Utilització de models d'IA

Iloc:	Institut d'Ensenyaments a Distància de les Illes Balears	Imprès per:	Carlos Sanchez Recio
Curs:	Models d'intel·ligència artificial	Data:	dilluns, 28 d'octubre 2024, 16:05
Llibre:	Utilització de models d'IA		

Taula de continguts

1. Introducció

Els problemes d'IA es poden abordar utilitzant diferents enfocaments depenent de la naturalesa del problema a què ens enfrontem. Aquests enfocaments es basen en el concepte d'agent racional. Un agent racional és aquell que fa el que és correcte donat l'objectiu que se li proporciona a l'agent, és a dir, maximitza l'objectiu (meta) amb la informació de què disposa. Aquesta definició es completa enumerant els quatre elements que ha de tenir un agent racional:

- La mesura de rendiment que defineix allò que és correcte (objectiu).
- El coneixement previ que l'agent té del món (entorn).
- Les accions que l'agent pot fer (actuadors).
- Això que l'agent ha percebut fins ara (sensors).

Per exemple, un tipus d'agent racional que solucioni un problema de robot aspirador podria tenir els elements següents:

- Objectiu: maximitzar la superfície neta.
- Entorn: superfície, persones, animals, mobles.
- Actuadors: adreça, accelerador.
- Sensors: càmeres, infrarojos, sistema de posicionament, acceleròmetre, bateria.

El comportament de l'agent anterior és definit per la seva **funció agent**, que és una descripció matemàtica abstracta que relaciona les percepcions rebudes de l'entorn amb accions concretes. La implementació d'aquesta funció sobre algun sistema concret s'anomena **programa agent**.

Hi ha moltes maneres de dotar d'intel·ligència un **agent racional**. Les tècniques més utilitzades són l'**aprenentatge automàtic**, els **sistemes basats en el coneixement** i els **sistemes basats en regles**, entre d'altres. En aquest capítol introduirem en detall els agents racionals, els tipus que hi ha per construir sistemes de resolució de problemes i alguns dels que hi ha orientats a modelitzar sistemes d'IA mitjançant regles.

2. Entorns de treball

La gran quantitat de problemes d'IA que poden sorgir és força àmplia, però, els podem descriure depenent de com siguin els entorns de treball on s'executaran aquests agents, i identificar les dimensions que siguin comunes. Aquestes dimensions determinen el disseny apropiat de l'agent i l'aplicabilitat de cadascuna de les principals famílies de tècniques per implementar-les:

- **Completament observable** o **parcialment observable**: si els sensors d'un agent proporcionen accés a l'estat complet de l'entorn en cada moment, deim que l'entorn és completament observable. Un entorn és completament observable si els sensors detecten tots els aspectes rellevants per a l'elecció de les accions. Si l'agent no té sensors, deim que l'entorn on es troba no és observable.
- **Agent individual** o **multiagent**: si a l'entorn hi ha més d'un agent, aleshores deim que ens trobam en un entorn multiagent; altrament, seria un entorn amb un agent individual. Podem detectar si els altres elements de l'entorn són realment o no agents, en funció de si a través del seu comportament es tracta de maximitzar la mètrica de rendiment donada, el valor de la qual depèn d'un altre agent. Així, un joc d'escacs serà un entorn multiagent (2 agents), mentre que un entorn on calgui resoldre un Sudoku únicament necessitarà un agent.
- **Determinista** o **no determinista**: si l'estat de l'entorn següent està completament determinat per l'estat actual i l'acció executada per l'agent (o agents), aleshores deim que l'entorn és determinista; en cas contrari, és no determinista.
- **Episòdic o seqüencial**: en un entorn episòdic, l'experiència de l'agent es divideix en episodis atòmics. A cada episodi, l'agent percep alguna cosa i realitza una única acció. El més important és que el següent episodi no dependrà de les accions realitzades als episodis anteriors. En canvi, als entorns seqüencials, la decisió actual podria afectar totes les decisions futures. Els entorns episòdics són molt més senzills que els seqüencials, perquè l'agent no necessita pensar en el futur.
- **Estàtic** o **dinàmic**: si l'entorn pot canviar mentre un agent està decidint, aleshores deim que l'entorn és dinàmic per a aquest agent; altrament, és estàtic. Els entorns estàtics són fàcils de manejar perquè l'agent no necessita continuar observant l'entorn mentre decideix una acció, ni s'ha de preocupar pel pas del temps. Els entorns dinàmics, en canvi, no deixen de qüestionar l'agent què vol fer; si encara no ho ha decidit, és com si no hagués decidit fer res.
- **Discret** o **continu**: la distinció entre discret i continu s'aplica a l'estat de l'entorn, a la manera de manejar el temps i a les percepcions i les accions de l'agent. Per exemple, l'entorn dels escacs tendria un nombre finit d'estats diferents i un conjunt discret de percepcions i accions.

Coneixent les dimensions de l'entorn és possible crear entorns simulats que serveixin per experimentar amb l'agent i construir-ne així la funció agent amb l'objectiu d'arribar a implementar programes agent concrets.

Sovint els experiments no es fan per a un únic entorn, sinó per a molts entorns extrets d'una classe d'entorn. Per exemple, per avaluar un conductor de taxi en un trànsit simulat, voldríem fer moltes simulacions amb diferents condicions de trànsit, il·luminació i clima, per la qual cosa ens interessa el rendiment mitjà de l'agent a la classe d'entorn.

3. Sistemes de resolució de problemes

Els problemes a resoldre mitjançant IA poden presentar-se en diferents entorns de treball, els quals poden ser abordats mitjançant agents racionals. En aquesta secció introduirem els tipus de problemes que ens podem trobar als diferents entorns i els tipus de programes agent que podem construir.

3.1. Tipus de problemes en entorns de treball

Els agents poden enfrontar-se a diversos tipus de problemes segons l'entorn de treball. En aquesta secció es fa una classificació dels diferents tipus de problemes.

Cerca d'estat o cerca de la seqüència d'accions

En els problemes de cerca d'estat, només es coneixen les propietats que ha de tenir un estat objectiu, però ni tan sols sabem si hi ha aquest estat. Només necessitam trobar un estat que satisfaci certes restriccions; no importa quina seqüència d'accions ens hi porti. La definició del que és òptim és definida per trobar el **millor estat possible**.

En els problemes de cerca de seqüència d'accions, es coneix l'espai d'estats per endavant, per tant, sabem quins estats són els objectius. En aquest cas, sí que hem de trobar la seqüència d'accions que ens porten a aquests estats objectius. El que és òptim en aquest cas significa trobar la **ruta menys costosa**.

Problemes en línia o problemes fora de línia

En un problema en línia, l'agent no coneix l'espai d'estats i ha de construir un model del mateix mentre actua. D'altra banda, en un problema fora de línia, les percepcions no importen gens, per la qual cosa un agent podria descobrir tota la seqüència d'accions abans de fer res.

Problemes d'absència de sensors

Aquest és un tipus de problema que es dona en entorns deterministes i no observables. En aquest cas, l'agent no sap on és, per això aborden mitjançant conjunts d'estats en què l'agent podria trobar-se.

Problemes de contingència

En aquest tipus de problemes, l'agent no sap quin efecte tindran les accions que faci. Això pot ser perquè l'entorn sigui parcialment observable o no determinista (altres agents afecten l'entorn). Una manera d'abordar aquest tipus de problemes és mitjançant la construcció de la solució mitjançant una llista d'accions que cal executar condicionalment.

3.2. Programes agent

Fins ara els agents racionals s'han descrit a partir del seu comportament, és a dir, l'acció que es du a terme després d'una seqüència donada de percepcions de l'entorn. Perquè aquesta acció sigui duta a terme, necessitem comprendre com funciona internament un agent racional, és a dir, atesa una funció agent, com podem construir un programa agent que la implementi.

Aquest programa agent s'executarà sobre algun dispositiu amb sensors físics i actuadors, el que es coneix com l'arquitectura de l'agent. Així, un agent vindrà definit per una arquitectura i un programa.

El programa agent percebrà com a entrada el que està succeint en el moment actual a l'entorn i actuarà en conseqüència. Si l'acció que l'agent ha de fer depengués de percepcions passades, aleshores l'agent necessitaria desar aquest històric de percepcions en memòria per recuperar-lo i poder-lo revisar.

El programa agent més bàsic que podem dissenyar s'aconsegueix a partir de la construcció d'una taula que relacioni totes les seqüències de percepcions que l'agent pot interpretar amb la corresponent acció. Podem anomenar aquest agent com a agent dirigit per taules. Aquest enfocament per construir programes agent és funcional, encara que no realista; la taula que caldria construir per a un problema tan simple com els escacs (on totes les regles i resultats de les accions estan ben definits) seria enorme, comptant almenys amb 10^{150} files.

Donat l'enorme mida d'aquestes taules, ens trobaríem que no hi ha cap dispositiu físic capaç d'emmagatzemar-les, el dissenyador del programa agent no tindria temps per crear les taules i cap classe d'agent racional seria capaç de trobar les files correctes a partir de l'experiència acumulada. Tot i això, l'agent dirigit per taules és teòricament funcional, assumint que la taula es construeixi correctament.

La implementació en Python d'un programa agent dirigit per taules es pot veure al Llistat 1. El programa agent pren la percepció actual com a entrada dels sensors i torna una acció als actuadors. La taula vindrà donada com un diccionari de parells tupla-acció, sent les tuples la seqüència de percepcions a què l'agent reaccionarà. El programa agent recordarà la seqüència de percepcions, cosa que implicarà augmentar la mida de la taula amb noves files que contemplin les noves seqüències de percepcions. En cas de no existir una acció per a una seqüència donada, l'agent executarà alguna acció per defecte definida a la taula (vegeu les línies 5-6). Com hem comentat, aquest agent és teòricament funcional, sempre que puguem construir una taula que contempli totes les seqüències de percepcions existents per a l'entorn on s'executi l'agent. Així, un agent d'aquest tipus seria factible per a petits dominis.

Llistat 1. Implementació en Python d'un agent basat en taula

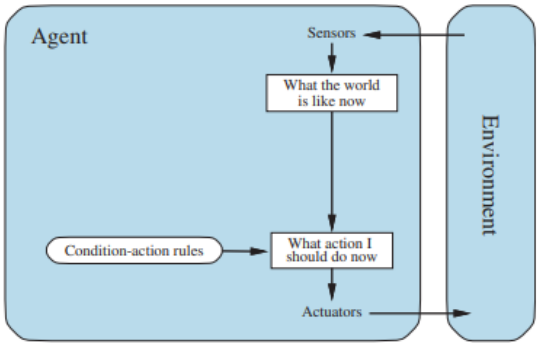
```
1 def table_driven_agent_program(table, current_percept, past_percepts=[]):
2     past_percepts.append(current_percept)
3     percept = tuple(past_percepts)
4     action = table.get(percept)
5     if action is None:
6          action = table.get(tuple())
7     return action
```

A la resta d'aquesta secció descrivim quatre tipus bàsics de programes agents que recullen els principis que fonamenten gairebé tots els sistemes intel·ligents.

A la darrera part, veurem també com qualsevol d'aquests quatre agents pot millorar el seu rendiment mitjançant l'aprenentatge.

3.3. Agents reactius simples

El tipus d'agent més senzill és l'**agent reactiu simple**. Aquests agents decideixen quina acció fer en funció de la percepció actual, ignorant les percepcions anteriors.



Per exemple, l'agent d'un robot aspirador que només pot netejar en dues ubicacions diferents seria un agent reactiu simple, perquè la seva decisió es basa només en la superfície actual i si aquesta superfície conté brutor, és a dir, no té en compte les superfícies que ja ha fet netes (i que podrien estar brutes un altre pic). Al llistat 2 es mostra una possible implementació mínima per a aquest programa agent.

Llistat 2. Implementació d'un programa agent reactiu simple per a un robot aspirador

```
1 def vacuum_reflex_agent_program(location, status):
2      if status == 'Dirty':
3           return 'vacuum'
4      elif location == 'A':
5           return 'right'
6      elif location == 'B':
7           return 'left'
```

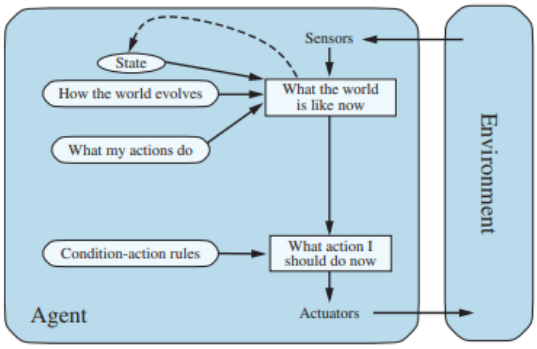
La implementació del programa agent anterior és específic per a un entorn de treball concret al del robot aspirador. Un enfocament més general i flexible és construir primer un intèrpret de propòsit general per a les regles de condició-acció i després crear conjunts de regles per a entorns de treball específics. En aquest programa general, les regles de condició-acció permeten a l'agent fer la connexió entre allò que percep i l'acció que cal fer. Un programa agent que utilitzi aquest intèrpret processaria el que s'ha percebut i executaria la primera regla del conjunt de regles disponible que coincideixi amb la percepció analitzada. La implementació d'aquesta lògica es podria fer utilitzant seqüències d'expressions condicionals if-then-else, circuits de portes lògiques Booleanes, o xarxes neuronals artificials.

Aquests agents tan simples presenten certs problemes en cas que l'agent presenti alguna dificultat a l'hora d'observar l'entorn. Per exemple, suposem que el programa agent del robot aspirador (vegeu el llistat 2) té el sensor d'ubicació trencat i només disposa del sensor de brutícia; aquest agent només tindrà dues percepcions possibles: brut i net. Pot aspirar en resposta a brut, però què hauria de fer en resposta a net? Moure's a l'esquerra fallarà si resulta que comença a la ubicació A, i moure's a la dreta també fallarà si resulta que comença a la ubicació B, donant lloc a un bucle infinit. Aquesta problemàtica sol ser inevitable per a aquest tipus d'agents que operen en entorns parcialment observables. Una possible solució per trencar sortir de possibles bucles infinits és fent que l'agent faci accions aleatòries. Per exemple, si l'agent percep la superfície com a neta, podríem definir que la meitat de vegades l'agent es dirigeixi a l'esquerra i l'altra meitat a la dreta. En entorns multiagent, aquest tipus de comportaments aleatoris pot estar justificat, però no en entorns d'agent individual, on aquests comportaments no solen ser racionals.

3.4. Agents reactius basats en models

Els agents reactius simples no són els més adequats quan ens enfrontam a entorns parcialment observables. Per enfrontar-nos a aquests entorns, podem construir agents que mantinguin un estat intern que depengui de l'històric de percepcions passat i que, per tant, serveixi per reaccionar davant d'algun possible aspecte no observat de l'estat actual.

L'actualització de la informació de l'estat intern a mesura que passa el temps requereix que el programa agent codifiqui dos tipus de coneixement. En primer lloc, necessitem informació sobre com canvia el món al llarg del temps, que es pot dividir aproximadament en dues parts: els efectes de les accions de l'agent i com evoluciona el món independentment de l'agent. Aquest coneixement sobre com evoluciona el món s'anomena model de transició del món. En segon lloc, necessitem informació sobre com es reflecteix l'estat del món a les percepcions de l'agent. Aquest tipus de coneixement s'anomena model de sensor.



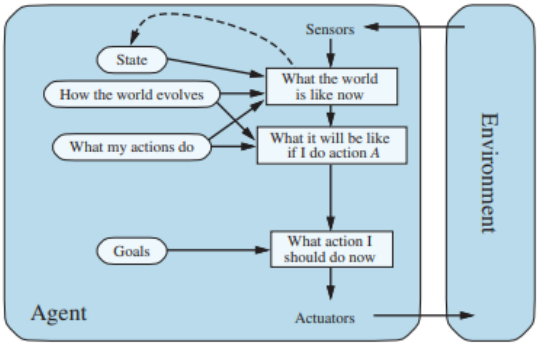
Aquests dos models anteriors permeten que l'agent mantingui un estat intern del món que l'envolta (fins a un cert límit donat pels sensors disponibles de l'agent). Els agents que usen aquests dos models s'anomenen agents reactius basats en models. En aquest tipus d'agents, l'estat intern antic es combina amb el que s'està percebent actualment per generar l'estat actualitzat. De la mateixa manera, al llistat 3 es mostra una possible implementació del programa agent corresponent; a les línies 1-2 es crea el nou estat intern a partir de l'estat actual (*state*), la descripció de com el següent estat dependrà de l'estat actual i l'acció executada (*transition_model*), la descripció de com l'estat del món actual es reflecteix a les percepcions rebudes per l'agent (*sensor_model*), el conjunt de regles de parells condició-acció (*rules*), i l'acció més recent que inicialment no serà cap (*action*).

Llistat 3. Implementació en Python d'un agent reactiu basat en model

```
1 def model_based_reflex_agent_program(percept):
2      state = update_state(state, action, percept,
3           transition_model, sensor_model)
4      rule = rule_match(state, rules)
5      action = rule.action
6      return action
```

3.5. Agents basats en objectius

Saber alguna cosa sobre l'estat actual de l'entorn no és suficient per decidir què fer. A més d'una descripció de l'estat actual, l'agent necessitarà informació sobre l'objectiu que descrigui situacions desitjables. El programa agent pot combinar-la amb el model (la mateixa informació que es va utilitzar a l'agent reactiu basat en el model) per triar les accions que permeten assolir l'objectiu.

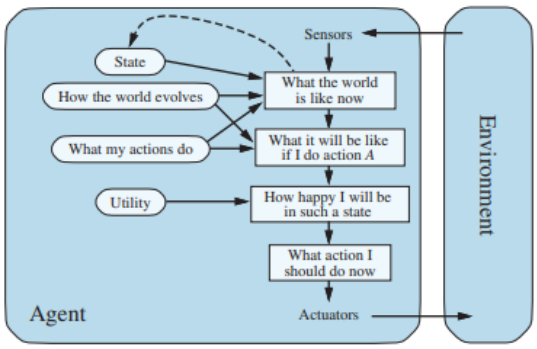


A diferència dels agents reactius, aquest tipus d'agents descriu les accions a realitzar sobre la base d'un objectiu final i la seqüència d'accions que s'han de fer per arribar-hi, en lloc que les accions estiguin relacionades directament amb les percepcions rebudes. Així, encara que puguin semblar menys eficients, resulten més flexibles perquè el coneixement que motiva la presa de decisions està representat explícitament i pot ser modificat. En el cas dels agents reactius, aquest coneixement és específic per a un objectiu concret, i s'ha de modificar manualment en cas que canviem l'objectiu.

Els algorismes de cerca i planificació fan servir agents d'aquest tipus.

3.6. Agents basats en utilitat

Definir objectius no és suficient per produir comportaments d'alta qualitat a la majoria d'entorns. Els objectius només permeten diferenciar entre dos estats: si s'han complert o no. Una mesura de rendiment més general hauria de permetre la comparació entre diferents estats del món segons el grau d'acompliment de l'objectiu, per a la qual cosa es fa servir el terme d'utilitat. Amb aquesta mesura d'utilitat, podríem instruir l'agent que completi el seu objectiu segons algun criteri addicional, en el cas d'un vehicle autònom, triant rutes que siguin més ràpides, segures, fiables o barates.

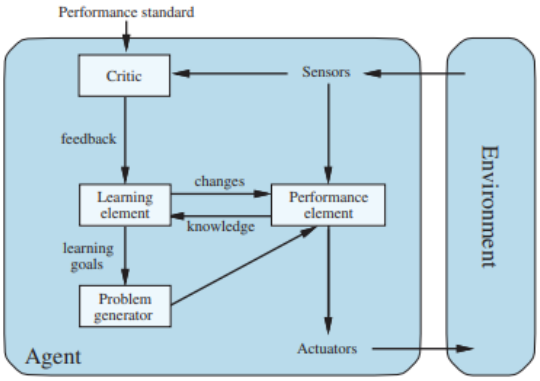


Els agents basats en utilitat presenten alguns avantatges sobre aquells basats en objectius en dos escenaris diferents: i) quan hi ha un conflicte amb la definició dels objectius, la mesura d'utilitat ajudaria a prendre la millor decisió (per exemple, a l'hora de definir un objectiu que contempli alhora una alta velocitat i gran seguretat, en el cas del vehicle autònom), i ii) quan hi ha diversos objectius, però cap d'ells pot ser assolit amb certesa, la mesura d'utilitat proporciona una forma de sospesar la probabilitat d'èxit davant la importància dels objectius.

Fent servir aquesta mesura d'utilitat, l'agent seleccionarà aquella acció que serveixi per assolir la millor utilitat possible.

3.7. Agents que aprenen

Els tipus d'agent que hem vist fins ara poden ser construïts com a **agents que aprenen**. De fet, la majoria de sistemes d'IA actuals tenen algun mecanisme d'aprenentatge automàtic, ja sigui mitjançant aprenentatge supervisat, no supervisat, profund, de reforç o per models probabilístics, entre d'altres. Els agents que aprenen presenten certs avantatges, com permetre a l'agent operar en entorns inicialment desconeguts i fer-se més competent del que el seu coneixement inicial li podria permetre.



Un agent que aprèn es pot dividir en quatre components conceptuals.

- **Element d'aprenentatge**: s'encarrega de modificar el programa per millorar el seu funcionament.
- **Element d'actuació**: s'encarrega de seleccionar les accions externes que s'han de realitzar. Es tracta d'algun dels programes agent vists fins ara.
- **Crítica**: proporciona retroalimentació a l'element d'aprenentatge sobre el rendiment de l'agent i determina com s'hauria de modificar l'element d'actuació per millorar.
- **Generador de problemes**: s'encarrega de suggerir accions que menin a experiències noves i instructives. Si l'element d'actuació pogués, seguiria realitzant les millors accions possibles, però si l'agent està disposat a explorar una mica i realitzar algunes accions una mica pitjors a curt termini, podria descobrir accions molt millors a llarg termini. La feina del generador de problemes és suggerir aquestes accions exploratòries.

D'aquesta manera, els agents que aprenen es poden interpretar com un procés de modificació dels components de l'agent per aconseguir una concordança més gran amb la informació de retroalimentació disponible, de forma que millori el rendiment de l'agent.

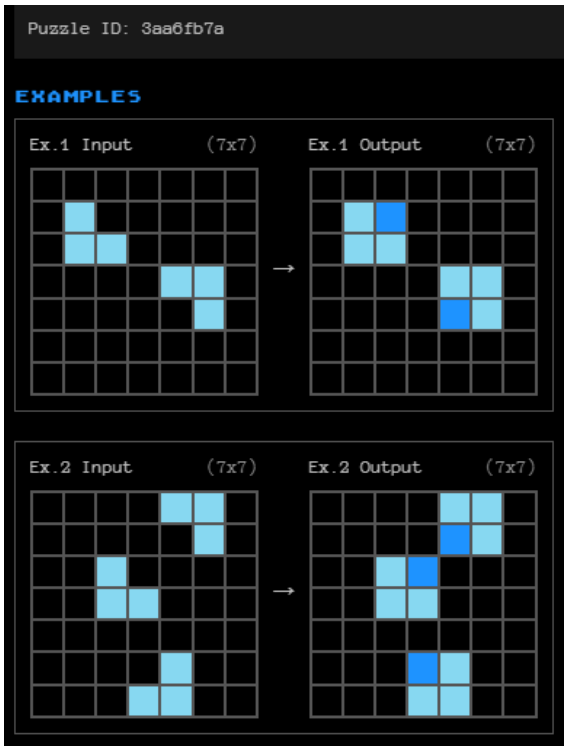
4. Aplicació d'agents: corpus d'abstracció i raonament

En aquesta secció, presentarem el problema de resoldre els puzzles plantejats dins el corpus ARC (Abstraction and Reasoning Corpus). L'investigador de Google François Chollet plantejà l'any 2019 una col·lecció de tasques aparentment simples sobre graelles rectangulars amb cel·les de diferents colors, al seu article On the Measure Of Intelligence.

A continuació, vegem els sis exemples de tasques que ofereix la web de la competició arcprize.org.

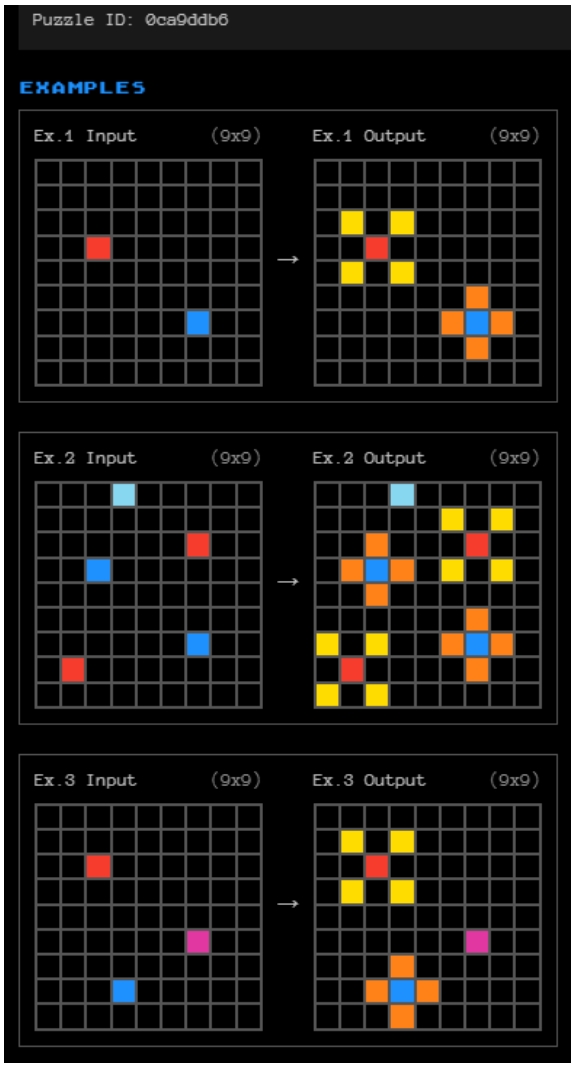
En el primer, l'operació que s'ha de realitzar es pot descriure com completar el quadrat. Quan a l'entrada es presenten tres quadres d'un quadrat de dimensions 2x2, el quadrat que hi manca canvia de negre a blau clar.

Aquest problema es pot resoldre amb un agent reflex simple comparable al de l'aspiradora presentada a l'apartat anterior. Aquí, però, l'entorn que cal observar és més ampli que una sola cel·la. Per decidir si cal pintar de blau clar un quadrat unitari, hi ha d'haver algun dels cantons del seu entorn 3x3 complet.



Llavors que floreixen

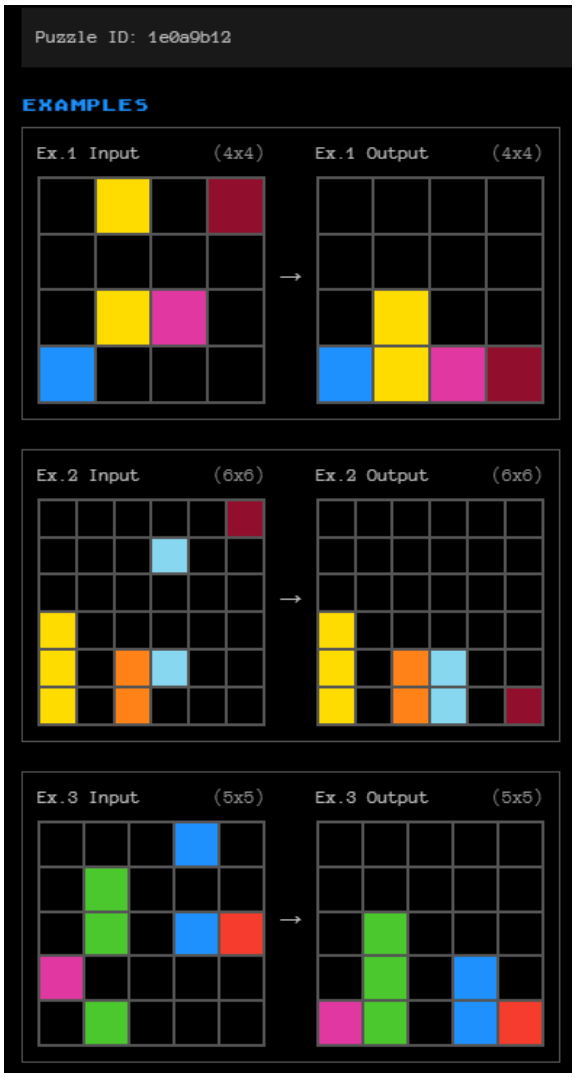
El segon exemple semblen llavors de flors, que segons la seva espècie, indicada pel color, floreixen en una direcció (horitzontal i vertical), en una altra (diagonals) o no floreixen.



En aquests exemples, les formes de la sortida tenen una certa simetria i són com a màxim de 3x3. Un sistema més general ha de funcionar correctament sense aquestes restriccions, amb figures arbitràries de qualsevol mida. Observem també que els quadrats en què s'inscriuen les figures podrien solapar-se encara que les parts acolorides no se sobreposassin. Per exemple, un pètal taronja podria quedar entre dos pètals grocs. Això complica l'aprenentatge automàtic de la transformació. No necessàriament les figures de sortida es poden separar fàcilment pels quadrats que les inclouen.

Tetris

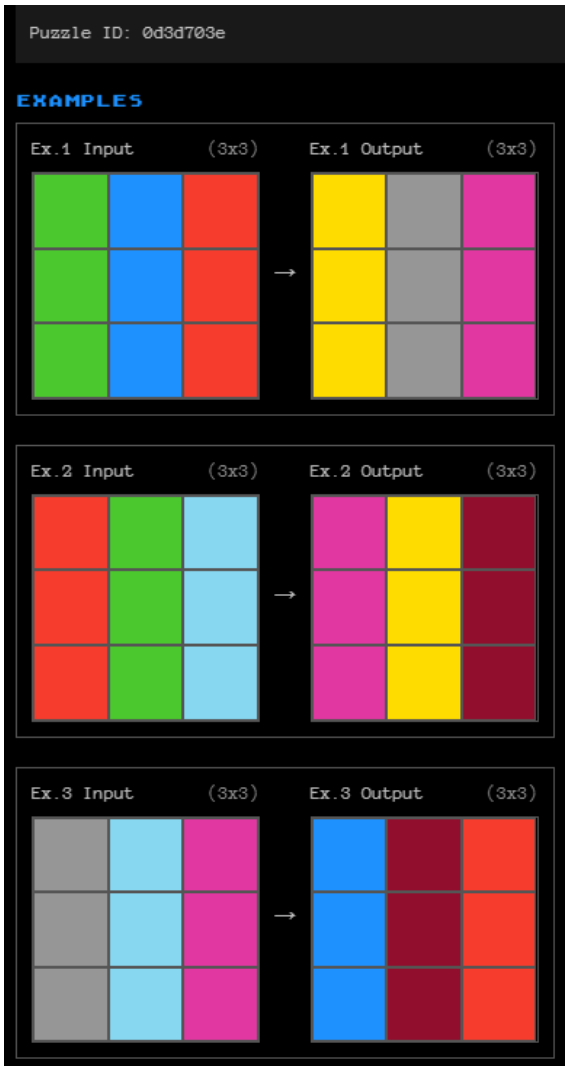
El tercer exemple recorda el famós joc en què blocs de diverses formes i colors cauen fins que topen amb altres blocs.



Si en base a aquests exemples plantejàssim una solució que només funcionàs quan els blocs de la mateixa columna són del mateix color (per exemple comptant-los i col·locant-los tots un damunt l'altre des de baix) aquest sistema fallaria en una situació en què els blocs de cada columna fossin de colors arbitraris.

Transformació puntual

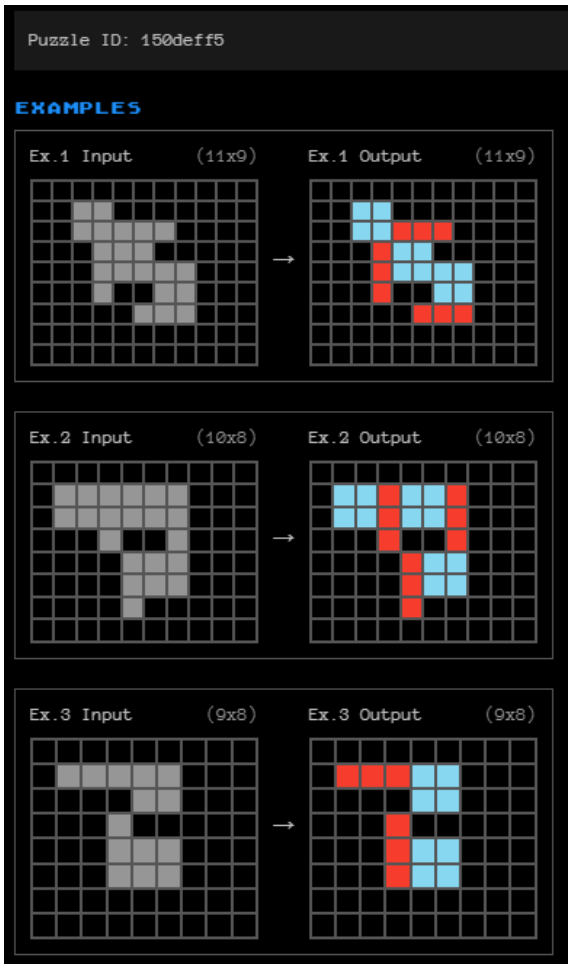
En aquest exemple, tot i que els patrons són de línies verticals, realment es pot pensar com un problema en una dimensió, ja que les diferents fileres sempre són iguals, tant a les graelles d'entrada com de sortida. Observem que les transformacions de colors són simètriques, és a dir, si un color es transforma en un altre, aquest altre també es transforma en l'anterior. Altres transformacions no tenen perquè complir aquesta propietat, i un agent que aprèn a partir de les observacions s'hi ha de poder adaptar.



En una implementació en Python, bastaria tenir un diccionari de colors, que per a cada color d'entrada donàs el seu color de sortida corresponent. Aplicar la transformació de color als nou punts de la graella resol el problema. En un cas més general, la graella podria ser de qualsevol mida. Convendria no utilitzar la propietat de simetria sempre (els colors es transformen l'un en l'altre per parelles) si volem que el sistema pugui funcionar en una situació en que la correspondència de colors sigui més general. Ara bé, si no han aparegut totes les possibilitats de colors d'entrada en els exemples d'entrenament, pot ser raonable suposar aquesta simetria.

Tangram

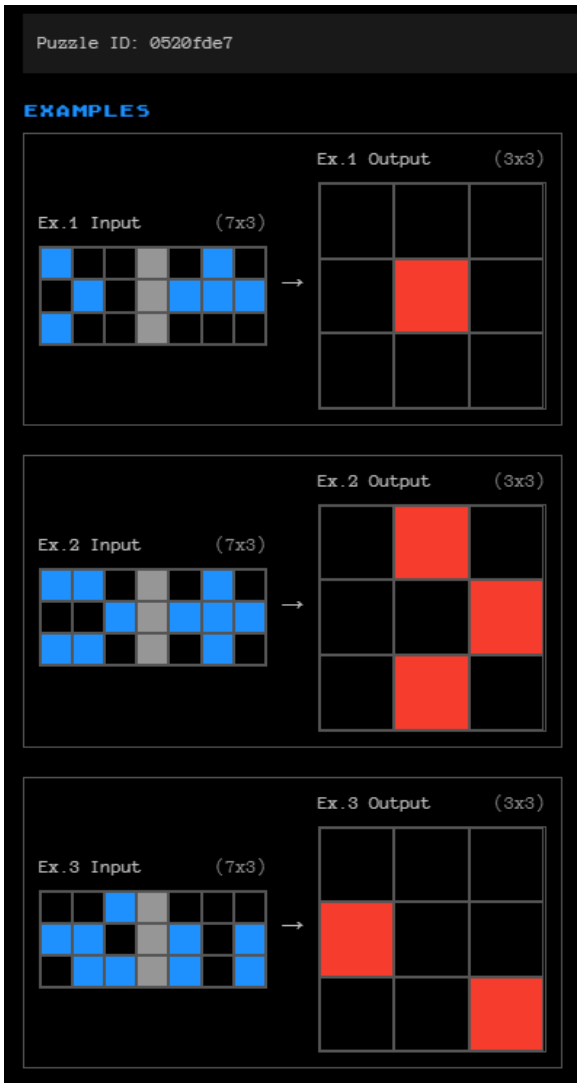
Aquest en principi és el problema més difícil, un trencaclosques en tota regla. Es tracta de trobar quina disposició de les peces possibles, una barreta vermella de tres quadrats o un quadrat cian de dos per dos aconsegueixen formar l'ombra proposada.



Aquest puzzle és el que resoldreu a la tasca, a partir d'uns elements dels quals ja disposareu. El farem només per a les peces d'aquestes dimensions. Un cas més general podria aprendre la forma de les peces a partir de la graella d'entrada, si hi hagués també un exemple en color de cada peça que es pot utilitzar. Seria un sistema molt més versàtil.

Comparació de camps

En aquestes figures, que semblen pistes de tennis, la solució s'obté comparant les graelles de les posicions corresponents. Si a l'entrada els dos quadres són blaus, el quadre corresponent de la sortida és vermell.



La línia divisòria grisa central ens dona una pista que divideix la graella d'entrada en dues seccions de 3x3, de la mateixa mida que la sortida, suggerint qualque mena d'operació entre les dues parts per obtenir el resultat. En aquest cas l'operació és comparació: si les dues cel·les d'entrada són de color, aleshores la cel·la de sortida també ho és. Hi podria haver moltes variants amb diferents funcions, que un sistema flexible ha de ser capaç de deduir sense intervenció humana a partir dels exemples d'entrada.

5. Models de sistemes d'IA

A partir del concepte de programa agent, podem construir sistemes d'IA que resolguin problemes utilitzant diferents enfocaments. A continuació en presentam alguns.

5.1. Planificació automàtica

La **planificació** és la tasca de trobar una seqüència d'accions per aconseguir una fita en un entorn discret, determinista, estàtic i completament observable. En un entorn així, es poden dissenyar agents que apliquin un procés de resolució del problema en quatre passes.

1. **Plantejament de l'objectiu**: els objectius permeten a l'agent organitzar el seu comportament, ja que restringeixen les accions que ha de considerar.
2. **Formulació del problema**: l'agent planteja una descripció dels estats i accions necessaris per aconseguir l'objectiu.
3. **Cerca de solucions**: abans de realitzar cap acció en el món real, l'agent simula seqüències d'accions en el seu model, cercant fins a trobar una seqüència d'accions que assolesqui l'objectiu.
4. **Execució**: l'agent ja pot executar les accions de la solució, d'una en una.

Ara bé, aquest procés planteja dos problemes. D'una banda, cal una heurística pròpia per a cada nou domini: una funció d'avaluació heurística per a la cerca i un codi concret en qualque llenguatge de programació per a l'agent. D'una altra banda, cal representar de forma explícita un espai d'estats exponencialment gran.

A causa d'aquestes limitacions, va sorgir una família de llenguatges de definició del domini de planificació (**PDDL**, *Planning Domain Definition Language*), que permeten expressar totes les accions possibles de forma única i no necessiten el coneixement específic del domini del problema. La sintaxi de PDDL és basada en la de Lisp, un llenguatge fonamentat en llistes. Presentarem PDDL a través d'una abstracció basada en la lògica de primer ordre.

5.2. Lògica de primer ordre

A diferència de la lògica proposicional, que només permet descriure fets que siguin vertaders, falsos o desconeguts, la **lògica de primer ordre** permet, a més, descriure objectes del món i les seves relacions, sigui per alguns o tots els objectes.

La sintaxi en els dos tipus de lògica és comuna. En la lògica proposicional, la sintaxi descriu les sentències que s'admeten. Les sentències atòmiques estan formades per un únic símbol de proposició, que pot ser vertader o fals. Aquests símbols venen representats per qualsevol paraula que comenci per una lletra majúscula o per una única lletra, i poden incloure subíndexs o no, com per exemple, P, Q, $S_{1,3}$. Les sentències complexes es construeixen a partir d'altres sentències més simples, usant parèntesis i operadors anomenats connectors lògics. Normalment se'n fan servir cinc.

- $\neg$ (no): serveix per a negar una sentència. Per exemple, $\neg Q$ es denomina la negació de Q.
- $\wedge$ (i): serveix per a construir conjuncions. Per exemple, $P \wedge Q$ es llegeix P i Q.
- $\vee$ (o): serveix per a construir disjuncions. Per exemple, $P \vee Q$ es llegeix P o Q.
- $\Longrightarrow$ (implica): serveix per a construir implicacions, també anomenades condicionals. Per exemple, $(P \wedge Q) \Longrightarrow R$ es llegeix P i Q implica R. Les implicacions també es coneixen com a regles o sentències *if-then*.
- $\Longleftrightarrow$ (si i només si): permet establir relacions bicondicionals d'equivalència entre dues expressions. Significa que una és vertadera quan l'altra ho és, i a la inversa. Per exemple, la sentència $P \Longleftrightarrow Q$ es llegeix P si i només Q.

Utilitzant aquesta sintaxi podem especificar la semàntica relacionada definint les regles que determinen la veritat d'una sentència respecte d'un model concret. Per exemple, un possible model per a una base de coneixement que usàs els símbols proposicionals P, Q i R podria ser la següent.

$$m = \{P = false, Q = false, R = true\}$$

La llista de regles que indica totes les possibles combinacions de veritat per a un conjunt finit de símbols proposicionals es denomina **taula de veritat**.

D'aquesta forma podem definir i avaluar sentències que siguin més o menys complexes. Per exemple, continuant amb l'exemple i els símbols definits abans, aplicam les regles de veritat per avaluar.

$$\begin{aligned} \neg P \wedge (Q \vee R) &= true \wedge (false \vee true) = true \wedge true \\ &= true \end{aligned}$$

L'avaluació anterior es pot complicar en funció del nombre de símbols i de fets que existesquin, per la qual cosa hi ha diverses tècniques per automatitzar aquesta avaluació, procés que es denomina **inferència**. Aplicant regles d'inferència podem derivar una **prova**, és a dir, una cadena de conclusions que ens permeti aconseguir l'objectiu desitjat. Dues de les regles d'inferència més conegudes són el **modus ponens** i la **simplificació**.

A més d'aquestes regles d'inferència, hi ha un conjunt d'equivalències lògiques que resulta útil per simplificar les sentències i facilitar l'aplicació de les regles d'inferència. Un agent pot treure'n profit a partir d'una base de dades de coneixement que utilitzi sentències en aquesta sintaxi i un motor d'inferència: emmagatzemant sentències sobre l'entorn a la seva base de coneixement, usant el motor d'inferència per obtenir noves sentències i utilitzant aquestes noves sentències per decidir quina acció realitzar.

En el llenguatge de la lògica, els models són les estructures formals que constitueixen els entorns possibles que s'han de considerar. Cada model vincula el vocabulari de les sentències lògiques amb elements de l'entorn, de manera que es pugui determinar la veritat de qualsevol sentència. Així, els models de la lògica proposicional vinculen els símbols proposicionals amb valors de veritat predefinits. Per tant, la lògica proposicional no és capaç d'adaptar-se a entorns de mida il·limitada perquè li manca la capacitat expressiva necessària per tractar de forma concisa el temps, l'espai i les relacions entre els objectes.

En la lògica de primer ordre, en canvi, els models són més realistes, ja que permeten incloure objectes i les seves relacions. El conjunt d'objectes d'un model es denomina **domini** del model. Els elements d'un model es representen mitjançant símbols, que poden ser:

1. constants, per definir els objectes

2. predicats, per definir les relacions
3. funcions, per definir expressions que relacionen una entrada amb una sortida (un altre objecte)

Per exemple, Joan i Pere poden ser símbols constants; *Germà*, *Persona*, *Enginyer*, *AlCap* i *Capell* podrien ser predicats; i *CamaDreta* podria ser una funció. Normalment, ens podem referir als símbols constants i a les funcions com a *termes*, ja que tots dos fan referència a objectes.

A més de tot això, cada model inclou una interpretació i que especifica exactament a quins objectes, relacions i funcions es refereixen els termes i els predicats, és a dir, el coneixement del món real que representen.

A partir d'aquí, podem construir sentències que utilitzin objectes i hi establesquin relacions mitjançant l'ús de predicats seguits d'una llista de termes, per exemple, la sentència *Germà(Joan,Pere)* significarà que Joan i Pere són germans. Un altre exemple de sentència podria ser *Casats(Pare(Joan),Mare(Pere))*, que significaria que el pare d'en Joan i la mare d'en Pere són casats. També podem fer servir connectors lògics per construir sentències més complexes com, per exemple, la següent.

$$Germa(Joan, Pere) \land Germa(Pere, Joan)$$

Les relacions entre els objectes són consistents entre les sentències d'exemple, per la qual cosa podem dir que totes les sentències són certes.

Finalment, es poden usar dos quantificadors a la lògica de primer ordre, que poden expressar propietats sobre col·leccions d'objectes completes. El primer és el **quantificador universal**, expressat com a $\forall$, i que permet establir regles com la següent.

Tots els enginyers són persones.

$$\forall x Enginyer(x) \implies Persona(x)$$

En aquest cas el símbol x és una variable que podria prendre el valor de qualsevol terme del domini. La sentència anterior serà certa si la mateixa sentència sense quantificar és veritat per a tots els termes del domini. Així, podríem reescriure aquesta sentència com la conjunció de totes les sentències possibles del domini:

$$(Enginyer(Joan) \implies Persona(Joan))$$
$$\land Enginyer(Pere) \implies Persona(Pere) \land \ldots$$

L'altre quantificador disponible és el **quantificador existencial**, expressat com a $\exists$, *per a algun*, i que serveix per a establir regles com la següent.

En Pere té algun germà.

$$\exists x Persona(x) \land Germa(x, Pere)$$

La sentència anterior serà vertadera si la mateixa sentència sense quantificar és veritat per a algun dels termes del domini.

$$Persona(Pere) \land Germa(Pere, Pere) \lor Persona(Joan)$$
$$\land Germà(Joan, Pere) \lor \ldots$$

La lògica de primer ordre també inclou una altra forma de construir sentències. Es pot utilitzar el símbol d'igualtat $=$ per indicar que dos termes es refereixen al mateix objecte i d'aquesta forma poder establir fets, per exemple,

$$Germa(Joan) = Pere$$

De la mateixa manera, també es pot definir la desigualtat, per exemple,

$$\exists x, y Germa(x, Pere) \land Germa(y, Pere) \land \neg(x = y)$$

per indicar que en Pere té com a mínim dos germans distints. La notació $x \neq y$ també es pot usar com a abreviació de $\neg(p = q)$.

Després d'aquesta breu introducció a la sintaxi de la notació lògica, podem continuar la secció següent sobre PDDL.

5.3. Planificació amb PDDL

La planificació automatitzada es refereix a la realització d'estratègies o seqüències d'acció, normalment per a l'execució per agents intel·ligents, robots autònoms i vehicles no tripulats. A diferència dels problemes clàssics de control i classificació, les solucions són complexes i s'han de descobrir i optimitzar en l'espai multidimensional.

En entorns coneguts amb models disponibles, la planificació es pot fer prèviament, fora de línia. Les solucions es poden trobar i avaluar abans d'executar-se. En canvi, en entorns dinàmics desconeguts, l'estratègia sovint s'ha de revisar mentre s'executa. En aquest cas s'han d'adaptar models i polítiques. Les solucions solen recórrer a processos d'assaig i error iteratius, usant programació dinàmica, aprenentatge de reforç i optimització combinatòria. Els llenguatges utilitzats per descriure la planificació i la programació sovint s'anomenen llenguatges d'acció.

Els llenguatges més usats per representar dominis i problemes de planificació, com STRIPS i PPDL, es basen en variables d'estat. Cada situació possible de l'entorn és una assignació de valors a les variables d'estat. Les accions determinen com els valors de les variables d'estat canvien quan es pren l'acció. Com que un conjunt de variables d'estat introdueix un espai de variables d'estat exponencial en el conjunt, la planificació pateix el problema de l'explosió combinatòria.

La sortida del planificador és un pla ordenat parcialment o totalment. Vegem quins són els elements que s'especifiquen en PDDL (*Planning Domain Definition Language*)

1. La descripció del domini consisteix en una definició del nom de domini, definició dels requisits, definició de la jerarquia de tipus d'objectes, definició dels objectes constants, definició de predicats, i també la definició d'accions possibles. Les accions tenen paràmetres, precondicions i efectes. Els efectes de les accions podrien ser condicionals.
2. La descripció del problema consisteix en una definició del nom del problema, la definició del nom de domini relacionat, tots els possibles objectes, les condicions inicials i els estats objectiu.

A continuació hi ha un exemple de PDDL en què es determina la ruta òptima d'avions entre aeroports que han de transportar unes determinades càrregues. A la tasca d'aquest lliurament executareu un planificador damunt aquestes definicions.

```
(define (domain transport)
    (:predicates
        (A ?c ?r)
        (Dins ?c ?a)
        (Carrega ?c)
        (Avio ?a)
        (Aeroport ?r))

    (:action Carregar
        :parameters (?c ?a ?r)
        :precondition (and (A ?c ?r) (A ?a ?r) (Carrega ?c) (Avio ?a) (Aeroport ?r) )
        :effect(and (not(A ?c ?r)) (Dins ?c ?a)  ) )


    (:action Descarregar
        :parameters (?c ?a ?r)
        :precondition (and (Dins ?c ?a) (A ?a ?r) (Carrega ?c) (Avio ?a) (Aeroport ?r) )
        :effect(and (A ?c ?r) (not (dins ?c ?a))))

    (:action Volar
        :parameters (?a ?origen ?desti)
        :precondition (and (A ?a ?origen) (Avio ?a) (Aeroport ?origen) (Aeroport ?desti))
        :effect (and (not(A ?a ?origen)) (A ?a ?desti))))
```

```
(define (problem transport-mad-bcn)
    (:domain transport)
    (:objects A1 A2 C1 C2 MAD BCN)
    (:init
        (Avio A1)
        (Avio A2)
        (Carrega C1)
        (Carrega C2)
        (Aeroport MAD)
        (Aeroport BCN)
        (A A1 MAD)
        (A A2 BCN)
        (A C1 MAD)
        (A C2 BCN))
    (:goal (and (A C1 BCN) (A C2 MAD))))
```

6. Raonament imprecís

En aquest apartat tractam com la lògica difusa permet raonar a partir de descripcions vagues, imprecises.

Per començar, descrivim la lògica difusa, basada en els conjunts difusos, i els operadors estàndard de Zadeh.

A continuació, desenvolupam una altra família d'operadors difusos, basats en una funció derivable. Això permet que les funcions obtingudes siguin més suaus.

Finalment, mostram un exemple implementat usant la llibreria **scikit-fuzzy**.

6.1. Vaguetat: conjunts difusos i lògica difusa

La **teoria de conjunts difusos** és una manera d'especificar com de bé un objecte satisfà una descripció vaga. Per exemple, considerem la proposició "n'Aina és alta". Això és cert si n'Aina fa 1,75 m? La majoria de persones dubtarien si respondre **cert** o **fals**, i potser s'estimarien més dir **més o manco**. Fixem-nos que això no és una qüestió d'incertesa sobre el món extern, sabem exactament quina és l'alçada de n'Aina. La qüestió és que el terme lingüístic **alt** no es refereix a una classificació precisa d'objectes en dues classes, sinó que hi ha graus d'alçada. En aquest sentit, la teoria de conjunts difusos no és un mètode per al raonament amb incertesa. En canvi, la teoria de conjunts difusos tracta "alt" com un predicat difús i diu que el valor de veritat de la funció **Alt(Aina)** és un **nombre entre 0 i 1**, en comptes de ser simplement cert o fals. El nom **conjunt difús** ve de la interpretació del predicat com una definició implícita del conjunt dels seus membres, un conjunt que no té unes fronteres precises.

La **lògica difusa** és un mètode per raonar amb expressions lògiques descrivint la pertinença a conjunts difusos. Per exemple, la sentència complexa **Alta(Aina) i Rica(Aina)** té un valor de veritat que és funció del valor de veritat dels seus components. Aquestes són les regles estàndard per avaluar la veritat difusa, T, d'una sentència complexa. Són els anomenats operadors de Zadeh. **Lotfi Zadeh** va ser l'introductor de la matemàtica difusa, començant amb el seu primer article de 1965 **Fuzzy sets** (conjunts difusos).

conjunció, **and** lògica: A i B

$$T(A \wedge B) = min(T(A), T(B))$$

disjunció, **or** lògica: A o B

$$T(A \vee B) = max(T(A), T(B))$$

negació, **not** lògica: no A

$$T(\neg A) = 1 - T(A)$$

La lògica difusa, per tant, és un sistema de veritat funcional, i això causa dificultats serioses. Per exemple, suposem que T(Alta(Aina)) = 0.6 i T(Rica(Aina)) = 0.4. Aleshores tindrem que T(Alta(Aina) i Rica(Aina)) = 0.4, cosa que sembla raonable. En canvi, també obtindrem el resultat T(Alta(Aina) i no Alta(Aina)) = 0.4, cosa que no sembla tan raonable. El problema sorgeix de la incapacitat d'una aproximació de veritat funcional de tenir en compte les correlacions i anticorrelacions entre les proposicions que componen una proposició complexa.

El control difús és una metodologia per construir sistemes de control en què la transformació entre els valors reals d'entrada i els paràmetres de sortida es representa mitjançant regles difuses. El control difús ha tingut molt d'èxit en productes comercials, com ara transmissions automàtiques, càmeres de vídeo o afaitadores elèctriques. Els crítics amb la lògica difusa (per exemple, Elkan) argumenten que aquestes aplicacions tenen èxit perquè tenen bases de regles petites, no hi ha encadenament d'inferències i hi ha paràmetres ajustables que poden millorar el funcionament del sistema. El fet que facin servir operadors difusos pot ser coincidència amb el seu èxit: la clau pot ser que donen una forma concisa i intuïtiva d'especificar una funció real interpolada suaument.

S'ha intentat explicar la lògica difusa en termes de teoria de la probabilitat, que poden capturar aspectes de la imprecisió sense introduir el problema dels graus de veritat. Així i tot, queden molts de temes oberts pel que fa a la representació de les observacions lingüístiques i les quantitats contínues. Aquests problemes, fora de la comunitat de lògica difusa, han estat majoritàriament obviats.

6.2. Construcció de sistemes de lògica difusa

L'esquema general per construir aquest tipus de sistemes té quatre passes ben diferenciades, d'acord amb el mètode de Mamdani.

1. Modelatge, en què es defineixen les variables i els seus dominis;
2. Fuzzificació, en què es mesura el grau de pertinença de les variables d'entrada a conjunts difusos;
3. Motor d'inferència, en què s'efectua el raonament;
4. Deffuzzificació, pas invers a la fuzzificació, només si es desitja un valor de sortida real del sistema.

6.3. Modelització

En aquest pas es defineixen les variables d'entrada, les variables de sortida i el domini de definició per a cada variable. El domini de definició és el rang de valors numèrics que pot prendre la variable i els conjunts difusos. Els conjunts difusos estaran formats per rangs de valors associats amb una etiqueta lingüística (denominada *variable difusa*).

Computacionalment, els conjunts difusos se solen representar mitjançant funcions d'ajust lineals en comptes d'altres de més complexes per raons de rendiment. Així, el conjunt difús de les persones amb alçada alta es pot definir amb els quatre punts $(0/170, 1/175, 1/180, 0/185)$, on el separador $/$ serveix per a associar el grau de pertinença amb l'altura a l'eix horitzontal.

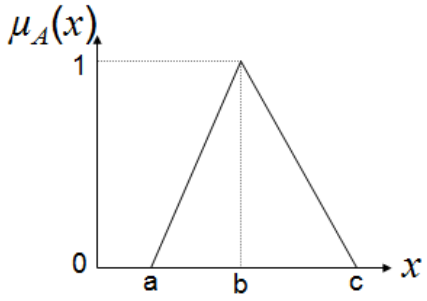
Tradicionalment, els conjunts difusos s'han definit a partir de l'opinió d'un o més experts en el domini del problema. Una tècnica més moderna es basa en l'ús de xarxes neuronals, que aprenen a partir de les dades disponibles del funcionament del sistema i després deriven els conjunts difusos automàticament.

6.4. Fuzzificació

En aquest pas es converteixen les variables d'entrada en graus de pertinença que serviran per quantificar el grau de possessió cap a la seva corresponent variable difusa. A l'exemple de l'alçada, aquestes variables representen les etiquetes *baix*, *alt*, *molt alt*, etc. Per exemple, si una persona fa 164 cm, podem dir que aquest valor d'alçada és entre els conjunts baix i mitjà, però amb una pertinença major al conjunt mitjà. A la pràctica, s'utilitzen funcions de pertinença per obtenir els graus de pertinença en funció de la forma que tenguin els conjunts del domini de definició.

Per exemple, la funció de pertinença d'una entrada x per a un conjunt difús triangular A definit per un límit inferior a, un límit superior b i un valor m, on $a < m < b$, ve donada de la forma següent.

$$\mu_A(x) = \begin{cases} 0 & if\ x \le a \\ \frac{x-a}{b-a} & if\ a \le x \le b \\ \frac{c-x}{c-b} & if\ b \le x \le c \\ 0 & if\ x \ge c \end{cases}$$



Imatge: *Funció de pertinença triangular*

D'una altra banda, per al cas del conjunt difús trapezoidal es defineix un límit inferior a, un límit superior d, un límit de suport inferior b i un límit de suport superior c on $a < b < c < d$.

Els conjunts difusos es poden alterar usant funcions de cobertura, relacions entre paraules (molt, bastant, moltíssim) i funcions concretes que serviran per pronunciar o suavitzar més o més poc els pendents de les rectes que formen els conjunts.

Es poden manipular els conjunts difusos aplicant-hi operacions de la teoria clàssica de conjunts, com ara el complement, subconjunts, intersecció o unió.

Independentment de la variable d'entrada, la suma de totes les pertinences als conjunts sempre valdrà 1. A l'exemple de l'alçada i amb funcions de pertinença trapezoidals, podem veure com el valor de 164 cm té una pertinença de 0.2 al conjunt de l'alçada baixa i de 0.8 al conjunt de l'alçada mitjana. Es verifica que $0.2 + 0.8 = 1$. Això significa que una persona amb aquesta alçada tendria una pertinença parcial a diversos conjunts.

6.5. Inferència

Després d'haver fuzzificat les variables d'entrada i d'haver determinat el grau de pertinença de les variables als conjunts difusos, es fa el raonament per a la presa de decisions. Això es pot aconseguir mitjançant la generació de regles difuses del tipus if-then: SI \(x \) és A, ALESHORES \(y \) és B, on \(x \) i \(y \) són variables difuses i \(A \) i \(B \) són valors lingüístics determinats pels conjunts difusos definits pels rangs de valors possibles; la part del **SI** de la regla s'anomena **antecedent** i la part de l'**ALESHORES** s'anomena **conseqüent**.

La diferència entre les regles clàssiques i les regles difuses és que les primeres utilitzen valors discrets per definir les condicions, mentre que les segones les defineixen amb valors difusos. Per exemple, podem definir una regla clàssica

SI velocitat > 100 km/h, ALESHORES distància_de_frenada = 250 m

o una regla difusa

SI velocitat és ràpida, ALESHORES distància_de_frenada és llarga

La segona regla defineix els possibles valors difusos de la *velocitat* com a *lenta*, *mitjana* i *ràpida*; i els possibles valors de la distància_de_frenada com a *curta*, *mitjana* i *llarga*, de forma que les regles difuses relacionen conjunts difusos.

Els sistemes de raonament imprecís basats en lògica difusa combinen les regles possibles i redueixen el nombre de regles que cal codificar en més d'un 90% respecte dels sistemes tradicionals basats en regles clàssiques.

Seguint amb l'exemple de l'alçada, podem construir la regla següent.

SI alçada és {molt baixa, baixa} i pes és {alt, sobrepès, obesitat} ALESHORES RecomanarDieta {estricta}

Aquesta regla s'activarà quan el grau de pertinença de l'alçada sigui més gran que \(0 \) per als conjunts *molt baixa* o *baixa*, i simultàniament el pes sigui algun d'aquests tres: *alt*, *sobrepès* o *obesitat*. Quan s'activi aquesta regla, aleshores el sistema executarà l'acció *RecomanarDieta* amb un determinat grau de pertinença a *estricta*.

Els antecedents de les regles difuses es poden avaluar associant funcions als operadors booleans (\(AND\), \(OR\)...). Habitualment, la funció associada a l'operador \(OR \) se sol avaluar com el màxim dels valors donats, mentre que l'associada a l'operador \(AND \) correspon al mínim dels valors donats.

Una vegada que s'han avaluat totes les regles, s'agreguen els resultats obtinguts unificant-los, és a dir, produint un conjunt difús final.

6.6. Desfuzzificació

Aquesta darrera passa és opcional, i només fa falta en el cas que vulguem obtenir un valor discret de sortida al sistema. Els mètodes més habituals són aquests.

1. Prendre el centre de masses del conjunt difús de la sortida.
2. Prendre el valor de la màxima pertinença del conjunt difús de la sortida.

El primer cas es pot estimar simplement prenent una mostra de punts de l'eix horitzontal del conjunt difús, sumant cada sèrie de mostres de forma individual i multiplicant-la pel seu grau de pertinença, i dividint finalment entre la suma de tots els graus de pertinença utilitzats.

6.7. Exemple: propina

En aquest quadern de Colab, Fuzzy tip, hi ha implementat un exemple complet d'aplicació de lògica difusa.

Es tracta de determinar una quantitat de propina adequada en funció de la qualitat del servei i la qualitat del menjar. L'interès de la lògica difusa aquí és que permet definir una funció que acaba sent numèrica a partir de paraules, que consisteixen en descripcions vagues, imprecises, relacionades amb les quantitats.

7. Bibliografia

https://www.mathworks.com/help/fuzzy/what-is-fuzzy-logic.html

https://pypi.org/project/simpful/