FIIT STU

Dokumentácia Zadanie 3

Umelá Inteligencia

Samuel Bubán
ID: 102879

Samuel Bubán 16.11.2020

Zadanie: Zenová záhrada, pomocou algoritmu Tabu search a simulated annealing

Problém: mních chce pohrabať záhradu tak, že vzniknú vodorovné/zvislé pásy. Mimo záhradky sa môže pohybovať ako chce. Pri narazení na prekážku, ak má na výber, či sa otočí doprava alebo doľava, vyberie si sám.

Bonus: V záhrade sa nachádza popadané lístie, ktoré musí zbierať v poradí od najsvetlejších po najtmavšie.

Riešenie: zadanie som riešil cez štyri algoritmy. Hill climber algoritmus, steep hill climber algoritmus, tabu search algoritmus a simulated annealing algoritmus.

Jeden stav – mních, ktorému je priradené jeho fitness. Každý mních má pole rozhodnutí (ak narazí na prekážku, či sa otočí vpravo alebo vľavo), a pole vstupných políčok (keď je mimo záhradky, tak cez ktoré políčko vstúpi).

Fitness – fitness je počítané mechanicky pre každého mnícha samostatne od základu. Vytvorí sa kópia záhrady, do ktorej sa tento mních pošle. Prechádza ju políčko za políčkom. Ak sa má rozhodnúť, vyberie si rozhodnutie zo svojho poľa rozhodnutí, a posunie jeho index ďalej. Takto sa prejde celá záhrada, a počet políčok, ktoré mních úspešne pohrabal sa rovná jeho fitness. Aj keď mních začne hrabať, a nepodarí sa mu vyjsť von, aj toto sa počíta do jeho fitness.

Vstupné políčka – pre zmenšenie množstva dát, ktoré si každý mních pamätá sú vstupné políčka vygenerované samostatne mimo, ako statické pole. Každý mních si teda ukladá len index do tohoto poľa vstupných políčok. Každé vstupné políčko tiež obsahuje informáciu o otočení, aby ju nebolo neskôr potrebné počítať. Aby boli implementované všetky možnosti, rohové políčka sa nachádzajú v tomto zozname 2x – majú rozdielne otočenie.

Argumenty funkcií – všetky algoritmické funkcie majú argument začiatočného mnícha, ktorého treba vygenerovať pred spustením algoritmov. Ako ďalší argument majú záhradu, ktorá je načítaná zo súboru, a tretí argument sú vstupné políčka.

Hill climber algoritmus (lite): Pre začiatočného mnícha sa začnú generovať susední mnísi (zmení sa jedno rozhodnutie na opačné, alebo sa vymenia dve vstupné políčka). Týmto sa môže stať, že sa niektoré stavy vyskytnú aj viackrát medzi susedmi, ale to nie je problém. Títo mnísi následne pohrabú svoju záhradu (vypočíta sa ich fitness). Keď narazí algoritmus na lepšieho mnícha, ako je sám, tak sa tento lepší mních zvolí ako nový aktuálny, a algoritmus pokračuje od začiatku (vygeneruje susedov, porovná ...).

Ak už žiaden zo susedných mníchov nemá väčší fitness ako aktuálny, tak algoritmus príde na koniec, a vráti aktuálneho mnícha ako výstupnú hodnotu.

Hill climber: Pre začiatočného mnícha sa vygenerujú všetci susední mnísi. Každý mních pohrabe svoju záhradu, a je mu tak pridelené fitness. Pole susedných mníchov sa následne zoradí podľa hodnoty fitness od najväčšej po najmenšiu. Z tohoto poľa je vybraný prvý mních. Ak má väčší fitness ako aktuálny mních, stáva sa aktuálnym mníchom a algoritmus pokračuje od začiatku.

Ak má ale menší fitness ako aktuálny, algoritmus skončí a ako výstupná hodnota je aktuálny mních.

Tabu search: Funguje rovnako ako Hill climber, ale s tým rozdielom, že neskončí, keď sa nenašiel lepší susedný mních. Tabu search funkcia má ďalšie dva vstupné argumenty – násobný kvocient pre veľkosť tabu zoznamu, a násobný kvocient pre počet navštívení tabu zoznamu. Tieto argumenty nie sú povinné, a sú nastavené na 1 a 1

Ak sa nenašiel lepší susedný mních, tak je aktuálny mních pridaný do tabu zoznamu. Preto ak by sa aj našiel lepší sused, tak treba najprv skontrolovať, či sa nenachádza v tabu zozname. Tento zoznam funguje ako rad – ak už v ňom nie je miesto, tak sa prvý, ktorý bol do neho pridaný vyhodí.

Algoritmus skončí až vtedy, keď sa bude za sebou pridávať niekoľkokrát do zoznamu nový mních bez toho, že by sa našiel lepší, alebo ak aktuálny mních dosiahne teoretické maximálne fitness (počet políčok záhrady mínus počet kameňov).

Tento algoritmus má ešte tú výhodu, že si pamätá celkovo najlepšieho mnícha, takže ak by aj musel hľadať horších, stále vie vrátiť toho najlepšieho.

Simulated annealing: Tento algoritmus je úplne iný ako predchádzajúce. Má tiež dva argumenty navyše – teplotu, a násobný kvocient pre ochladzovanie. Z aktuálneho mnícha si zvolí náhodného suseda, následne porovná ich fitness. Ak je fitness nového mnícha väčší, tak sa stane aktuálnym mníchom, a algoritmus ide od začiatku.

Ak nie je väčší fitness, tak je určitá pravdepodobnosť, že sa aj tak zvolí, ako aktuálny mních, a pokračuje od začiatku.

Po každom jednom cykle algoritmu sa teplota zníži, a keď dosiahne hodnotu menšiu rovnú ako 0.1, tak skončí algoritmus, a vráti aktuálneho mnícha.

Teplotu som skúšal znižovať najprv lineárne, a potom násobne. Keď sa znižovala lineárne, tak algoritmus dosahoval mizerné výsledky, ale keď som ju znižoval násobne, tak boli výsledky o dosť lepšie, preto som sa rozhodol pre násobné znižovanie teploty.

Ideálne hodnoty pre voliteľné argumenty Tabu search a Simulated annealing:

V priečinku data je priložený Excel súbor data.xlsx, ktorý obsahuje testovacie dáta tohoto programu.

Pre **tabu search** som skúsil rôzne hodnoty, (všetky možnosti z [1, 2, 5], [1, 2, 5]). Tieto hodnoty vyjadrujú veľkosť tabu zoznamu a počet navštívení (násobný kvocient).

Porovnával som to na piatich rôznych mapách, a pre každú bolo vykonaných 10 rôznych spustení – na získanie priemerných hodnôt.

Tieto dva kvocienty mali len minimálny vplyv na výsledok, ale zato mali celkom veľký vplyv na časovú zložitosť. Preto som sa rozhodol, že ako defaultná hodnota budú oba kvocienty nastavené na 1. Takto sa zabezpečí najrýchlejší algoritmus len s malým dopadom na výsledné hodnoty.

Pre **simulated annealing** som skúsil hodnoty kombinácii [25, 50, 100], [0.25, 0.5, 1, 2, 5]. Prvá je teplota, druhý je kvocient pre ochladzovanie – vyšší kvocient znamená rýchlejšie ochladzovanie.

Takisto som to porovnával na piatich rôznych mapách a pre každú bolo vykonaných 10 rôznych spustení na získanie priemerných hodnôt.

Zmena teploty nemala na výsledné hodnoty takmer žiaden vplyv. Urýchlovanie ochladzovania ale negatívne ovplyvňovalo priemerný fitness, preto som sa tiež rozhodol pre defaultné hodnoty 25 a 1.

Testovanie za behu:

Pre 5 máp som spustil výpis počas behu programu. Tieto testovania ale mohli bežať len jedenkrát, ináč by sa nedalo dobre na grafe ukázať, ako sa menil fitness.

V priloženom Excel súbore data.xlsx sa nachádzajú postupné zmeny fitness. Je tam vidieť, že hillclimber algoritmus a tabu search začínajú rovnako, ale tabu search ešte môže nájsť lepšiu hodnotu po tom, ako hillclimber dosiahne maximum.

Hillclimber lite nemá nikdy lepší priebeh ako hillclimber, nakoľko vyberá prvého lepšieho suseda.

Simulated annealing má na začiatku väčšie skoky, ale tie sa postupne stabilizujú, a nájde výsledok podobný, ako ostatné algoritmy.

Veľké testovanie:

Nechal som celý program, aby prešiel cez 80 náhodne vygenerovaných máp, s náhodným počtom kameňov aj listov rôznych farieb (1-5), a pre každú mapu sa opakovali 10-krát, na získanie priemerných hodnôt.

Najprv boli vygenerovaní počiatoční mnísi pre každý beh (aby všetky algoritmy začínali s rovnakými podmienkami). Všetky algoritmy som nechal bežať naraz vďaka multithreadingu, čím sa znížil čas asi na polovicu. Program bežal asi deň vkuse, kým vyhodnotil všetky údaje.

Na prvých (menších) 40tich mapách mali všetky algoritmy porovnateľné výsledky - maximálne aj priemerné fitness hodnoty. Čas dopadol ako sa dalo očakávať – hillclimber lite mal horšiu časovú zložitosť, ako hillclimber, kvôli tomu, že musel prejsť cez viac mníchov, aby dosiahol podobný výsledok. Tabu search malo najhoršiu časovú zložitosť, keďže prechádzalo cez zoznam po tom, ako hillclimber skončil. Simulated annealing malo po celý čas viac menej stabilnú časovú zložitosť, keďže teplota sa stále znižovala pomalšie. Takisto to bol aj najrýchlejší algoritmus zo všetkých.

Pre zvyšných 40 máp boli ale výsledky iné. Nielenže mal hillclimber lite niekedy lepšie dosiahnuté maximum, ale dokonca aj priemerné hodnoty boli pri niektorých vstupoch vyššie. Simulated annealing malo ale horšie výsledky, nakoľko sa pre väčšie mapy nezvyšovalo množstvo operácii, ktoré vykoná.

Časová zložitosť dopadla rovnako, ako pri prvých 40tich mapách – tabu search mal najhoršie výsledky, a simulated annealing malo prakticky nulovú zložitosť.

GUI:

Pre lepšie zobrazenie riešenia som vytvoril grafické zobrazenie, kde mních pohrabe záhradu ako animácia. Tu je možné si zvoliť jeden z troch algoritmov (hillclimber, tabu search, simulated annealing), veľkosť populácie (koľko opakovaní sa spraví), vstupný súbor pre mapu a nastavenie nepovinných hodnôt pre tabu search a simulated annealing.

Po tom, ako prejde algoritmus cez celú populáciu mníchov si môže používateľ zvoliť, ktorého zo zobrazených by chcel vypísať.

Toto sa môže opakovať stále.

Vylepšenie:

Napadá mi jedno možné vylepšenie, a tým je, že by sa násobný kvocient pre väčšie mapy znižoval pre simulated annealing. Týmto by sa zabezpečilo prehľadanie väčšieho priestoru, pričom by mal stále neporovnateľný čas vzhľadom na ostatné algoritmy.

Programy:

algorithms.py – 4 algoritmy riešenia

analyzer.py – na získanie dát veľkého testovania

map_generator.py – vytváranie náhodných máp

monksolve.py – mních a jeho správanie (hrabanie záhrady ...)

optimizations.py – na získanie dát voliteľných parametrov

ZEN_GUI.py – grafický výpis záhrady