ÖSSZEFOGLALÁS

Az MI az intelligens gondolkodás számítógépes reprodukálása szempontjából hasznos elveket, módszereket, technikákat kutatja, fejleszti, rendszerezi.

Miről ismerhető fel az MI?

- Megoldandó feladatai: nehezek. A feladat problématere hatalmas, a megoldás megkeresése kellő intuíció hiányában kombinatorikus robbanáshoz vezethet.
- A megoldást biztosító szoftverek intelligensen működnek. Úgy, mintha a háttérben egy ember tevékenykedne (Turing teszt), továbbá gyakran tanulnak működésük közben, és ennek következtében javulhat az eredményességük és hatékonyságuk.
- Megoldási módszereire az átgondolt reprezentáció és heurisztikával megerősített algoritmusok használata jellemző.

Sok feladat fogalmazható át útkeresési problémává úgy, hogy a feladat modellje alapján megadunk egy olyan élsúlyozott irányított gráfot, amelyben adott csúcsból adott csúcsba vezető utak jelképezik a feladat egy-egy megoldását. Ezt a feladat *gráfreprezentációjának* is szokás nevezni, amely magába foglal egy úgynevezett δ*-gráfot* (olyan élsúlyozott irányított gráf, ahol egy csúcsból kivezető élek száma véges, és az élek költségére megadható egy δ pozitív alsó korlát), az abban kijelölt startcsúcsot és egy vagy több célcsúcsot. Ebben a reprezentációs gráfban keresünk egy startcsúcsból kiinduló célcsúcsba futó utat, esetenként egy legolcsóbb ilyet. A reprezentációs gráffal megfogalmazott feladatok *problématere* (a megoldandó probléma lehetséges válaszainak halmaza) többnyire a startcsúcsból kiinduló utak halmaza. Néha a problématér a reprezentációs gráf csúcsainak halmaza. (lásd n-királynő probléma)

Állapottér-reprezentáció – egy olyan (de nem az egyetlen) módszer a feladatok modellezésére, amelyet aztán természetes módon lehet gráfreprezentációkét is megfogalmazni.

Négy eleme van:

- Állapottér, amely a probléma homlokterében álló adat (objektum) lehetséges értékeinek (állapotainak) halmaza. Gyakran egy alaphalmaz, amelyet egy alkalmas invariáns leszűkít.
- Műveletek (előfeltétel+hatás), amelyek állapotból állapotba vezetnek.
- Kezdőállapot(ok) vagy azokat leíró kezdőfeltétel.
- Célállapot(ok) vagy célfeltétel.

Az *állapot-gráf* (egy speciális reprezentációs gráf) az állapotokat, mint csúcsokat, a műveletek hatásait, mint éleket tartalmazza.

Keresések

Egy általános *kereső rendszer* részei: a *globális munkaterület* (a keresés memóriája), a *keresési szabályok* (a memória tartalmát változtatják meg), és a *vezérlési stratégia* (adott pillanatban alkalmas szabályt választ). A vezérlési stratégiának van egy általános, elsődleges eleme, lehet egy másodlagos (az alkalmazott reprezentációs modell sajátosságait kihasználó) eleme és a konkrét feladatra építő eleme. Ez utóbbi a *heurisztika*, a konkrét feladatból származó extra ismeret, amelyet közvetlenül a vezérlési stratégiába építünk be az eredményesség és a hatékonyság javítása céljából.

Lokális keresések

A keresés egyetlen csúcsot (és annak környezetét) látja (tárolja a globális munkaterületen), amelyet minden lépésben a szomszédjai közül vett lehetőleg „jobb" gyerekcsúccsal cserél le (ez a csere a keresési szabály). A vezérlési stratégia a „jobbság" eldöntéséhez *rátermettségi függvényt* használ, amely olyan heurisztikára épül, ami várhatóan annál jobb értéket ad egy csúcsra, minél közelebb esik az a célhoz. Mivel a keresés „elfelejti", hogy honnan jött, a döntések nem vonhatók vissza, ez egy *nem-módosítható vezérlési stratégia*.

Lokális kereséssel megoldható feladatok azok, ahol egy lokálisan hozott rossz döntés nem zárja ki a cél megtalálását. Ehhez vagy egy erősen összefüggő reprezentációs-gráf, vagy jó heurisztikára épített célfüggvény kell. Jellemző alkalmazás: adott tulajdonságú elem keresése, függvény optimumának keresése.

- *Hegymászó algoritmus*: Minden lépésben az aktuális csúcs legjobb gyermekére lép, de kizárja a szülőre való visszalépést. Zsákutcába (aktuális csúcsból nem vezet ki él) beragad, körök mentén végtelen ciklusba kerülhet, ha a rátermettségi függvény nem tökéletes.
- *Tabu keresés*: Az aktuális csúcson (n) kívül nyilvántartja még az eddig legjobbnak bizonyult csúcsot (n^*) és az utolsó néhány érintett csúcsot; ez a (sor tulajdonságú) *tabu halmaz*. Minden lépésben az aktuális csúcs gyermekei közül, kivéve a tabu halmazban levőket, a legjobbat választja új aktuális csúcsnak, (ezáltal felismeri a tabu halmaz méreténél nem nagyobb köröket), frissíti a tabu halmazt, és ha n jobb, mint az n^*, akkor n^*-ot lecseréli n-re.
- *Szimulált hűtés algoritmusa*: A következő csúcs választása véletlenszerű. Ha a kiválasztott csúcs (r) célfüggvény-értéke jobb, mint az aktuális csúcsé (n), akkor odalép, ha rosszabb, akkor az új csúcs elfogadásának valószínűsége fordítottan arányos $f(n) - f(r)$ különbséggel. Ez az arány ráadásul folyamatosan változik a keresés során: ugyanolyan különbség esetén kezdetben nagyobb, később kisebb valószínűséggel fogja a rosszabb értékű r csúcsot választani.

Visszalépéses keresések

A startcsúcsból az aktuális csúcsba vezető utat (és az arról leágazó még ki nem próbált éleket) tartja nyilván (globális munkaterületen), a nyilvántartott út végéhez egy új (ki nem próbált) élt fűzhet vagy a legutolsó élt törölheti (visszalépés szabálya), a visszalépést a legvégső esetben alkalmazza. A visszalépés teszi lehetővé azt, hogy egy korábbi továbblépésről hozott döntés megváltozhasson. Ez tehát egy *módosítható vezérlési stratégia*. A keresésbe sorrendi és vágó heurisztika építhető. Mindkettő lokálisan, az aktuális csúcsból kivezető, még ki nem próbált élekre vonatkozik.

Visszalépés feltételei: zsákutca, zsákutca torkolat, kör, mélységi korlát.

- VL1 (nincs kör- és mélységi korlát figyelés) véges körmentes irányított gráfokon terminál, és ha van megoldás, akkor talál egyet.
- VL2 (általános) δ-gráfokon terminál, és ha van megoldás a mélységi korláton belül, akkor talál egyet.

Könnyen implementálható, kicsi memória igényű, mindig terminál, és ha van (a mélységi korlát alatt megoldási út), akkor megtalál egyet. Nem garantál optimális megoldást, egy kezdetben hozott rossz döntést csak nagyon sok lépés után képes korrigálni és egy zsákutca-szakaszt többször is bejárhat, ha abba többféle úton is el lehet jutni.

Gráfkeresések

A globális munkaterületén a startcsúcsból kiinduló már feltárt utak találhatók (ez az ún. *kereső gráf*), külön megjelölve az utak azon csúcsait, amelyeknek még nem (vagy nem eléggé jól) ismerjük a rákövetkezőit. Ezek a *nyílt csúcsok*. A keresés szabályai egy nyílt csúcsot terjesztenek ki, azaz előállítják (vagy újra előállítják) a csúcs összes rákövetkezőjét. A vezérlési stratégia a legkedvezőbb nyílt csúcs kiválasztására törekszik, ehhez egy *kiértékelő függvényt* (*f*) használ. Mivel egy nyílt csúcs, amely egy adott pillanatban nem kerül kiválasztásra, később még kiválasztódhat, ezért itt egy módosítható vezérlési stratégia valósul meg.

A keresés minden csúcshoz nyilvántart egy odavezető utat (π visszamutató pointerek segítségével), valamint az út költségét (*g*). Ezeket az értékeket működés közben alakítja ki, amikor a csúcsot először felfedezi vagy később egy olcsóbb utat talál hozzá. Mindkét esetben (amikor módosultak a csúcs ezen értékei) a csúcs nyílttá válik. Amikor egy már korábban kiterjesztett csúcs újra nyílt lesz, akkor a már korábban felfedezett leszármazottainál a visszafelé mutató pointerekkel kijelölt út költsége nem feltétlenül egyezik majd meg a nyilvántartott *g* értékkel, és az sem biztos, hogy ezek az értékek az eddig talált legolcsóbb útra vonatkoznak. Csökkenő kiértékelő függvényt (egy csúcs *f* értéke nem nő az algoritmus működése során, sőt, amikor egy csúcs visszakerül a nyílt csúcsok közé, akkor annak új *f* értéke kisebb annál, mint amivel onnan korábban kikerült) használva viszont a küszöbcsúcsok (olyan csúcs, amely *f* értéke minden korábban kiterjesztett csúcs *f* értéknél nagyobb vagy egyenlő) kiterjesztésének pillanatában ilyen inkonzisztencia garantáltan nem áll fenn.

- Nem-informált gráfkeresések: *mélységi gráfkeresés* (*f = -g,* minden *(n,m)* élre *c(n,m)=1*), *szélességi gráfkeresés* (*f = g, c(n,m)=1*), *egyenletes gráfkeresés* (*f = g*)
- Heurisztikus gráfkeresések *f*-je a *h* a heurisztikus függvényre épül, amely minden csúcsban a hátralevő optimális *h** költséget becsli. Ilyen az *előre tekintő gráfkeresés* (*f = h*), az *A algoritmus* (*f = g+h, h≥0*), az *A* algoritmus* (*f = g+h, h*≥ h≥0 – h megengedhető*), az A^C algoritmus (*f = g+h, h*≥h≥0,* minden *(n,m)* élre *h(n)-h(m)≤c(n,m)*), és *B algoritmus* (ahol az *f= g+h, h≥0* helyett a *g*-t használjuk a kiterjesztendő csúcs kiválasztására azon nyílt csúcsok közül, amelyek *f* értéke kisebb, mint az eddig kiterjesztett csúcsok *f* értékeinek maximuma).

Véges δ-gráfokon minden gráfkeresés terminál, és ha van megoldás, talál egyet. A nevezetes gráfkeresések többsége végtelen nagy gráfokon is találnak megoldást, ha van megoldás. (Kivétel az előre-tekintő keresés és a mélységi korlátot nem használó mélységi gráfkeresés.) Az *A**, A^C algoritmusok optimális megoldást találnak, ha van megoldás. Az A^C algoritmus egy csúcsot legfeljebb egyszer terjeszt csak ki.

Egy gráfkeresés memória igényét a kiterjesztett csúcsok számával, futási idejét ezek kiterjesztéseinek számával mérjük. (Egy csúcs általában többször is kiterjesztődhet, de δ-gráfokban csak véges sokszor.) *A**algoritmusnál a futási idő legrosszabb esetben exponenciálisan függ a kiterjesztett csúcsok számától, de ha olyan heurisztikát választunk, amelyre már A^C algoritmust kapunk, akkor a futási idő lineáris lesz. Persze ezzel a másik heurisztikával változik a kiterjesztett csúcsok száma is, így nem biztos, hogy egy A^C algoritmus ugyanazon a gráfon összességében kevesebb kiterjesztést végez, mint egy csúcsot többször is kiterjesztő *A**algoritmus. A *B* algoritmus futási ideje négyzetes, és ha olyan heurisztikus függvényt használ, mint az *A**algoritmus (azaz megengedhetőt), akkor ugyanúgy optimális megoldást talál (ha van megoldás) és a kiterjesztett csúcsok száma (mellesleg a halmaza is) megegyezik az *A**algoritmus által kiterjesztett csúcsokéval.

Kétszemélyes (teljes információjú, véges és determinisztikus, zéró összegű) ***játékok***

A játékokat állapottér-reprezentációval szokás leírni, és az állapot-gráfot faként ábrázolják.

A *győztes (vagy nem-vesztes) stratégia* egy olyan elv, amelyet betartva egy játékos az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje (ne veszítse el) a játékot. Valamelyik játékosnak biztosan van győztes (nem-vesztes) stratégiája. Győztes (nem-vesztes) stratégia keresése a *játékfában* kombinatorikus robbanást okozhat, ezért e helyett részfa kiértékelést szoktak alkalmazni a soron következő jó lépés meghatározásához.

A *minimax* algoritmus az aktuális állásból felépíti a játékfa egy részét, kiértékeli annak leveleit aszerint, hogy azok által képviselt állások milyen mértékben kedveznek nekünk vagy az ellenfélnek, majd szintenként váltakozva az ellenfél szintjein a gyerekcsúcsok értékeinek minimumát, a saját szintjeinken azok maximumát futtatjuk fel a szülőcsúcshoz. Ahonnan a gyökérhez kerül érték, az lesz soron következő lépésünk.

A minimax algoritmusnak számos módosítása ismert, amelyek különféle javításokat tartalmaznak. A legfigyelemreméltóbb az *alfa-béta* algoritmus, amely egyfelől kisebb memória igényű (egyszerre csak egy ágat tárol a vizsgált részfából), másfelől egy sajátos vágási stratégia miatt jóval kevesebb csúcsot vizsgál meg, mint a minimax.

Evolúciós algoritmus

Egy adott pillanatban a problématérnek nem csak egy elemét (*egyedét*) tárolja, hanem azoknak egy részhalmazát (*populációját*), és arra törekszik, hogy a populáció egyedei minél jobbak legyenek (a helyes válaszhoz közelítsenek). Ennek megítéléséhez egy *rátermettségi függvényt* használ. Az egyedeket úgy kódolja, hogy azok tulajdonságai darabolhatóak legyenek: egy kódszakasz megváltoztatásával más tulajdonságú egyedet kapjunk.

A populációt lépésről lépésre változtatja úgy, hogy egyedeinek egy részét azokra hasonló, de lehetőleg jobb egyedekre cseréli. (A populáció megváltozása visszavonhatatlan. Ez tehát egy nem-módosítható stratégiájú keresés.) Ehhez négy műveletet hajt végre:

- *Szelekció*: Kijelölünk rátermett egyedeket (de a kevésbé rátermettek is kapnak esélyt)
- *Rekombináció*: A kiválasztott egyedek párjaiból utódokat állítunk elő.
- *Mutáció*: Az utódokat kismértékben módosítjuk.
- *Visszahelyezés*: Új populáció kialakítása az utódokból és a régi populációból.

Gépi tanulás

Egy algoritmus akkor tanul, ha egy feladatcsoport minta (tanító) feladataira végrehajtott futtatásai során mind a mintafeladatokra, mind a feladatcsoport más feladataira adott megoldása működésről működésre egyre jobb lesz. A tanulás célja az, hogy egy φ:X→Y leképezést tanuljunk meg azáltal, hogy a helyettesítéséhez egy olyan f:P×X→Y leképezést (P a paraméterek halmazát jelöli) konstruálunk, amelyik minél kisebb hibával közelíti a φ-t. (Ezektől eltérő, egyedi tanuló algoritmusokkal is találkozhatunk, mint például Mérő *B'* gráfkereső algoritmusa, amelyik a kezdetben megadott heurisztikát módosítja.) A tanuláshoz az X halmazból vett tanító mintákat használunk.

Felügyelt tanulás

Felügyelt tanulás esetén a tanító minták φ szerinti eredményeit is ismerjük: azaz a tanító halmaz ilyenkor a $T = \{(x_n, y_n)$, n=1,…,N és $y_n=\varphi(x_n)\}$. A φ és az f leképezéseknek a tanító mintákra vett hibáját az $L(\theta,T) = \sum_{n=1}^{N} l(f(\theta, x_n), y_n)$ hibafüggvény mutatja, ahol l:Y×Y→$\mathbb{R}$ egy alkalmas távolság függvény. A tanulási folyamat célja egy olyan θ-nak a megtalálása, amelyre az $L(\theta,T)$ kellően kicsi. Attól függően, hogy a hibafüggvény hiper-paramétereit ($f(\theta, x)$, $l(x, y)$, stb.) hogyan választjuk meg, különféle tanulási módszerekhez jutunk.

- A *K legközelebbi szomszédok* (KNN) módszernél az $f(\theta, x)$ értéke úgy áll elő, hogy vesszük tanító minták közül azt a K darabot, amelyek x_n-jei a legközelebb esnek x-hez, és ezen K darab minta y_n-jeinek számtani közepét képezzük. Az $f(\theta, x)$ kiszámítása ennél a módszernél lassú. A θ-t itt tulajdonképpen a minták jelentik. A módszer K-ra érzékeny.
- A *véletlen erdők* (RF) módszer először K darab döntési fát épít fel a tanító minták alapján úgy, hogy egy fa építéséhez a tanító mintáknak is, és a minták attribútumainak is egy-egy véletlen részhalmazát használja csak fel.[1] Egy véletlen erdő birtokában egy x bemenetre megállapíthatjuk, hogy az x az egyes döntési fák melyik levelére képződik le. Vesszük az ezen levelekhez tartozó tanító példák részhalmazait (T_k), és az $f(\theta, x)$ értékét az y_n-ek súlyozott összegeként számoljuk úgy, hogy a súly azon $1/|T_k|$ értékek számtani közepe, amelyre a T_k tartalmazza az y_n-hez tartozó x_n-t. Az $f(\theta, x)$ kiszámítása (a véletlen erdő ismeretében) gyors. A θ-t itt a véletlen erdő adja.
- A *mély neurális hálózatok* (DNN) esetében az $f(\theta, x)$ egy $g_K(\theta_K, g_{K-1}(\dots g_2(\theta_2, g_1(\theta_1, x_n))\dots))$ alakú összetett függvény, amellyel rétegről rétegre történik a leképezés: $x_n^{(k)}=g_k(\theta_k, x_n^{(k-1)})$. Egy tipikus (sűrűn rétegelt) változata ennek az, amikor a θ_k paraméterek ún. W_k súlymátrixok, és egy réteg az $x_n^{(k)}=h_k(W_k,x_n^{(k-1)}+b_k)$ képlettel számolható, ahol h_k egy folytonos differenciálható függvény, b_k az m dimenziós ún. bias vektor. Az $f(\theta, x)$ kiszámítása ennél a módszernél gyors, a súlyok betanulása gradiens módszer alapján történik.

[1] Egy döntési fa egy X→Y leképezést ír le, ahol egy X típusú bemenet attribútumok sorozatával adható meg. A fa csúcsai egy-egy attribútummal vannak címkézve, és annyi gyereke van, ahány értéket az adott attribútum felvehet (illetve ahány tartományát megkülönböztetjük az értékeknek). A fa levelei egy-egy kimeneti értéket képviselnek. Egy bemeneti elem a fa egy levelére képződik le azáltal, hogy a fa gyökerétől elindulva úgy haladunk lefelé, hogy minden csúcsnál megvizsgáljuk, hogy a bemenetnek mi az adott csúcs attribútumára adott értéke, és ezen értéknek megfelelő gyerekcsúcsra lépünk. Amikor levélcsúcshoz érünk, az megadja a keresett kimeneti értéket.
A döntési fát megadott tanító mintából és azok ismert attribútumaiból építjük fel úgy, hogy a fa csúcsai a tanító példákat hierarchikusan osztályozzák. Minden csúcshoz a tanító példák egy részhalmaza tartozik, amelyeket a csúcs attribútuma alapján osztályozunk, és ezen osztályokat a csúcs gyerekcsúcsaihoz rendeljük. Egy levélcsúcs kimeneti értéke az, amellyel a levélcsúcshoz került tanító példák többsége rendelkezik.

Nem felügyelt tanulás

Nem felügyelt tanulás esetében a tanító mintákhoz nem tartoznak kimeneti értékek, ezek nélkül próbálunk valamilyen általánosítható ismerethez jutni a $T = \{x_n,, n=1,\ldots,N\}$ tanító mintákat nyújtó X halmaz elemeiről.

A klaszterezés során részekre (klaszterekre) bontjuk az X halmazt úgy, hogy egy klaszteren belül minél hasonlóbb elemeket találjunk, két különböző klaszterből való elemek pedig minél kevésbé legyenek hasonlóak.

- A *k-means algoritmus* egy hard klaszterező módszer, azaz minden elem pontosan egy klaszterhez tartozhat. Előre meg kell adni a klaszterek számát. A klaszterek középpontja a klaszterhez tartozó minták átlaga. A cél, hogy a klaszterbe tartozó mintáknak a klaszter középpontjától vett távolságnégyzeteinek összege minimális legyen. Ezt a célt iterációval érjük el: minden lépésben meghatározzuk a klaszterek középpontjaihoz tartozó (azokhoz legközelebb eső) minták halmazait, majd ezen halmazok középpontjaiba igazítjuk a klaszter középpontokat. Mivel így csak lokális optimumokat tudunk előállítani érdemes véletlen kezdeti beállításokkal többször futtatni.
- A *téma modellezés* egy soft klaszterező eljárás, azaz a mintákról azt mondja meg, hogy milyen mértékben tartoznak egy klaszterhez. Itt a minták dokumentumok, a klaszterek pedig a témák. Az LSA (Latent Semantic Analysis) egy szó-dokumentum mátrix szinguláris érték felbontásaként áll elő. A bemeneti mátrixban minden sor egy szóhoz, minden oszlop egy dokumentumhoz tartozik, és a mátrix értékei, hogy az adott szó hányszor fordult elő az adott dokumentumban. A szinguláris érték felbontás eredményeként megkapjuk a szó-téma és a téma-dokumentum mátrixokat, amik hasonlóan épülnek fel.

A *dimenzió csökkentés* során adatpontjainkat egy alacsonyabb dimenziós térbe képezzük le. Ennek célja többek között lehet az adatok láttatása, vagy a lényeges információ kiemelése (a zaj csökkentése). Sokszor előfordul, hogy kisebb dimenziós adatok egy nagy dimenziós térben vannak.

- A *főkomponens analízis* az adatpontokat egy olyan koordinátarendszerbe transzformálja, ahol az első tengely mentén megtartjuk a lehető legtöbb szórást az adatokból, a második tengelyen a második legtöbb szórást, és így tovább. A tengelyeket főkomponenseknek hívjuk. Mivel a főkomponensek a megtartott szórás szerint rendezettek, az első k főkomponens megtartásával egy olyan dimenziócsökkentéshez jutunk, ami a lehető legtöbb szórást tartja meg az adatokból. Ha az adatpontokat egy mátrix soraiba tesszük, és nulla átlagúra hozzuk, akkor a főkomponensek e mátrix kovariancia mátrixának sajátvektorai, amik szinguláris érték felbontással az input mátrixból könnyen kiszámíthatók.

Az *autoenkóderek* olyan neuronhálók, melyek célja egy, az input dimenziójánál rendszerint kisebb dimenziós reprezentáció (*kód*) tanulása az input rekonstruálásával mint célfüggvénnyel. A legegyszerűbb, egy rejtett rétegű autoenkóderek a nemlinearitások ellenére egy főkomponens analízist valósítanak meg. Az autoenkóderek előnye, hogy tetszőlegesen bővíthetőek, építhetőek. Néhány autoenkóder változat a zajtalanító autoenkóder, ami egy zajos inputból próbálja meg visszaállítani az eredeti zajtalan inputot, illetve a ritka autoenkóder, ahol a kódban egyszerre csak néhány neuron lehet aktív. A variációs autoenkóderek már egy valószínűségi modellt feltételeznek, és adott outputok helyett outputok eloszlását tanulják, így az autoenkódereket általánosítják.