Fakulta informatiky a informačných technológií
STU v Bratislave

Umelá inteligencia

Hľadanie pokladu

Zadanie 3

Zsolt Kiss

Utorok 14:00-15:50
Ing. Ivan Kapustík

2020/2021

Obsah

Hlavná myšlienka

Zadanie je vypracované v programovacom jazyku Java. Za úlohu som dostal vyriešit problém b, hľadanie pokladu. Majme hľadača pokladov, ktorý sa pohybuje vo svete definovanom dvojrozmernou mriežkou a zbiera poklady, ktoré nájde po ceste.

Môže sa pohybovať štyrmi rôznymi smermi: hore H, dole D, doprava P a doľava L. K dispozícii má konečný počet krokov. Jeho úlohou je nazbierať čo najviac pokladov. Za nájdenie pokladu sa považuje len pozícia, pri ktorej je hľadač aj poklad na tom istom políčku. Susedné políčka sa neberú do úvahy.

O zdrojovom kóde

Opis dôležitých častí kódu

V triede Algorithm.java

V tejto triede sa uskutoční inicializácia. Táto operácia prebehne počas behu algoritmu vždy iba raz. Pozostáva z 2 častí. Uložím referenciu na objekty Mapy, Hľadača a Virtuálny stroj s ktorými budem pracovať počas krokov algoritmu. V durhej časti pomocou funkcie public void generatePopulation() vygenerujem prvú generáciu. Pre počet jedincov sa do každého poľa **population**[] náhodne vygeneruje nový jedinec.

public Subject converge (int maxGenerationCount)

> Táto funkcia vráti najúspešnejšieho jedinca. Začneme s nultou generáciou, a ohodnotíme postupne všetky generácie pomocou virtuálneho stroja, a zapíšeme fitness hodnoty. Pracujeme so 100 jedincami, a zistíme koľko najviac pokladov našla daná generácia. Vytvoríme pole jedincov pre novú populáciu, ktorou nahradíme pôvodnú populáciu pri krížení a mutáciach. Vytvoríme prioritný rad, do ktorého vložíme všetkých jedincov, a pomocou Comparatora ich porovnávame. Nasleduje elitarizmus. Jednoducho vyberáme najlepších 20% z celej populácie (podľa fitness hodnoty). Ak už medzi tými najlepšími existuje taký jedinec, ktorý našiel všetky poklady, tak už nemáme ísť ďalej a môžeme ukončit cyklus, a vrátiť index daného jedinca. V opačnom prípade funkcia vráti -1, ale tých najlepších jedincov naklonuje do novej populácie. Keďže máme celkom 100, jedincov, to znamená, že pre vytvorenie novej generácie ešte nám chýba 80 jedincov. To ďalej riešíme tak, že pre prvých 20 jedincov selektujeme pomocou metódy rank selection a zvyšných 60 pomocou metódy turnaja. Následne mutujeme x percent z celej novej populácie a nakoniec nahradíme pôvodnú populáciu novou.

public int elitism (int eliteSubjectsCount, int newSubjectsCount, Subject [] newPopulation, Subject [] bufferPopulation, PriorityQueue<Subject>frontSubjects, int treasuresFoundByGeneration)

> Táto funkcia vráti -1 ak medzi tými 20% najlepšími nie je taký jedinec, ktorý našiel všetky poklady. V opačnom prípade vráti index tohto jedinca. Vo funkcií odstránime 20 najlepších jedincov z prioritnej rady a vkladáme ich do pola novej generácie. Zvyšných 80 jedincov uložíme do pomocného poľa.

public void rankSelection(Subject[] newPopulation, Subject[] bufferPopulation, int eliteSubjectsCount, int rsSubjectsCount)

> Táto funkcia zmení poradie v pomocnom poli tak, aby boli vo zostupnom poradí podľa fitness hodnôt. Následne priradíme hodnosť pre 20 jedincov v tomto poli. Do spájaného zoznamu vložíme jedince toľko krát, koľko bola jej pravdepodobnosť výskytu. To som vypočítal tak, že hodnosť som roznásobil s 1000 a delil som Gaussovou formulou (čo je konštanta v tomto prípade). Príklad: jedinca (v prvej generácií) s hodnosťou 12 som vložil do rulety 57 krát, jedinca s hodnosťou 18 som vložil už 85 krát. Nakoniec som náhodne vybral 2 rodičov z rulety a krížením som vytvoril z nich nového jedinca. Cyklus sa vykonal 20 krát.

public void tournament(Subject[] newPopulation, int eliteSubjectsCount, Subject[] population, int tSubjectsCount, int rsSubjectsCount)

V tejto funkcií spracúvam zvyšných 60 jedincov potrebných pre vytvorenie novej generácie tak, že z populácie náhodne vyberem dvoch, tí budú súbojovať, a z víťaza bude prvý rodič. Potom vyberám ďalších dvoch (keďže vybranie je náhodné, môže sa stať, že vyberieme tých istých jedincov), zase sa vykoná súboj, a víťazcom bude druhý rodič. Na konci z dvoch víťazcov vytvárame jedného jedinca, a vložíme ho do novej populácie. Cyklus prebehne kým do poľa sme nevložili 60 jedincov.

public void mutate(Subject[] newPopulation)

Mutácia sa vyskytuje tak, že podľa pravdepodobnosti mutujeme x percent celej populácie. Pre jedinca zmeníme jednu bunku tak, že nahradíme ho náhodnou hodnotou.

V triede VirtualMachine.java

public void run (Subject subject) throws CloneNotSupportedException

V tejto funkcií vykonávame inštrukcie pomocou virtuálneho stroja. Najprv som deklaroval jednotlivé operácie, a to sú inkrementácia, dekrementácia a skok. Výpis som nedeklaroval, lebo ak hodnotu bunky násobíme s 192, dostaneme tú istú bunku. Klonujem jedinca, aby som zachoval pôvodny, kvôli tomu v tejto funckií zmením hodnoty pre jedinca v jeho bunkách. Prebiehajú inštrukcie, kým:

- zbehne 500 inštrukcii
- nájdeme všetky poklady
- jedinec išiel mimo mapy

Začneme s 0. inštrukciou, a kontrolujeme bunku jedinca. Zistíme operáciu podľa prvých dvoch bitov. Inkrementujeme/dekrementujeme hodnotu bunky alebo v prípade keď operácia je skok, tak v ďalšom kroku programu načítame danú bunku. Ak operácia je výpis, kontrolujeme posledné 2 bity a to tak, že spravíme logický súčin s 4. Výsledok je buď 00, 01, 10 alebo 11. Podľa výsledku posunieme hľadača pokladov na mapy a uložíme danú pozíciu pre jedinca. Ak hladač išiel mimo mapy alebo ak našiel všetky poklady končíme cyklus. Vypočítame fitness hodnotu a nastavíme preňho.

V triede TreasureFinder.java

public Position moveTo(int pohybX, int pohybY) throws CloneNotSupportedException

Táto funkcia má za úlohu posunúť jedinca na súradnici. Ak nájdeme poklad, tak zvýšime počet nájdených pokladov pre jedinca. Funkcia vráti aktuálnu pozíciu daného jedinca.

Vysvetlenie vstupného súboru

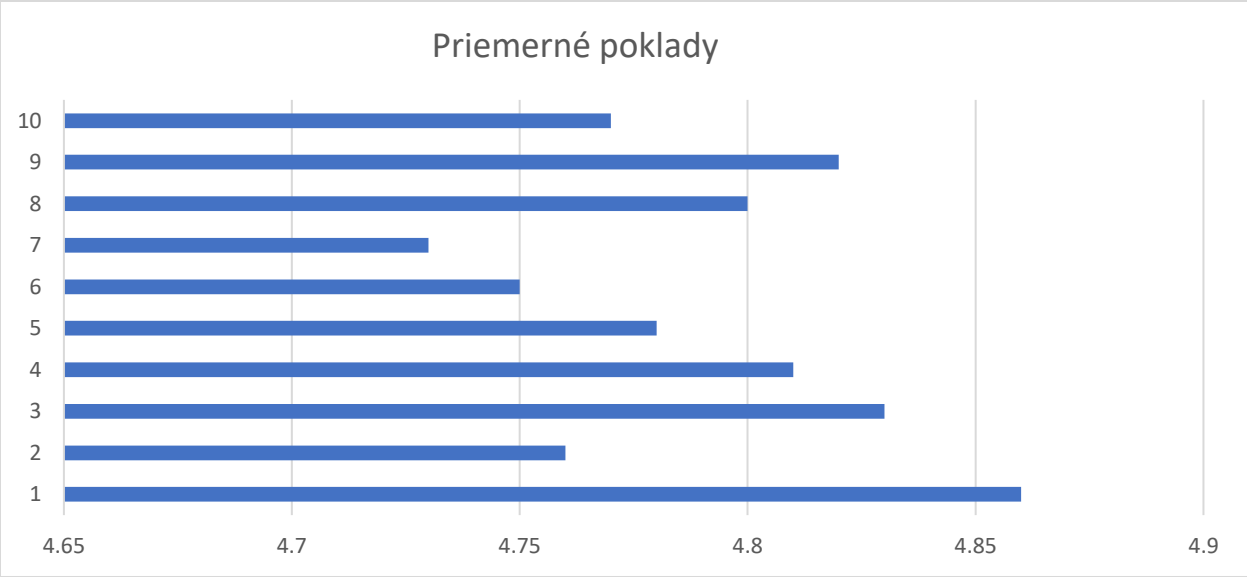
input.txt		
riadok	**hodnota (príklad)**	**čo je tam?**
1	5	Počet pokladov na mape
2	7	Počet stĺpcov
3	7	Počet riadkov
4	1,4	Súradnice 1. pokladu (x,y)
5	2,2	Súradnice 2. pokladu (x,y)
6	4,1	Súradnice 3. pokladu (x,y)
7	4,5	Súradnice 4. pokladu (x,y)
8	6,3	Súradnice 5. pokladu (x,y)
9	1	Začiatočný stĺpec jedinca (x)
10	3	Začiatočný riadok jedinca (y)
11	500	Maximálny počet generácií
12	0.15	Pravdepodobnosť mutácie

Z toho vyplýva, že ak zadáme počet pokladov n, tak od riadku 4 po 4+n-1 musia byť deklarované súradnice pokladov.

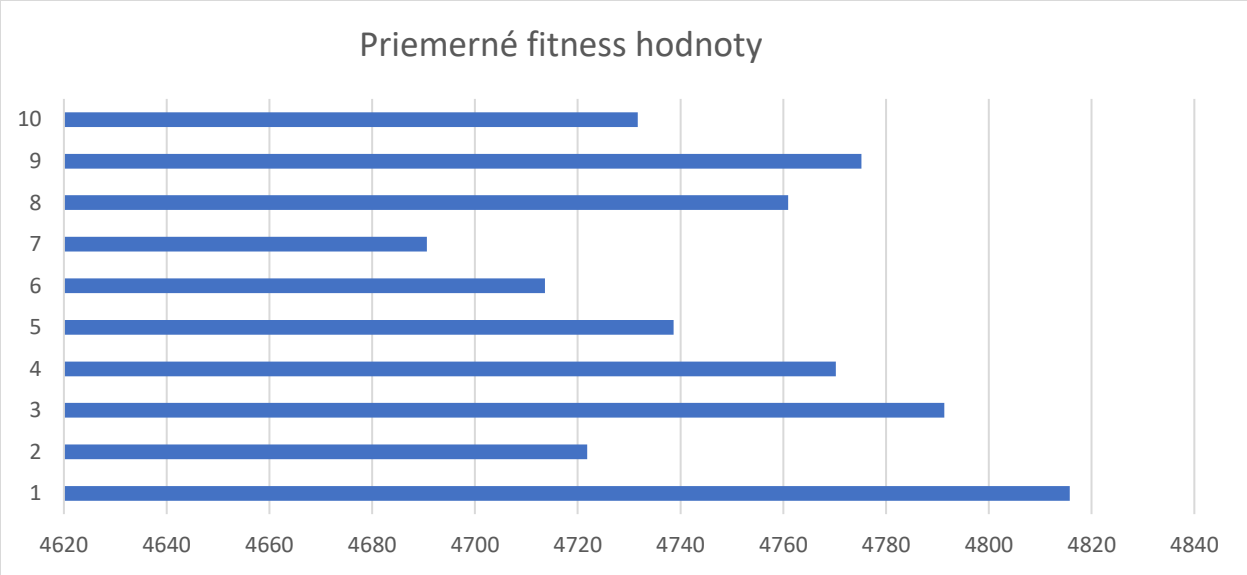
Testovanie

Pre testovanie som vytvoril funkciu **void** benchmark (v triede Solve.java), ktorý spustí algoritmus 100 krát. To znamená, že 100 krát vytvoríme maximálne 500 generácií (je ale možné zmeniť túto hodnotu) s 100 jedincami pre každú generáciu. Funkcia vypíše priemerný počet krokov, priemerné nájdené poklady a priemernú fitness hodnotu, a tým pádom jednoducho môžeme zistiť, aký efektívny bol náš algoritmus. Testoval som viaceré scenáre. Pre istotu som všetky prípady testoval viackrát, aby som zistil, že všetko funguje v poriadku, a medzi jednotlivými spusteniami nie sú veľké rozdiely. Dole uvedené výsledky sú priemerné hodnoty po spustení daného scenára 10 krát. Pre všetky testy som používal hore uvedené vstupné parametre. (viď obrázok hore)

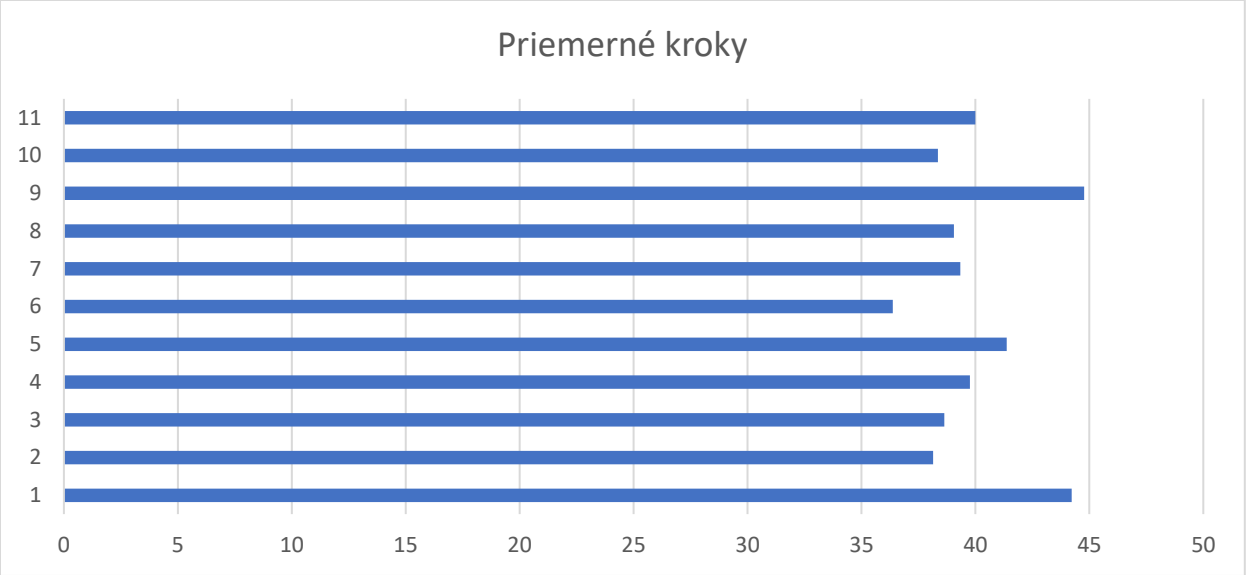
Test 1 – Používanie oboch metód selekcie: 20 elitov/60 jedincov vybraného turnaja/20 jedincov vybraných podľa hodností



Priemerné poklady: 4.791

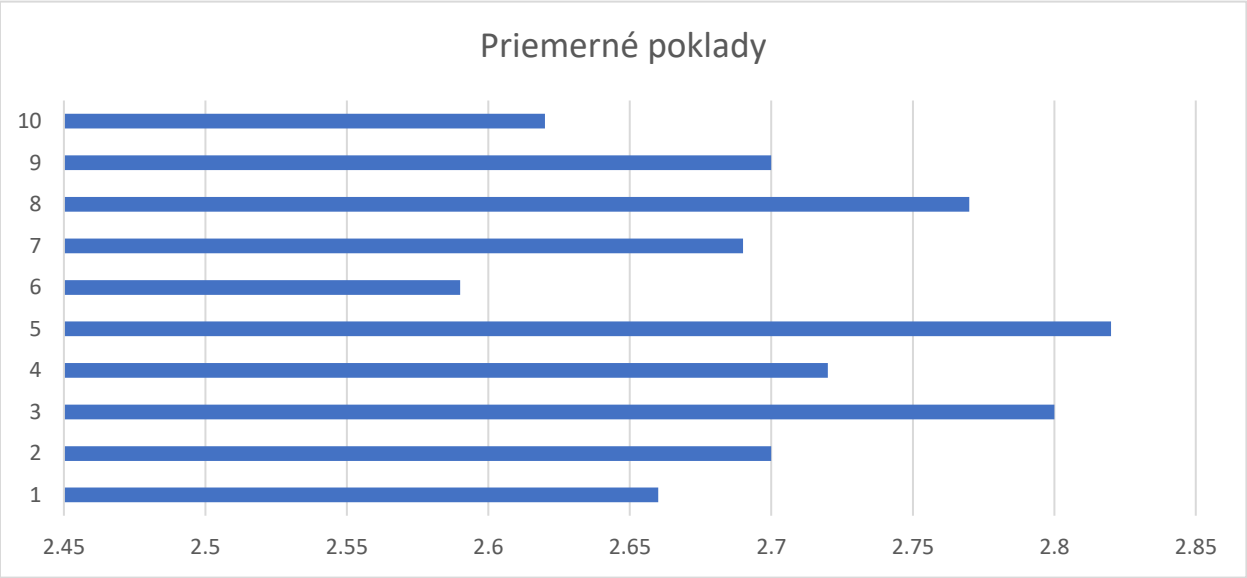


Priemerné fitness hodnoty: 4750.998

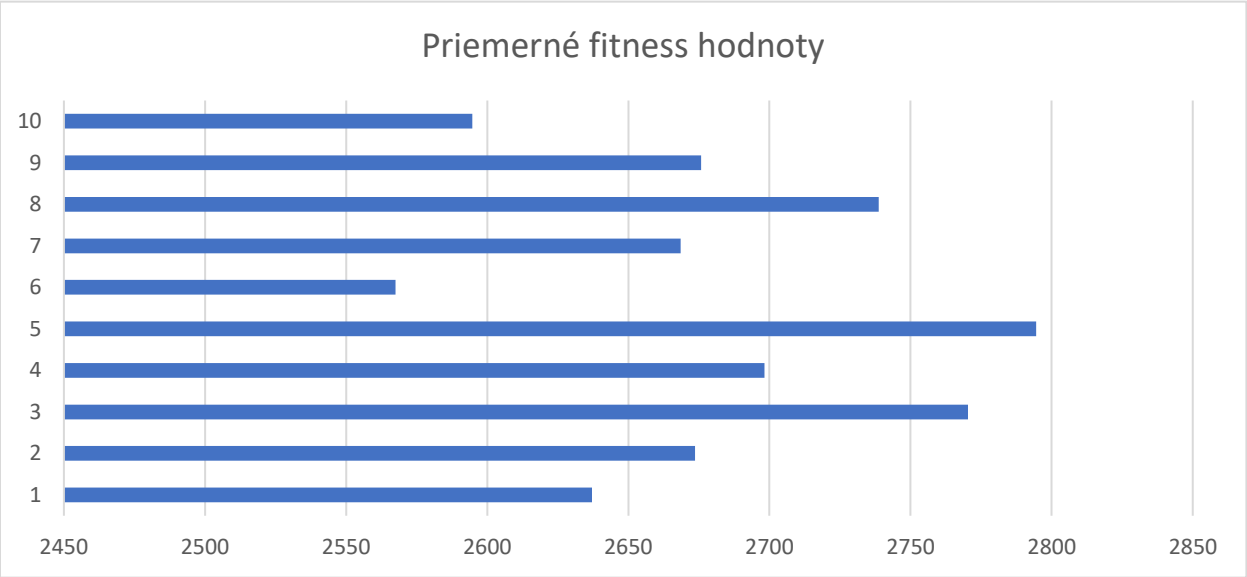


Priemerné kroky: 40.002

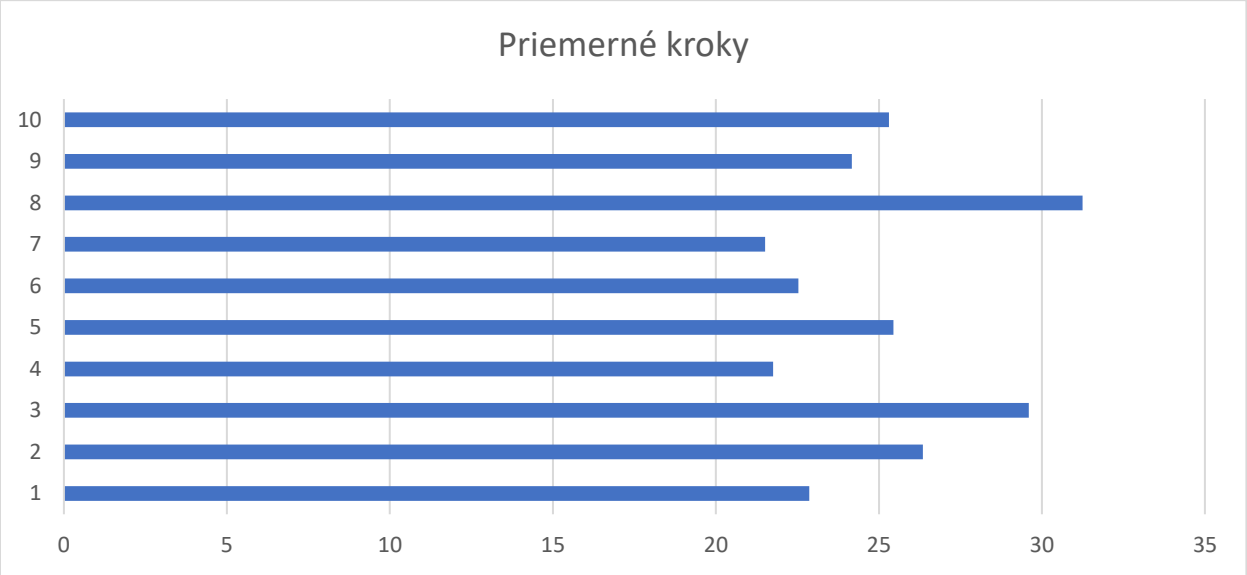
Test 2 – Používanie iba metódy rank selection: 20 elitov/80 jedincov vybraných podľa hodností



Priemerné poklady: 2.707



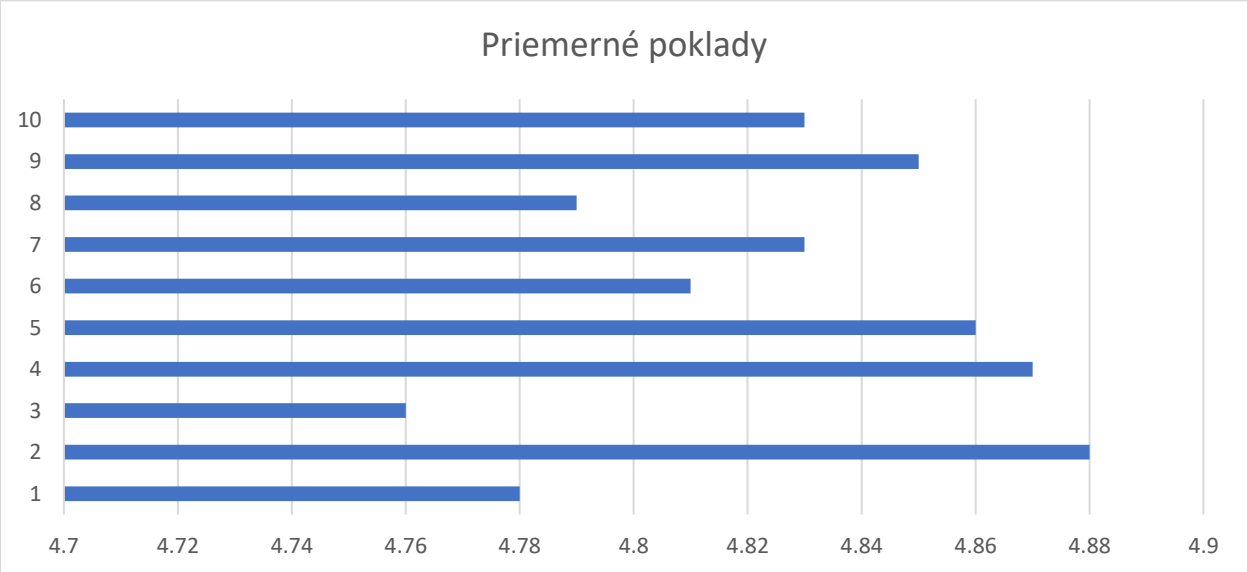
Priemerné fitness hodnoty: 2681.922



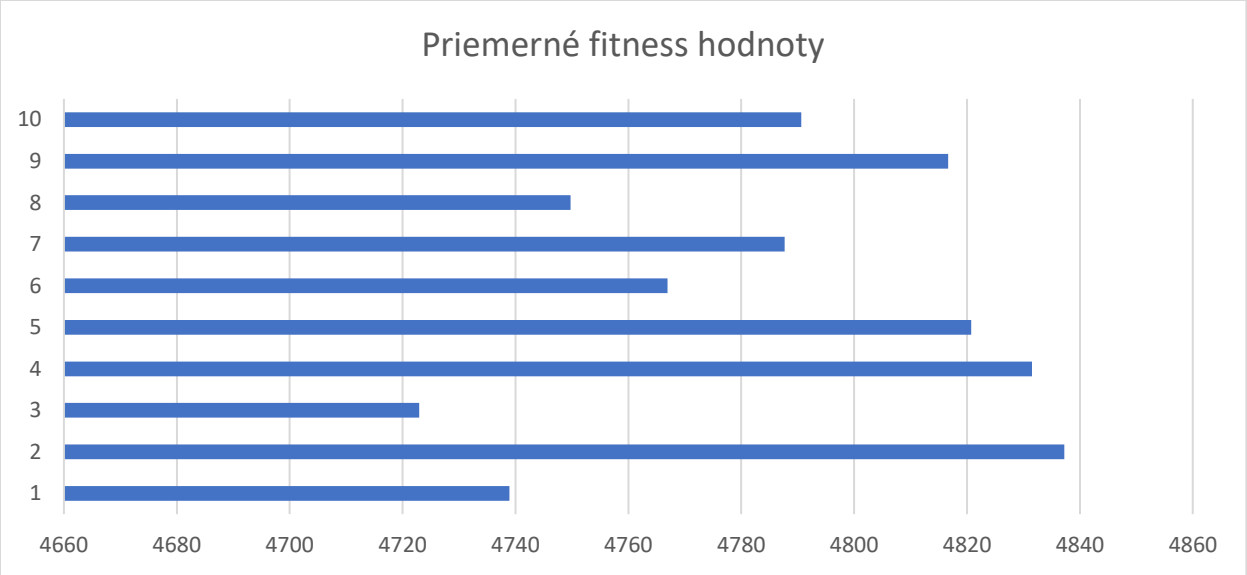
Priemerné kroky: 25.078

Poznámka:
Program bol určite pomalší, lebo vo všetkých prípadoch bolo potrebné vytvoriť 500 generácií, a to preto, lebo hľadač pokladov nenašiel všetky poklady.

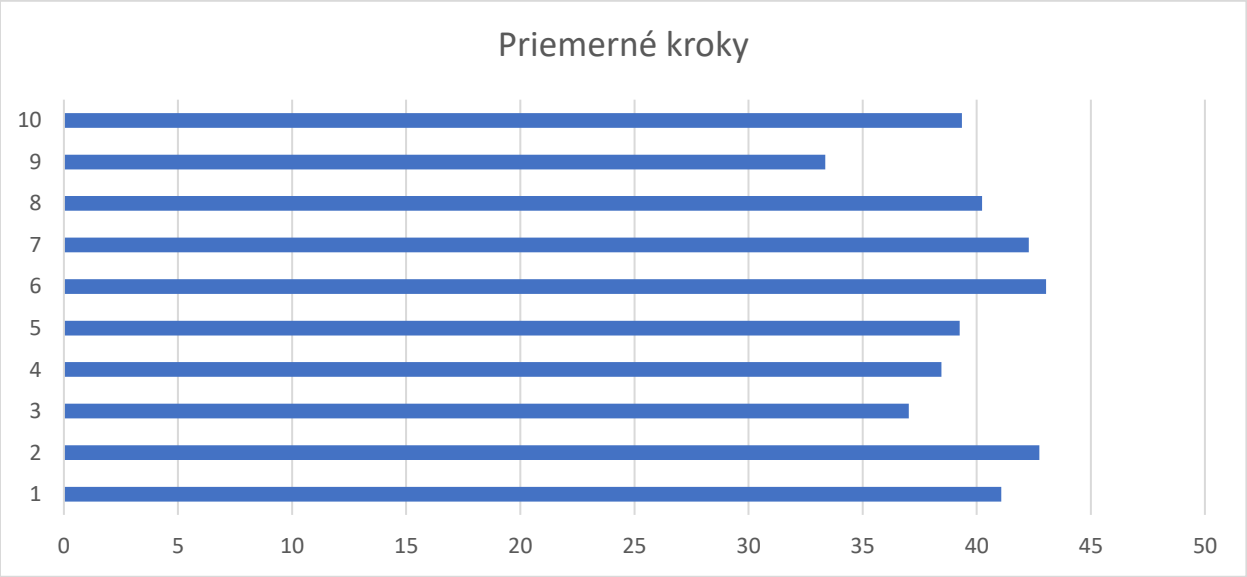
Test 3 – Používanie iba metódy turnaja: 20 elitov/80 jedincov vybraného turnaja



Priemerné poklady: 4.826

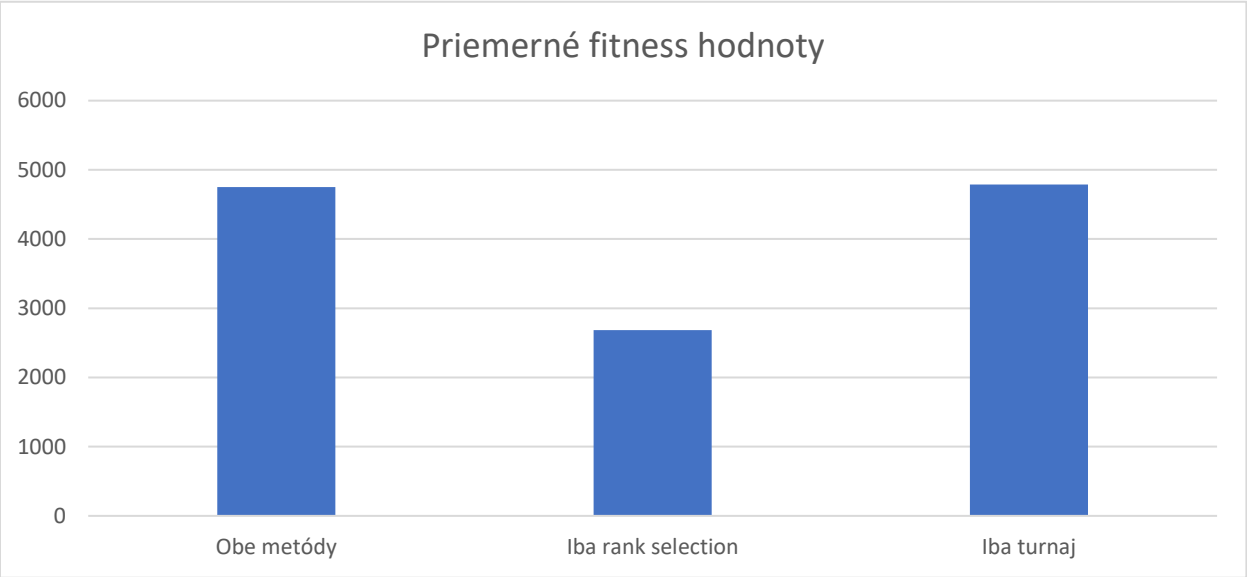


Priemerné fitness hodnoty: 4786.316



Priemerné kroky: 39.684

Vyhodnotenie výsledkov testu a zhrnutie



Priemerné fitness hodnoty:

1. Iba turnaj: 4786.316
2. Obe metódy: 4750.998
3. Iba rank selection: 2681.922

Podľa grafov vidíme, že najefektívnejšia metóda bola používať len metódu turnaja na selekcie rodičov. Selekcia podľa hodnosti nepomohla pri vylepšení algoritmu, a samostatne produkovala najhoršie výsledky. Pri implementovaní rank selection som musel sortovať populáciu podľa fitness hodnoty, nastaviť zhodnosť pre každého jedinca, a vložiť ich do spájaného zoznamu pre vytvorenie rulety. Vykonávať toľko inštrukcií pre program trvá oveľa dlhšie, ako v prípade metódy turnaja, kde sme vybrali náhodne jedincov z pôvodnej populácie a zbojili sme ich. Nakoniec u jedincoch, u ktorých rozdiel medzi fitness hodnotami bola veľká, často bola pravdepodobnosť výberu len trošku odlišná. To znamenalo, že aj jedinec s veľmi nízkou fitness hodnotou malo takmer toľko šance na vybratie, ako jedinec, ktorý mal o tisíce vyššiu fitness hodnotu.

Priestor na zlepšenie

Myslím si, že priestor ako zlepšiť algoritmus by bolo implementácia lepšej metódy selekcie namiesto rank selection. Avšak testy poukázali na to, že aj bez používania viacerých metód, len použitím metódy turnaja sme získali veľmi dobré výsledky.