ELTE IK - Programtervező Informatikus BSc

Záróvizsga tételek

6. Mesterséges intelligencia

Mesterséges intelligencia

MI problémák és az útkeresési feladat kapcsolata. Modellezési technikák (pl. állapottér modell, dekompozíciós modell). Heurisztikus útkereső algoritmusok: lokális keresések (hegymászó módszer, tabu-keresés, szimulált hűtés), visszalépéses keresés, heurisztikus gráfkereső eljárások (A, A*, A^C, B algoritmusok). Kétszemélyes játékok.

1 Bevezetés

Az MI az intelligens gondolkodás számítógépes reprodukálása szempontjából hasznos elveket, módszereket, technikákat kutatja, fejleszti, rendszerezi. Megoldandó feladatai: nehezek, mert ezek problématere hatalmas, a megoldás megkeresése kellő intuíció hiányában kombinatorikus robbanáshoz vezethet. A szoftver intelligensen viselkedik, és sajátos eszközöket használ. A reprezentáció átgondolt a feladat modellezéséhez, és az algoritmusok hatékonyak, heurisztikával megerősítve.

2 Útkeresési problémák

Útkeresési problémaként sok MI feladat fogalmazható meg 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.

3 Modellezési technikák

3.1 Állapottér-reprezentáció

Állapottér-reprezentáció egy olyan lehetséges (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. Az állapottér nem azonos a problématérrel, hiszen a problématér elemei a startcsúcsból kivezető utak (műveletsorozatok), nem pedig az állapotok (csúcsok). A megoldás a problématér egy eleme, ami egy olyan műveletsorozat, ami startcsúcsból célcsúcsba vezet.

3.2 Dekompozíciós modell

A probléma dekompozíciólényege az, hogy egy feladatot részfeladatokra bontunk, majd azokat tovább részletezzük, amíg nyilvánvalóan megoldható feladatokat nem kapunk. A reprezentációhoz meg kell adnunk:

- a feladat részproblémáinakáltalános leírását,
- az eredeti problémát,
- az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem, és
- a dekomponálóműveleteket:

Dekomponálóműveleteket nagyon nehéz megtalálni:

- Nem biztos, hogy megtaláljuk.
- Hamis dekomponálóműveletek.
- Nem minden feladat dekomponálható.

Az egyszerű probléma felismerése sem egyértelmű. A megoldás kiolvasása sem nyilvánvaló. Egy dekompozíciós eprezentációhoz tartozó (R, s, T) gráfreprezentációban

- az $R = (N, A, c)$ egy olyan ÉS/VAGY gráf (dekompozíciósgráfban), ahol
- N a részprobémákat,
- A a dekomponálóműveleteket,
- c azok költségeit szimbolizálják,
- s az eredeti problémát,
- T az egyszerű problémákat jelöli.

A probléma megoldását egy $s \rightarrow M \subseteq T$ közönséges irányított kört nem tartalmazó hiperút, az úgynevezett megoldásgráf megtalálása jelenti. Az eredeti probléma megoldása ebből a megoldásgráfból nyerhető ki. A megoldás költsége többnyire nem függ a megoldásgráf költségétől, ezért nem cél az optimális megoldásgráf előállítása.

4 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 (ez lehet nemmódosítható vagy módosítható), 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.

5 Lokális keresések

A lokális keresések egyetlen aktuális csúcsot és annak szűk környezetét tárolják a globális munkaterületen. Keresési szabályai az aktuális csúcsot minden lépésben a szomszédjai közül vett lehetőleg „jobb" gyerekcsúccsal cserélik le. A vezérlési stratégiájuk a „jobbság" eldöntéséhez egy *rátermettségi függvényt* használ, amely 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.

6 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), akkor megoldást talál. De 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.

7 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, vagyis előfordulhat, hogy elromlik a keresőgráf korrektsége.

- 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 heurisztikus függvényre épül, amely minden csúcsban a hátralevő optimális h* költséget becsli. Ilyen az *előre tekintő gráfkeresés* ($f = h$), az *A algoritmus* ($f = g + h, h \geq 0$), az *A* algoritmus* ($f = g+h, h^* \geq h \geq 0$ – h megengedhető), az *A^C algoritmus* ($f = g+h, h^* \geq h \geq 0$, minden (n, m) élre $h(n)-h(m) \leq c(n, m)$), és *B algoritmus* (ahol az $f = g+h$, $h \geq 0$ 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.

8 Kétszemélyes (teljes információjú, zéró összegű, véges) 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 legismertebb módosítása 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. Saját szinten α, ellenfelén β értéket adunk meg, kezdetben $-\infty$, illetve $+\infty$ értékkel. Visszalépéskor változtatunk rajta, α-t növeljük, β-t csökkentjük. Vágás akkor történik, ha az úton vannak olyan értékek, hogy $\alpha \geq \beta$. További módosítások még az átlagoló (legnagyobb m és legkisebb n darab érték átlagát vesszük), illetve a váltakozó mélységű kiértékelésű minimax (minden ágon reális értéket mutasson a kiértékelő függvény nyugalmi teszttel), továbbá a negamax algoritmus (ellenfél szintjén (-1)-szeres érték, maximumot választunk minden szinten az ellentettekből).