistic context-free grammar (PCFG): ogni produzione della CFG è associata ad una probabilità
- $A \rightarrow \beta [p]$, p è la probabilità condizionale della derivazione β dato il non-terminale A
 - $\sum_{\beta} P(A \rightarrow \beta) = 1$
 - Permette di risolvere le ambiguità strutturali scegliendo l'interpretazione più probabile
 - Un PCFG consistente ha la somma delle probabilità di tutte le frasi di un linguaggio pari a 1.
 - PCFG assegna una probabilità all'intero parse tree per disambiguare
 - La probabilità di un parse tree T per la frase S è data da: $P(T, S) = \prod_i P(RHS_i | LHS_i)$, ovvero dal prodotto delle probabilità di tutte le n regole usate per espandere gli n nodi non-terminali
 - Probabilità della stringa S è data dalla somma delle probabilità dei parse tree $P(S) = \sum P(T, S)$
 - La probabilità risultante è sia la probabilità congiunta del parse che della frase. Scegliamo il parse tree con probabilità più alta
 - Un aspetto interessante è la possibilità di assegnare probabilità a sottostringhe di una frase, in maniera simile agli N-gram.
 - Tuttavia è più flessibile perché gli N-gram possono usare solo poche parole del contesto, mentre una PCFG può sfruttare l'informazione strutturale dell'intera frase per predire una parola.
 - I.e. "The contract ended with a loss of 7 cents after trading as low as 9 cents".
 - Il modo più semplice per apprendere le probabilità delle produzioni è usare un treebank con un corpus già parsato come il Penn Treebank, contando il numero di volte che una derivazione avviene e normalizzando.
 - $P(\alpha \rightarrow \beta | \alpha) = \frac{Count(\alpha \rightarrow \beta)}{Count(\alpha)}$
- Dependency Parsing
- Invece di descrivere la struttura sintattica della frase tramite costituenti, si usano le parole della frase a cui sono associate un insieme di relazioni binario tra di esse
 - **Typed dependency structure**: le relazioni sono rappresentate con archi direzionati ed etichettati dalla testa alla parola dipendente
 - Il nodo *root* fa da radice dell'albero, ovvero la testa dell'intera struttura
 - Le relazioni codificano importanti informazioni che altrimenti sarebbero implicite e "seppellite" nella struttura constituency, soprattutto nel caso di termini connessi ma distanti tra loro. "I prefer the morning flight through Denver"
 - Le grammatiche per dipendenza permettono di gestire linguaggio morfologicamente ricchi e che hanno relativa libertà d'ordine delle parole, come il ceco, a dispetto dell'inglese che ha una struttura piuttosto fissa
 - Una struttura a phrases richiederebbe una regola per ogni possibile posizione di un avverbio nel parse tree
 - Una struttura basata sulle dipendenze invece avrebbe semplice un solo tipo di collegamento rappresentate quel tipo di avverbio
 - Le relazioni testa-dipendente forniscono anche un'approssimazione alla relazione semantica tra predicati e i loro argomenti, molto utile per problemi di information extraction o question answering
 - Approcci basati sui costituenti forniscono informazioni simili ma vanno estratte indirettamente dai parse tree usando tecniche specifiche
 - Relazioni di dipendenza
 - Sono alla base delle relazioni grammaticali, formate da una testa (**head**) e un termine dipendente (**dependent**)
 - Analoghi ai costituenti dove il nome primario che fa da testa in una frase nominale oppure un verbo in una frase verbale, solo che invece di connettere le parole dipendenti formando gruppi costituenti, si aggiunge un collegamento diretto tra le parole relazionate a quelle head
 - Ogni relazione rappresenta un tipo di funzione grammaticale, come soggetto, oggetto diretto/indiretto ma anche relazioni più complesse. Il progetto **Universal Dependencies** fornisce un inventario di relazioni di dipendenza presenti nei diversi linguaggi naturali e computazionalmente utili
 - Le relazioni grammaticali si dividono in due gruppi principali:
 1. Relazioni di clausola che descrivono il ruolo rispetto ad un predicato (verbo)

2. Relazioni di modifica che descrivono il modo in cui le parole possono modificare le teste (aggettivi)

- Come per i costituenti, sono state creati dei dependency treebanks, in particolare per linguaggi morfologicamente ricchi come ceco, hindi o finlandese.
 - Il maggior dependency treebank per l'inglese è stato generato dal WSJ corpus del Penn Treebank
 - Algoritmo di parsing **shift-reduce**: si adotta un approccio semplice ed elegante delle CFG
 1. Data una CFG, uno stack ed il testo da parserizzare come lista di tokens
 2. I tokens sono aggiunti alla pila man mano e i primi due elementi della pila sono cercati nella right-hand side delle regole della grammatica. Quando è stato trovato un match, i due elementi sono rimpiazzati (*reduced*) nello stack dal simbolo non-terminale della lhs della regola
 3. Nel caso di un parsing delle dipendenze, invece di aggiungere il simbolo terminale, l'operazione di riduzione introduce una relazione di dipendenza tra la parola e la sua testa o viceversa
 - Algoritmi alternativi con programmazione dinamica: ricerca efficiente nello spazio degli alberi, usando ad esempio le dipendenze come componenti in CKY oppure un'euristica di somma dei punteggi degli archi, come MST
- Question answering:
 - Approccio basato su Information-retrieval (IR): sfrutta la vastità quantità di informazione testuale nel web o in collezioni come PubMed
 - L'obiettivo principale è estrarre la query (**query formulation**), le parole chiavi da utilizzare per trovare i documenti rilevanti
 - Tecniche per trovare i documenti rilevanti, che vengono letti da reti neurali o altre tecniche per trarre una risposta da porzioni di testo
 - L'IR query viene passata all'IR engine che trova i documenti rilevanti ordinati per grado di rilevanza ed è suddivisi in genere in paragrafi o frasi. I passaggi che non contengono sicuramente la risposta vengono scartati.
 - Il passo finale è estrarre la risposta specifica da ogni passaggio (**span labeling**), ad esempio utilizzando l'apprendimento automatico per apprendere un classificatore che decida se una porzione di testo o frase contenga la risposta
 - Approccio knowledge-based: viene costruita una rappresentazione semantica della domanda in forma logica, usata poi come query in database di fatti
 - Nato per rispondere a domande sul BASEBALL usando il database di partite e statistiche
 - I sistemi che mappano una stringa di testo ad una forma logica sono chiamati parser semantici.
 - Ad esempio mappano ad una query SQL
 - I database possono essere relazioni o strutturati come insieme di triplette RDF
 - **Tripletta RDF**: è una 3-tupla costituita da un predicato con 2 argomenti, in modo da esprimere una relazione semplice o una proposizione. "Ada Lovelace - birth place - 1815."
 - La task quindi è trovare una tripletta RDF che risponda ad una domanda che consiste nel trovare l'argomento mancante della tripletta. "When was Ada Lovelace born?" → birth year (Ada Lovelace, ?x)
 - IBM Watson: fornisce un approccio ibrido in cui trova le risposte candidate e valuta la bontà di ciascuna.
 1. Question processing: la domanda viene parsata e le informazioni estratte, come ad esempio il tipo di risposta, relazioni tra i termini o named entity tagging
 - Viene estratto il **focus** della domanda, ovvero la parte correlata alla risposta ed usata ad esempio per trovare il passaggio rilevante nell'IR
 - Il type della risposta fornisce indicazioni sulla semantica della risposta
 2. Candidate answer generation: la domanda parsata viene combinata con documenti esterni ed altre fonti di conoscenza (IMDB, DBpedia etc.) per generare un vettore delle potenziali risposte, contenenti le evidenze di ognuna
 3. Candidate answer scoring: usa diverse fonti di evidenze da assegnare ad ogni risposta candidata
 4. Answer merging and scoring
 - Unisce le risposte equivalenti
 - Sono usati dizionari di sinonimi creati quando termini simili puntano alla stessa pagina Wikipedia oppure, per nomi comuni, possiamo usare la somiglianza morfologica (Word2vec embedding)
 - Unisce anche le evidenze di ciascuna risposta assegnando un valore di confidence per ogni risposta tramite un classificatore appreso precedentemente che assegna la probabilità che la risposta sia corretta
 - Recurrent Neural Networks (RNN):
 - I linguaggi sono fenomeni temporali, le parole di una frase hanno relazioni temporali, mentre molti approcci di ML invece assumono di avere tutti gli aspetti dell'input contemporaneamente per fare sentiment analysis o classificazione
 - L'approccio sliding window per il POS tagging processa il testo usando una finestra di dimensione fissa che slitta sull'input man mano. La decisione presa da una finestra non ha impatto sulle decisioni future
 - Problema: come per l'assunzione di Marjov, non è in grado di estrarre informazione fuori dal contesto/finestra. Nei linguaggi comuni invece le parti di un'informazione possono essere arbitrariamente distanti
 - I RNN sono una classe di reti neurali capaci di tenere in conto dell'aspetto temporale dei linguaggi usando cicli all'interno della rete

- Simple RNN: una normale rete feed-forward in cui il hidden layer è collegato al hidden-layer del passo precedente, fornendo una sorta di memoria, che codifica il processing precedente e informa delle decisioni da prendere più in avanti
 - Il nuovo insieme di pesi U che connette il hidden layer del precedente passo temporale al corrente determina il contesto del passato da usare nel calcolo dell'input corrente
- Stacked RNN: un insieme di molteplici RNN in cui usiamo una sequenza di output da una RNN come input di un'altra RNN
 - Stacked RNN sono più efficaci di una sola RNN in quanto sono capaci di creare rappresentazioni ai livelli di astrazione. I layers iniziali possono creare rappresentazioni più utili dell'input originale da usare come astrazioni per i layers futuri
 - RNN bi-direzionale: poiché il hidden layer rappresenta la conoscenza della rete fino ad un certo punto della sequenza, possiamo vederlo come il contesto della rete da sinistra fino al tempo corrente.
 - Usando una seconda RNN con la sequenza rovescia, possiamo usare anche il contesto da destra. Otteniamo una combinazione bidirezionale usando la **rete forward e backward**
 - Un modo semplice per combinare i due contesti è usare l'addizione o moltiplicazione element-wise o anche la concatenazione per il labeling
 - LSTM e GRU:
 - Nella pratica è difficile apprendere una RNN per usare informazioni distanti dal punto corrente del processing
 - L'informazione codificata negli hidden states tende ad essere locale e più rilevanti per le parti recenti della sequenza e per le decisioni recenti
 - *"The flights the airline was cancelling were full"*
 - Idealmente una rete dovrebbe conservare le informazioni distanti finché non sono necessarie, mentre ancora processa parti intermedie
 - Si richiede alla RNN di fare due cose in contemporanea: mantenere le informazioni per decisioni future e fornisce informazioni utili per la decisione corrente
 - **Long short-term memory**
 - adotta una strategia di gestione del contesto dividendolo in due sottoproblemi:
 1. Rimuovere l'informazione che non è più necessaria nel contesto
 2. Aggiungere l'informazione che probabilmente sarà utile per future decisioni
 - Sfrutta un layer aggiuntivo composto da unità specializzate che usano gates per controllare il flusso di informazione in input ed output dai layers della rete, una sorta di maschera binaria
 - Forget gate: cancella le informazioni del contesto che non sono più utili, calcolando una sommata pesante dello stato del precedente hidden layer e dell'input corrente
 - Add gate: seleziona l'informazione dal hidden layer precedente e dall'input corrente necessaria per la decisione corrente. Tale informazione viene aggiunta al contesto corrente
 - Output gate: decide quale informazione è richiesta per lo stato corrente del hidden layer
 - **Gated Recurrent Units**
 - LSTM introduce un numero considerevole di parametri aggiuntivi alla RNN, causando un alto maggior costo di training
 - GRU riduce il numero di gates a 2:
 - Reset gate: decide quali aspetti del hidden layer precedente sono rilevanti per il contesto corrente e cosa può essere ignorato
 - Update gate: decide quali aspetti del nuovo contesto saranno usati nel hidden layer e quali devono essere preservati per il futuro
 - Machine translation (MT)
 - Si usano reti encoder-decoder che prendono un input e creano una rappresentazione contestualizzata, passato poi al decoder che genera l'opportuna sequenza di output (traduzione in questo caso)
 - Encoder e decoder sono in genere implementati con la stessa architettura come LSTMs o GRUs o architetture stacked di questi
 - Istruiamo una rete a predire la parola successiva in una sequenza, usando un corpus (modelli autoregressive)
 - Condizioniamo a generare le parole seguenti usando sia il hidden state della sequenza prefisso che l'embedding della parola appena generata
 - L'idea geniale della MT è usare coppie di testi (**bitexts**) composte da frasi in lingue differenti di cui una è la traduzione dell'altra (source-target).
 - Queste coppie sono concatenate da un token di fine sequenza e usate come dati di training dal modello autoregressivo
 - Tradurre una frase successivamente si tratta quindi di calcolare il hidden layer state per il source e chiedere di predire parola per parola, man mano, il testo target, ovvero la traduzione.
 - L'encoder genera il hidden state del source come rappresentazione contestualizzata, il decoder utilizza questo stato per autoregressivamente generare l'output.

- I sensori di immagine catturano gli oggetti in una **scena** e creano un'immagine bi-dimensionale.
- All'interno delle camere, l'immagine è formata su un piano rivestito di alogenuro d'argento o una griglia di milioni di pixel fotosensibili.
- Ogni fotone che arriva al sensore produce un effetto la cui forza dipende dalla lunghezza d'onda del fotone
- L'immagine di output è la somma dei fotoni rilevati nella stessa finestra temporale, come media pesata dell'intensità di luce percepita
- Per assicurarsi di vedere un'immagine **a fuoco**, bisogna far sì che i fotoni dello stesso punto della scena arrivino approssimativamente nello stesso punto del **piano immagine**.
- Pinhole camera: un'apertura pinhole, O, nella parte anteriore di un box e un piano immagine nella parte posteriore. Deve essere piccola abbastanza per far sì che l'immagine sia a fuoco
- Le equazioni della **proiezione di prospettiva** definiscono il punto (x, y, z) proiettato sul piano immagine dal punto (X, Y, Z) a distanza f dal pinhole

$$x = \frac{-fX}{Z}, y = \frac{-fY}{Z}$$

Z nel denominatore indica che più l'oggetto è lontano e quindi rimpicciolito nell'immagine.

- Con pinhole piccola arriva poca luce e l'immagine appare scura, con pinhole più grande l'immagine è sfocata perché gli stessi fotoni arrivano su più punti del piano immagine.
 - Si utilizzano **lenti** che focalizzano la luce mantenendo l'immagine in focus, ma solo per oggi all'interno del range di profondità, centrato nel piano focale. Nell'occhio, per cambiare fuoco le lenti cambiano di forma mentre nelle fotocamere si muovono avanti e indietro
- I **bordi** sono linee dritte o curve dell'immagine in cui c'è un cambio significativo di luminosità. L'obiettivo del **rilevamento dei bordi** è ottenere una rappresentazione più compatta e astratta dell'immagine.
- Un modo naïve per rilevare i bordi è calcolare la derivata della luminosità e cercare i punti dove la derivata è grande, cioè c'è più variazione.
 - Spesso non funziona a causa del rumore dell'immagine
 - Appliciamo prima lo smoothing assumendo distribuzione Gaussiana del rumore:
 - **Filtro gaussiano**: assegniamo ad ogni pixel il valore medio dei vicini. I pixel più vicini avranno maggior peso e via via meno peso man mano che aumenta la distanza dal pixel
 - Questo tipo di somma pesata è chiamata **convolution**
 - Per le proprietà delle funzioni convolution si può applicare lo smoothing e il rilevamento dei bordi in un solo passo
 - Una volta rilevati i punti che sono probabili bordi, due pixel di bordo adiacenti con orientamento consistente appartengono alla stessa curva di bordo
- **Object detection** nelle immagini
 - Determinato dove gli oggetti sono localizzati nell'immagine (**object localization**) e a che categoria appartengono (**object classification**)
 - **Informative region selection**: scannerizza l'immagine usando una sliding window a dimensione variabile
 - Le finestre candidate sono tante, il processo è computazionalmente intenso e restituisce molte finestre ridondanti. Allo stesso tempo con poche dimensioni fisse di finestre le regioni sono poco soddisfacenti
 - **Feature extraction**: estrarre le caratteristiche che forniscono una rappresentazione semantica e robusta degli oggetti, come le Histogram of oriented gradients (HOG) sebbene sia difficile usare un descrittore robusto per tutti i tipi di oggetti a causa delle diverse condizioni di apparenza, luce e sfondo.
 - HOG: la tecnica conta le occorrenze dell'orientamento del gradiente nelle celle in cui è divisa un'immagine, nate per rilevare pedoni in strada
 - Difficile a causa di
 - Prospettiva: produce una visione distorta dell'oggetto
 - Lato: gli oggetti hanno aspetto diverso a seconda del letto
 - Occlusione: parti di oggetti possono essere nascoste a causa del punto di vista o di altri oggetti
 - Deformazione: i gradi di libertà interni dell'oggetto cambiano il suo aspetto, come la posizione delle gambe e delle braccia negli umani
 - Due principali strategie:
 - Region proposal based:
 - Processo multi-stage pipeline
 1. La R-CNN (Region-based CNN) propone 2k regioni candidate
 2. Una CNN estrae 4096 caratteristiche come rappresentazione finale per ogni regione
 3. Una SVM multiclasse classifica ciascuna regione e viene fatta una regressione per ottenere il bounding box
 - Se i dati sono pochi le CNN possono essere trainate con Imagenet Large Scale Visual Recognition Challenge (ILSVRC)
 - Molto lento e con tante regioni candidate ridondanti

- Regression/classification based:
 - Passa direttamente dall'immagine in pixel alle coordinate dei bounding box e le probabilità delle classi
 - YOLO (You only look once)
 - Divide l'immagine input in una griglia $S \times S$ ed ogni cella è responsabile per la predizione dell'oggetto centrato in tale cella
 - Ogni cella predice B bounding box e il relativo punteggio di confidence che indica quanto è probabile che il bounding box contenga un oggetto e la confidence della predizione
 - I bounding box e le class probabilities sono combinate insieme per dare i bounding box finali e le classificazioni degli oggetti
 - Maggior efficienza: può operare in real-time a 45fps o addirittura 155fps nella versione semplificata, ma meno accurato per oggetti in primo piano, più accurato sullo sfondo.

Domande primo appello

1. RBFS spiegare in dettaglio l'algoritmo e le sue proprietà (completezza/ottimalità). Spiegare le ragioni computazionali per cui si usa.
2. Parlare delle euristiche e delle loro proprietà e come si usano nei vari algoritmi di ricerca
3. Spiegare in cosa consiste il processo di proposizionalizzazione e come si usa quando bisogna fare inferenza in FOL
4. Spiegare cosa sono i constraint satisfaction problems e illustrare gli algoritmi visti per risolverli
5. Spiegare che ruolo ha dal punto di vista computazionale l'indipendenza condizionale nell'ambito della gestione dell'incertezza
6. Spiegare i vari metodi e algoritmi di syntactic parsing