Genetické algoritmy

Genetické algoritmy (GA) - sú triedou evolučných výpočtových metód inšpirovaných procesmi prirodzeného výberu a genetiky pozorovanými v prírode. Hlavnou myšlienkou GA je simulácia biologickej evolúcie na riešenie optimalizačných a vyhľadávacích problémov.

Hlavné vlastnosti genetických algoritmov

1. Populácia riešení

Genetický algoritmus nepracuje s jedným riešením, ale s populáciou možných riešení (tzv. jedincov). Každý jedinec v populácii je zakódovaný ako chromozóm, ktorý môže byť reprezentovaný napríklad ako binárna postupnosť alebo pole čísel.

2. Operátory genetickej evolúcie

Selekcia: proces, pri ktorom sa vyberajú najlepší jedinci (riešenia) na základe ich fitness (kvality). Lepší jedinci s väčšou pravdepodobnosťou zanechajú potomstvo.

Kríženie: Proces kombinovania chromozómov dvoch rodičov s cieľom vytvoriť jedného alebo viac potomkov. To umožňuje prenos dobrých vlastností z rodičov na deti.

Mutácia: náhodné zmeny v chromozómoch jedincov s cieľom zachovať genetickú rozmanitosť. Mutácia pomáha algoritmu vyhnúť sa uviaznutiu v lokálnych minimách a preskúmať nové oblasti priestoru riešenia.

3. Fitness funkcia

Každý jedinec v populácii sa hodnotí pomocou funkcie fitness, ktorá meria, aké dobré je riešenie reprezentované daným jedincom. Fitness funkcia závisí od konkrétneho problému a používa sa na vzájomné porovnávanie riešení.

4. Vývoj generácií

Proces evolúcie sa opakuje počas mnohých generácií. V každej fáze sa prostredníctvom selekcie, kríženia a mutácie vytvára nová generácia jedincov, ktorá postupne zvyšuje priemernú zdatnosť populácie.

5. Zastavenie algoritmu

Evolučný proces sa môže ukončiť, keď sa dosiahne určitý počet generácií, nájde sa riešenie s prijateľnou fitness alebo keď sa zastaví zvyšovanie fitness (žiadny pokrok).

Inšpirácia pre genetické algoritmy

Genetické algoritmy sú inšpirované teóriou evolúcie Charlesa Darwina a procesmi genetickej dedičnosti v biológii.
Napodobňujú princípy:

- Prírodný výber, kde majú vhodnejší jedinci väčšiu šancu na prežitie a odovzdanie svojich génov budúcim generáciám.
- Genetickú rozmanitosť, ktorú zabezpečujú mutácie a rekombinácia genetického materiálu prostredníctvom kríženia.
- Prežitie najsilnejších, kde evolučný proces vedie k vzniku jedincov, ktorí sú lepšie prispôsobení prostrediu (alebo riešeniu optimalizačného problému).

Genetické algoritmy sú založené na hypotéze, že kombinácia najlepších vlastností z rôznych riešení môže viesť k ešte lepším riešeniam v nasledujúcich generáciách.

Zdroje informácií o genetickom algoritme

1. *Holland, J.H. (1975). "Adaptation in Natural and Artificial Systems"*.
2. *Goldberg, D.E. (1989). "Genetic Algorithms in Search, Optimization, and Machine Learning"*.
3. *Mitchell, M. (1996). "An Introduction to Genetic Algorithms"*.
4. *Haupt, R.L., & Haupt, S.E. (2004). "Practical Genetic Algorithms"*.
5. *Deb, K. (2001). "Multi-Objective Optimization using Evolutionary Algorithms"*.
6. *Simon, D. (2013). "Evolutionary Optimization Algorithms"*.
7. *Whitley, D. (1994). "A Genetic Algorithm Tutorial"*.
8. *Srinivas, M., & Patnaik, L.M. (1994). "Genetic Algorithms: A Survey"*.
9. *Charbonneau, P. (1995). "Genetic Algorithms in Astronomy and Astrophysics"*.
10. *Fonseca, C.M., & Fleming, P.J. (1993). "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization"*.

Reálne príklady genetických algoritmov

- Vzdelávanie: rozvrhnúť hodiny alebo skúšky tak, aby sa predišlo konfliktom medzi učiteľmi, študentmi a zariadeniami.

- Doprava: Optimalizovať rozvrhy verejnej dopravy alebo letov s cieľom minimalizovať meškania a náklady.
- Navigácia: Hľadanie najlepšej cesty pre autonómneho robota v neznámom prostredí s cieľom vyhnúť sa prekážkam a dosiahnuť cieľ.
- Evolučný dizajn robotov: navrhovanie tvarov a správania robotov prostredníctvom evolučných stratégií s cieľom zlepšiť ich výkon v konkrétnych prostrediach.
- Mobilné siete: prideľovanie komunikačných kanálov s cieľom minimalizovať rušenie a maximalizovať kvalitu komunikácie.
- Internetová prevádzka: optimalizácia smerovania prevádzky s cieľom znížiť oneskorenie a zvýšiť účinnosť siete.
- Aerodynamická optimalizácia: hľadanie optimálnych tvarov lietadiel a automobilov s cieľom znížiť odpor vzduchu a zvýšiť palivovú účinnosť.
- Optimalizácia konštrukcií: napríklad navrhovanie mostov alebo budov s minimálnymi materiálovými nákladmi a maximálnou odolnosťou voči zaťaženiu.
- Hľadanie optimálnych portfólií: výber akcií alebo iných finančných nástrojov s cieľom maximalizovať výnosy a minimalizovať riziká.
- Algoritmické obchodovanie: optimalizácia obchodných stratégií pre automatizované obchodné systémy, ktoré reagujú na signály trhu.
- Zosúladenie sekvencií DNA: hľadanie najlepších spôsobov porovnávania genetických sekvencií s cieľom identifikovať podobnosti a rozdiely medzi rôznymi organizmami.
- Zostavovanie genómov: optimalizácia procesu zostavovania sekvencií genómov s cieľom umožniť rýchlejšiu a presnejšiu identifikáciu génov.
- Smerovanie vozidiel: hľadanie najlepších trás pre nákladné vozidlá alebo iné vozidlá s cieľom znížiť náklady na palivo a čas doručenia.
- Riadenie zásob: optimalizácia rozloženia zásob v skladoch a času dodania tovaru s cieľom zabrániť nedostatku alebo prebytku zásob.
- Dolaďovanie zariadení: hľadanie optimálnych prevádzkových režimov pre stroje alebo zariadenia s cieľom znížiť spotrebu energie a zvýšiť produktivitu.

- Plánovanie výroby: optimalizácia poradia operácií v továrňach s cieľom znížiť prestoje a zvýšiť efektívnosť.
- Umelá inteligencia postáv: Optimalizácia správania virtuálnych protivníkov alebo NPC (nehráčskych postáv), aby boli pre hráčov prispôsobivejší a náročnejší.
- Generovanie úrovní: automatické vytváranie komplexných úrovní a miest pre hry na základe náhodných a evolučných princípov.
- Diagnostika chorôb: analýza veľkého množstva údajov o pacientoch s cieľom identifikovať vzory, ktoré môžu naznačovať prítomnosť choroby alebo genetickej predispozície k nej.

Príklady a porovnanie niektorých algoritmov

1. Klasický genetický algoritmus (CGA)

Klasický genetický algoritmus je základom všetkých variantov GA. Používa základné operátory genetickej evolúcie - selekciu, kríženie a mutáciu.

- Je jednoduchý a univerzálny.
- Na výber rodičov používa turnajovú alebo ruletovú metódu.
- Je dôležité správne nastaviť pravdepodobnosti kríženia a mutácie, aby sa zabezpečil stabilný vývoj.

2.

```
input: π, max_generations, population_size, gene_length, crossover_rate, mutation_rate
output: best individual r ∈ S

r = ∅
g(r) = -∞  // Initialize worst fitness
i = 1      // Generation counter

repeat
   population = initialize_population(population_size, gene_length)

   for each individual s ∈ population
      f(s) = compute_fitness(s)
   end_for

   while (# (population) > 0)
      parent1 = selection(population, f)
      parent2 = selection(population, f)
      child1, child2 = crossover(parent1, parent2, crossover_rate)
      child1 = mutation(child1, mutation_rate)
      child2 = mutation(child2, mutation_rate)

      if (g(child1) > g(r)) then
         r = child1
      end_if

      if (g(child2) > g(r)) then
         r = child2
      end_if
   end_while

   i = i + 1

until i > max_generations

if (valid(r)) then
   return r
else
   return ∅
```

Genetický algoritmus s elitizmom (EGA)

Elitizmus je vylepšením klasického genetického algoritmu, pretože najlepší jedinci (jedinci s najvyššou fitness) sa automaticky prenášajú do ďalšej generácie v nezmenenej podobe. Tým sa zabezpečí, že najlepšie riešenie predchádzajúcej generácie sa nestratí v dôsledku náhodných mutácií alebo zlého kríženia.

- Zaručený prenos najlepších riešení do ďalšej generácie.
- Znižuje riziko straty dobrých riešení v dôsledku náhodnosti evolučného procesu.
- Zvyšuje stabilitu, ale môže znížiť rozmanitosť populácie.

```
Input: π (problem instance), max_generations, population_size, gene_length, crossover_rate, mutation_rate
Output: best individual r ∈ S
1. Initialize:
   r ← ∅                          // Best individual
   g(r) = −∞                      // Initialize worst fitness
   i = 1                         // Generation counter
2. Repeat:
   a. population ← initialize_population(population_size, gene_length)
   b. For each individual s in population:
       f(s) = compute_fitness(s)     // Evaluate fitness for each individual
     end_for
   c. Apply Elitism:
       elite_individuals ← SelectBestIndividuals(population, f, ELITE_SIZE)  // Keep the top elite individuals
   d. While (|population| > 0):
     - parent1 = selection(population, f)       // Select parents
     - parent2 = selection(population, f)
     - child1, child2 = crossover(parent1, parent2, crossover_rate)   // Perform crossover
     - child1 = mutation(child1, mutation_rate)   // Apply mutation
     - child2 = mutation(child2, mutation_rate)
     - If g(child1) > g(r):        // Update best individual
         r = child1
       End_If
     - If g(child2) > g(r):
         r = child2
       End_If
   end_while
   e. Create new population:
     population ← elite_individuals + offspring[0: POPULATION_SIZE - ELITE_SIZE]
   f. Increment generation counter:
     i = i + 1
3. Until i > max_generations
4. Final check:
   If valid(r):
     Return r
   Else:
     Return ∅
```

3. Paralelný genetický algoritmus (PGA)

Paralelný genetický algoritmus rozdeľuje populáciu na niekoľko subpopulácií, ktoré sa vyvíjajú nezávisle od seba. Medzi subpopuláciami sa periodicky vymieňajú jedinci, aby sa zachovala rozmanitosť a zlepšila účinnosť vyhľadávania.

- Zvyšuje rýchlosť konvergencie vďaka paralelnému vykonávaniu.
- Zlepšuje globálne vyhľadávanie prostredníctvom migrácie medzi subpopuláciami.
- Používa sa pri rozsiahlych problémoch, kde si tradičné metódy vyžadujú značné zdroje.

```
Input: POPULATION_SIZE, GENE_LENGTH, MAX_GENERATIONS, CROSSOVER_RATE, MUTATION_RATE, ISLANDS, MIGRATION_INTERVAL, MIGRATION_RATE
Output: Best individual from all islands

Function fitness(x):
  Return x^2  // Maximize function f(x) = x^2

Function binary_to_decimal(binary):
  Return integer value of binary string

Function initialize_population():
  Create empty list population
  For each _ from 0 to POPULATION_SIZE - 1:
    chromosome = randomly generate a binary string of length GENE_LENGTH
    Add chromosome to population
  Return population

Function crossover(parent1, parent2):
  If random() < CROSSOVER_RATE:
    point = random integer from 1 to GENE_LENGTH - 1
    child1 = parent1[0:point] + parent2[point:]
    child2 = parent2[0:point] + parent1[point:]
    Return child1, child2
  Return parent1, parent2

Function mutate(chromosome):
  mutated = empty string
  For each gene in chromosome:
    If random() < MUTATION_RATE:
      mutated += '1' if gene is '0' else '0'
    Else:
      mutated += gene
  Return mutated

Function select_parent(population, fitness_values):
  total_fitness = sum of fitness_values
  pick = random number between 0 and total_fitness
  current = 0
  For each individual in population with index i:
    current += fitness_values[i]
    If current > pick:
      Return individual

Function evolve_island(island_id, population, generations):
  For generation from 0 to generations - 1:
    fitness_values = array of fitness values for each ind in population
    best_individual = population[index of max(fitness_values)]
    best_value = binary_to_decimal(best_individual)
    Print("Island " + island_id + ", Generation " + (generation + 1) + ": Best solution = " + best_value + ", Fitness = " + fitness(best_value))

    new_population = empty list
    While length of new_population < POPULATION_SIZE:
      parent1 = select_parent(population, fitness_values)
      parent2 = select_parent(population, fitness_values)
      child1, child2 = crossover(parent1, parent2)
      Add mutate(child1) to new_population
      If length of new_population < POPULATION_SIZE:
        Add mutate(child2) to new_population

    population = new_population

    If (generation + 1) % MIGRATION_INTERVAL == 0:
      Return population  // Return population for migration

  Return population


Function parallel_genetic_algorithm():
  // Create isolated islands
  islands = array of ISLANDS initialized with populations from initialize_population()

  For generation from 0 to MAX_GENERATIONS with step MIGRATION_INTERVAL:
    // Evolve each island in parallel
    Execute evolve_island for each island using parallel processing
    // Handle migration between islands
    For each island_id from 0 to ISLANDS - 1:
      migrants = random sample of MIGRATION_RATE individuals from islands[island_id]
      target_island = (island_id + 1) % ISLANDS  // Next island for migration
      Replace first MIGRATION_RATE individuals in islands[target_island] with migrants

// Start the algorithm
parallel_genetic_algorithm()
```

Algoritmus	Funkcie	Výhody	Nevýhody
Klasický GA	Štandardná schéma s križením a mutáciou	Jednoduchá implementácia, univerzálnosť	Môže uviaznuť v lokálnych extrémoch
GA s elitárstvom	Zachovanie najlepších riešení	Zaručenie zachovania najlepších riešení	Znižuje rozmanitosť populácie
Paralelné GA	Viaceré subpopulácie s migráciou	Zvyšuje diverzitu a zlepšuje globálne vyhľadávanie	Zložitosť implementácie, vyžaduje viac zdrojov

Príklad úloh, ktoré možno riešiť pomocou týchto algoritmov

1. Klasický genetický algoritmus (CGA)

- Problém obchodného cestujúceho, problém batoha, plánovanie procesov, plánovanie.
- Hľadanie najkratšej trasy cez dané mestá je klasickým príkladom kombinatorickej optimalizácie. CGA môže hľadať približnú trajektóriu postupným zlepšovaním riešenia križením a mutáciou postupností miest.
- CGA možno použiť na nájdenie najlepších hyperparametrov, ako je počet vrstiev neurónovej siete, veľkosť trénovacej množiny alebo počet stromov v rozhodovacom lese.

2. Genetický algoritmus s elitizmom

- Optimalizácia konštrukcií, technologických procesov, elektronických obvodov.
- Pomocou elitizmu možno zachovať najlepšie riešenia, čím sa zabezpečí, že sa nestratia žiadne efektívne riešenia. Napríklad pri návrhu mosta môže algoritmus nájsť optimálne parametre na min materiálových nákladov a zabezpečenie stability konštrukcie.
- Plánovanie trajektórie, prideľovanie zdrojov pre roboty.
- GA s elitizmom sa môžu použiť na nájdenie optimálnych dráh pre roboty vo výrobnom prostredí alebo v autonómnych vozidlách. Elitizmus zabezpečuje zachovanie najlepších trajektórií počas evolúcie.

3. Paralelný genetický algoritmus (PGA)

- Vo veľkovýrobe možno optimalizovať niekoľko premenných súčasne (čas výroby, spotreba materiálu a energie). Paralelný vývoj umožňuje rýchlejšie nájsť optimálne riešenia pre zložité modely.
- Riešenie komplexných problémov v genetike a molekulárnej biológii.
- PGA umožňuje paralelné vyhľadávanie najlepšieho zarovnania genomických sekvencií, čo urýchľuje analýzu údajov v bioinformatike.
- Tréning agentov v prostrediach so zložitou dynamikou.
- V systémoch učenia s posilňovaním môže PGA paralelne trénovať viacerých agentov s cieľom nájsť najlepšie stratégie v zložitých herných alebo reálnych prostrediach.

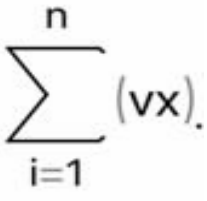
Problém s batohom (Knapsack Problem)

Opis problému:

Problém batohu je klasický optimalizačný problém, ktorého cieľom je maximalizovať hodnotu predmetov, ktoré možno umiestniť do batohu s určitou kapacitou. Úlohou je vybrať súbor predmetov so známou hmotnosťou a hodnotou tak, aby celková hmotnosť nepresiahla kapacitu batohu a aby sa maximalizovala celková hodnota.

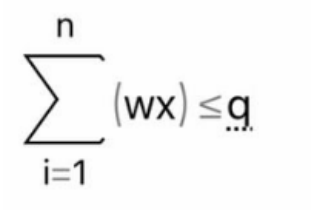
Účel:

1. Vstupné údaje:
    - Súbor položiek n, z ktorých každá má: váhu w a hodnotu v ;
    - Kapacita batohu q (maximálna hmotnosť, ktorú možno vziať);
2. Cieľová funkcia:
    - Maximalizovať celkovú hodnotu predmetov v batohu:



kde x - je binárna premenná, ktorá udáva, či je položka zahrnutá i je zahrnutá v batohu (1 - zahrnutá, 0 — nezahrnutá).

3. Obmedzenia:
    - Celková hmotnosť položiek nesmie prekročiť kapacitu batohu:



Model na testovanie algoritmu

1 Vytvorte pole objektov, napríklad:

Položka 1: váha = 2, hodnota = 3

Položka 2: váha = 3, hodnota = 4

Položka 3: hmotnosť = 4, hodnota = 5

Položka 4: hmotnosť = 5, hodnota = 6

Kapacita batohu q=5.

2 Populácia:

Vytvorte počiatočnú populáciu chromozómov, kde každý chromozóm je binárnou reprezentáciou množiny položiek (napr. [1, 0, 1, 0], čo znamená, že sú zahrnuté položky 1 a 3).

3 Pre každý chromozóm vypočítame:

* Celkovú hmotnosť položiek v batohu.

* Celkovú hodnotu položiek v batohu.

Ak hmotnosť presahuje kapacitu , q fitness je 0. V opačnom prípade je fitness rovná celkovej hodnote.

4 Evolučný proces:

* Na vytvorenie nových generácií používame operátory selekcie, kríženia a mutácie.

* Paralelná evolúcia na niekoľkých „ostrovoch“ na zachovanie genetickej rozmanitosti.

5 Ukončenie:

Algoritmus sa ukončí, keď sa dosiahne maximálny počet generácií alebo keď sa nájde vyhovujúce riešenie (maximálna hodnota).

Prvý experiment: Zmena počtu generácií v genetickom algoritme

V tomto experimente meníme počet generácií v genetickom algoritme, aby sme zistili, ako to ovplyvňuje efektívnosť riešenia. Počet generácií sa mení od 100 do 1100 s krokom 100. Väčší počet generácií môže viesť k lepšiemu výsledku.

Graf
ukazuje, ako počet generácií ovplyvňuje najlepšiu prispôsobivosť batohu. So zvyšovaním počtu generácií pozorujeme zlepšenie výsledku, pretože genetický algoritmus má viac príležitostí na evolúciu a hľadanie optimálneho riešenia. Po prekročení určitej úrovne však môže byť ďalšie zlepšenie zanedbateľné.

Druhý experiment: Zmena veľkosti populácie

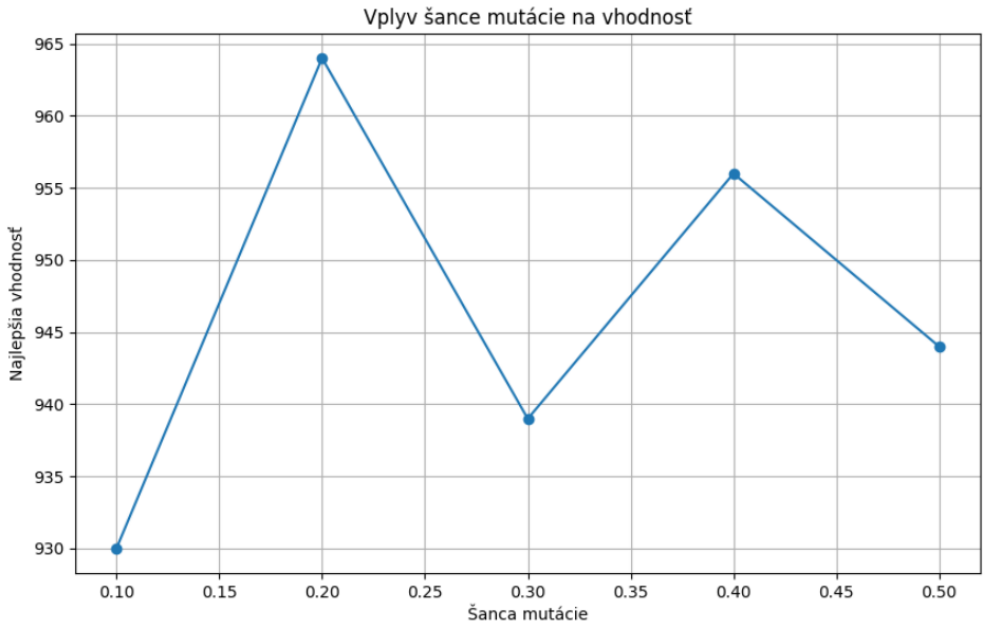
V tomto experimente skúmame, ako veľkosť populácie (veľkosť populácie) v genetickom algoritme ovplyvňuje najlepšiu prispôsobivosť riešenia problému batohu. Veľkosť populácie sa mení od 50 do 500 s krokom 50, pričom počet generácií je fixovaný na 500. Cieľom je zistiť, či väčšia populácia vedie k efektívnejšiemu riešeniu úlohy, a tiež určiť, či existuje hranica, po ktorej ďalšie zvyšovanie populácie neprináša významné zlepšenie.

Graf ukazuje, ako zväčšovanie populácie ovplyvňuje najlepšiu prispôsobivosť riešenia. Vo všeobecnosti možno očakávať, že so zväčšovaním populácie genetický algoritmus nájde lepšie riešenia, pretože väčší počet jedincov umožňuje lepšie skúmanie priestoru možných riešení. Avšak so zvyšovaním populácie tiež rastú výpočtové zdroje potrebné na chod algoritmu a po určitej hranici sa zlepšenie môže stať nevýznamným alebo úplne zastaviť.

Tretí experiment: Vplyv šance na mutáciu

V tomto experimente skúmame, ako šanca na mutáciu (Šanca mutácie) v genetickom algoritme ovplyvňuje najlepšiu prispôsobivosť riešení problému batohu. Šanca na mutáciu sa mení od 0,1 do 0,5, pričom počet generácií je fixovaný na 500, čo umožňuje zistiť, ako zmena tohto parametra ovplyvňuje rozmanitosť a kvalitu riešení

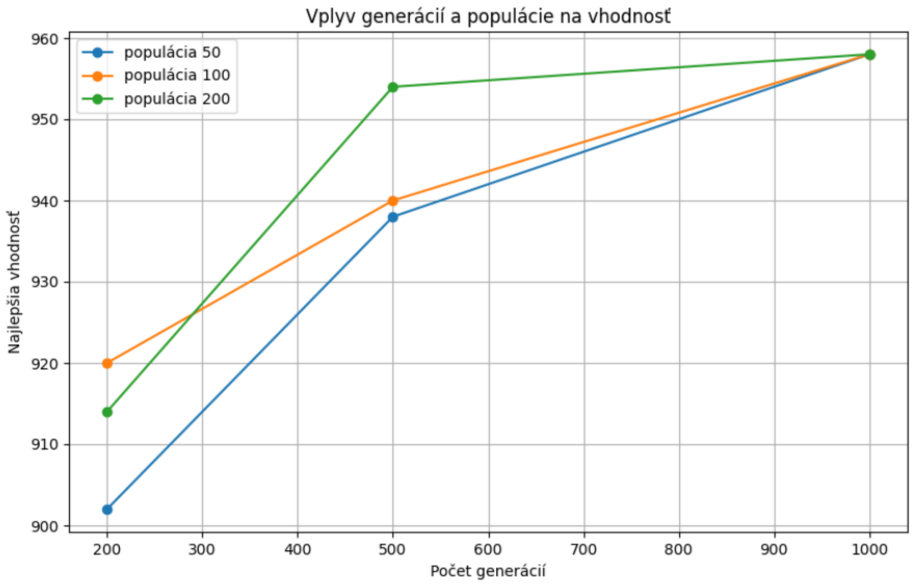
v populácii. Hlavným cieľom je zistiť, či zvyšovanie šance na mutáciu vedie k lepším riešeniam, alebo naopak zhoršuje stabilitu už nájdených riešení.
Graf ukazuje závislosť medzi šancou na mutáciu a najlepšou prispôsobivosťou riešení. Z celkovej analýzy možno očakávať, že so zvyšovaním šance na mutáciu sa zvýši rozmanitosť riešení, ale môže to tiež negatívne ovplyvniť stabilitu a kvalitu dosiahnutých výsledkov. Pri veľmi vysokých hodnotách šance na mutáciu môže



genetický algoritmus začať generovať náhodné riešenia, čo znemožní dosiahnutie optimálnych výsledkov

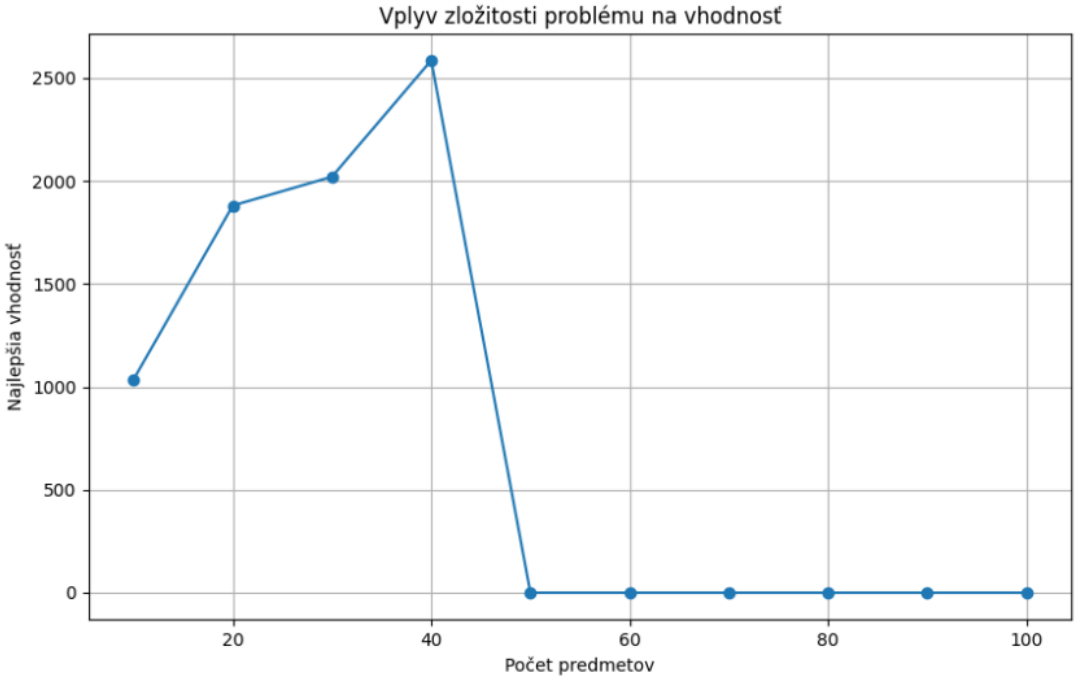
Štvrtý experiment: Vplyv počtu generácií a veľkosti populácie

V tomto experimente súčasne meníme počet generácií a veľkosť populácie, aby sme analyzovali ich vplyv na prispôsobivosť riešení v úlohe batohu. Pomocou troch rôznych hodnôt pre každý parameter (50, 100 a 200 pre veľkosť populácie a 200, 500 a 1000 pre počet generácií) sa snažíme zistiť, ako tieto faktory interagujú a ako ich kombinácia ovplyvňuje efektívnosť algoritmu. Očakáva sa, že väčší počet generácií a väčšia veľkosť populácie povedie k zlepšeniu výsledkov, avšak to tiež predĺži čas potrebný na vykonanie algoritmu.



Piaty experiment: Vplyv šance na mutáciu

V tomto experimente meníme počet predmetov v úlohe batohu, aby sme zistili, ako to ovplyvňuje prispôsobivosť riešení. Zvyšovanie počtu predmetov môže úlohu skomplikovať, čo môže vyžadovať viac generácií alebo väčšiu populáciu na dosiahnutie kvalitného riešenia. Cieľom experimentu je zistiť, ako nárast počtu predmetov ovplyvní priemernú najlepšiu prispôsobivosť riešení a či sa úloha stáva náročnejšou pre algoritmus.



Graf ukazuje, ako zmena počtu predmetov ovplyvňuje najlepšiu prispôsobivosť riešení v úlohe batohu. Vo všeobecnosti možno očakávať, že so zvyšovaním počtu predmetov sa prispôsobivosť riešení môže znižovať, pretože úloha sa stáva náročnejšou, čo môže vyžadovať väčšie úsilie na nájdenie optimálneho riešenia.

Výsledky však môžu tiež naznačiť, že algoritmus je schopný nachádzať dobré riešenia aj pri väčšom počte predmetov, čo zdôrazňuje dôležitosť parametrov, ako je veľkosť populácie a počet generácií.

Záver experimentu

V priebehu experimentov sme zistili, že optimalizácia parametrov genetického algoritmu má zásadný vplyv na kvalitu riešení problému ruksaku. Po analýze rôznych nastavení môžeme konštatovať, že kombinácia 1000 generácií a populácie s 300 jedincami priniesla najlepšie výsledky, pričom sa dosiahla maximálna prispôsobivosť v rozmedzí 95-98% od teoreticky optimálneho riešenia. Toto nastavenie umožnilo genetickému algoritmu dostatočne preskúmať priestor riešení a zabezpečilo dobrú rovnováhu medzi diverzifikáciou a stabilitou v populácii.