Semestrální práce BI-ZUM (Karbaník s kamennou tváří)

Úvod

Jako moji semestrální práci jsem si zvolil zadání Karbaník s kamennou tváří. Konkrétně jsem se zabýval návrhem a vytvořením vytvořením umělé inteligence pro hraní pokeru ve verzi (texas hold'em). Aplikace je zpracována v jazyce C++ a jako rozhranní s uživatelem je využita konzole.

Cílem práce je zjistit pravděpodobnost výhry v daném kole podle držených karet, dále najít strukturu pro reprezentaci herní strategie (jak moc je AI agresivní) a nakonec tyto informace využít k určení výše sázky. A optimalizace herní strategie.

Aproximace pravděpodobnosti výhry

Pro aproximaci pravděpodobnosti výhry hráče jsem využil **Monte Carlo** simulaci. Výhodou této metody je především jednoduchost implementace a obecnost – snadno lze pravděpodobnost spočítat ve všech fázích hry.

U této techniky všem protivníkům vyberu náhodné karty z balíčku a pokud je na stole méně než 5 karet, tak se karty doplní i tam. Poté se již jen vyhodnotí, jestli hráč vyhrál, nebo prohrál.
Toto je jedna simulace. Pravděpodobnost výhry je pak určena jako #výher / #simulací.
Pozorováním jsem zjistil, že po 500 simulacích se již aproximace pravděpodobnosti blíží skutečné hodnotě (+- 2 %). Ve své aplikaci používám 500 * počet hráčů simulací.

Herní strategie

Pro rozhodování toho, jakou má AI zvolit sázku hledám funkci, která bude mít na vstupu: herní fázi (pre-flop, plop, turn, river), suma žetonů, které má hráč k dispozici, suma žetonů, které má hráč vsazené, počet žetonů na stole, výherní pravděpodobnost. Výstupem funkce je počet žetonů ke vsazení.

Při vymýšlení toho, jak strojově reprezentovat herní strategii jsem hledal takovou strukturu, kt
Pro tento problém jsem zvolil optimalizační strategii nesystematické prohledávání – konkrétně genetický algoritmus. Pro tento účel jsem zvolil strukturu několika vektorů (1 vektor pro každou herní fázi).
Každý vektor obsahuje 3 racionální čísla:

1) váha pravděpodobnosti výhry
2) váha sumy zbývajících žetonů
3) threshold (procentuální hodnota o kterou se musí lišit aktuální vs predikovaná sázka – jinak se aplikuje aktuální sázka – zabraňuje vysoké ocilaci sázek o marginální hodnoty)

Predikovaná sázka se poté vypočítá následovně:
p = žetony na stole * pravděpodobnost výhry * váha pravděpodobnosti výhry * relativní počet žetonů hráče (relativní ke startovnímu počtu hráče)
Tato hodnota se porovná, jestli překračuje threshold a pokud ano, hodnota se vrátí, jinak se vrátí aktuální sázka.

Myšlenka byla, že taková, že strategii reprezentovanou souborem vektorů bude možné optimalizovat pomocí GA. Jako fitness funkci používám simulaci hry mezi několika natrénovanými agenty. Počáteční suma žetonů je 500, každé kolo, kdy ještě agent nebyl vyřazen inkrementuji fitness skóre o 1, pokud se dostane až do 30. kola, přičtu ještě finální počet žetonů.

Celou populaci takto ohodnotím a vyberu 2 jedince s nejvyší fitness hodnotou (jako operátor selekce využívám turnajovou selekci, ruletová selekce vracela často velice špatné jedince).

Jedince zkřížím, tolikrát, abych doplnil celou populaci a poté aplikuji mutaci.
Používám vícebodové křížení – jeden bod pro každý vektor, pravděpodobnost mutace je 1%.

Použití aplikace

1) kompilace:
g++ -Wall -pedantic main.cpp
2) spuštění hry:
Je třeba předat parametr „play“ a dále dvojici parametrů za každého hráče. První parametr dvojice je název hráče a druhý je cesta k natrénovanému agentovi (případně klíčové slovo human, pokud je hráč interaktivní).

Upozornění: pro správné zobrazení karet v terminálu je třeba zvolit správný font (monospace). U některých fontů je šířka symbolu na kartě atypická a zobrazení karet je poté posunuté.
Správné zobrazení je garantované např. U fontu *DejaVu Sans Mono*.
3) natrénování agenta
Je třeba předat parametr „train“ a dále n cest k agentům. První agent je trénovaný a zbytek agentů slouží pouze jako oponenti pro vyhodnocení fitness funkce. Předposlední parametr určuje velikost populace a poslední parametr je počet generací, po kterých se má natrénovaný model uložit do souboru. Pokud ještě nemáte žádný model ke trénování, je třeba ho náhodně inicializovat – vizte bod 4.
4) inicializace agenta
Je třeba předat parametr „init“ a dále cestu, kam se má náhodně inicializovaný agent uložit.

Závěr

Na závěr bych rád zhodnotil plody mého snažení, kde bohužel musím konstatovat, že se mi bohužel tento postup v praxi nepodařilo přetavit v optimální výsledky.
Při učení se fitness populace příliš nezvyšuje, ale spíše tato hodnota osciluje kolem fitness hodnoty náhodného agenta, nebo se dokonce i s postupujícími generacemi zhoršuje.

Snažil jsem se upravit některé parametry (operátor selekce, pravděpodobnost mutace, velikost populace, počet kol při hodnocení ve fitness funkci…), ale bohužel výsledek byl stále podobný a agent nekonverguje k optimálnímu řešení. Zkusil jsem také operátor křížení upravit tak, že od náhodně zvoleného rodiče použiji celou strategii (za cenu vyšší pravděpodobnosti mutace) při generování potomka, čímž vlastně degraduji algoritmus na hill climbing – ovšem ani v tomto případě se optimálnost agenta nezlepší. Nepodařilo se mi tedy bohužel ani za dlouhé hodiny identifikovat příčinu problému.

Zdroje

Přednáškové slidy BI-ZUM