UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Hopfieldova síť a problém cestujícího

2024 Lucie Scholzová

Obsah

Seznam obrázků

Seznam zdrojových kódů

Úvod

Problém obchodního cestujícího (Traveling Salesman Problem, TSP) je klasický optimalizační problém, který nalézá uplatnění v mnoha oblastech, od logistiky po plánování a organizaci dat v počítačových vědách. Jde o úlohu, kde cestující musí navštívit několik měst a vrátit se do výchozího města, přičemž cílem je minimalizovat celkovou délku cesty. Kvůli své NP-těžkosti a široké aplikační oblasti byly vyvinuty desítky různých algoritmů pro jeho řešení.

Tato práce se primárně zaměřuje na Hopfieldovu síť, což je forma rekurentní neuronové sítě, která byla původně navržena pro řešení problémů asociativní paměti, ale ukazuje se, že má zajímavé aplikace i v optimalizačních problémech jako je TSP. Cílem této analýzy je prozkoumat efektivitu, omezení a praktickou aplikovatelnost Hopfieldovy sítě v kontextu TSP a srovnat ji s dalšími dvěma populárními algoritmy: genetickým algoritmem a mravenčím algoritmem.

Genetické algoritmy využívají principy evoluční biologie, jako je selekce, křížení a mutace, k optimalizaci řešení. Na druhou stranu, mravenčí algoritmy simulují chování mravenců při hledání potravy k nalezení nejkratších cest v grafech, což je přirozeně podobné TSP.

Tato práce poskytne hlubší porozumění výhodám a nevýhodám Hopfieldovy sítě ve srovnání s těmito algoritmy a nastíní potenciální oblasti pro další výzkum. Důraz bude kladen na teoretické podklady algoritmů, jejich implementaci v jazyce Python a analytické porovnání jejich výkonu pomocí dvou klíčových metrik: rychlosti a přesnosti řešení TSP.

1. Hopfieldova síť

Hopfieldova síť je forma rekurentní neuronové sítě, která se vyznačuje svou schopností uchovávat informace ve stabilních vzorech a může být použita pro řešení různých optimalizačních úloh, včetně problému obchodního cestujícího (TSP). Tato část poskytuje hlubší pohled na principy fungování Hopfieldovy sítě, její aplikaci na TSP a diskutuje o jejích výhodách a omezeních.

Hopfieldova síť se skládá z jedné vrstvy neuronů, přičemž každý neuron je propojen se všemi ostatními neurony, kromě sebe sama. Každé spojení má váhu, která určuje sílu vlivu mezi dvěma neurony. Tyto váhy jsou symetrické, což znamená, že váha mezi neuronem A a B je stejná jako váha mezi neuronem B a A.

Stav sítě je reprezentován vektorem binárních hodnot, kde každý neuron může nabývat hodnoty +1 (aktivní) nebo -1 (neaktivní). Aktualizace stavu sítě probíhá iterativně pomocí asynchronního update, kde se stav každého neuronu aktualizuje na základě vážené sumy jeho vstupů a aplikace prahové funkce, typicky signum funkce, která určuje, zda bude neuron aktivní či neaktivní.

Základem Hopfieldovy sítě je energetická funkce, která je definována pro každý možný stav sítě. Cílem je dosáhnout stavu, ve kterém je tato energetická funkce minimalizována. Energetická funkce má tvar:

$$E = -\frac{1}{2}\sum_{i \neq j} w_{ij} s_i s_j + \sum_i \theta_i s_i$$

kde s_i a s_j jsou stavy i-tého a j-tého neuronu, w_{ij} jsou váhy spojení mezi neurony a θ_i jsou prahové hodnoty pro i-tý neuron.

Pro aplikaci Hopfieldovy sítě na TSP se musí přizpůsobit energetická funkce tak, aby penalizovala nevalidní cesty a odměňovala kratší cesty. To se obvykle dosahuje nastavením vah spojení mezi neurony tak, aby odpovídaly vzdálenostem mezi městy v TSP. V Hopfieldově síti pro TSP každý neuron odpovídá jedné pozici (městu) v cestě, a síť se snaží najít konfiguraci, kde každé město je navštíveno právě jednou a celková cesta je co nejkratší.

Hopfieldovy sítě mohou efektivně najít dobrá řešení pro menší instance TSP díky své schopnosti rychle konvergovat ke stabilním stavům. Nicméně, pro větší instance problému mohou být výsledky méně spolehlivé a síť může uváznout v lokálních minimech energetické funkce, což brání nalezení globálně optimálního řešení. Toto omezení je možné částečně překonat pomocí různých technik pro optimalizaci parametrů sítě a průzkum různých počátečních stavů.

2. Porovnávané algoritmy

2.1. Genetický algoritmus

Genetický algoritmus (GA) je inspirován principy přírodní evoluce, jako jsou selekce, křížení a mutace. Algoritmus začíná s populací potenciálních řešení, zvaných jedinci, které se postupně vyvíjejí.

Populace jedinců je typicky inicializována náhodně. Každý jedinec reprezentuje možné řešení problému TSP ve formě cesty mezi městy.

Jedinci jsou vybíráni na základě jejich fitness, což je měřítko, jak dobře jedinec řeší daný problém. Vybraní jedinci jsou poté použiti pro generování potomků pomocí křížení a mutace.

Křížení zahrnuje kombinaci genetického materiálu dvou jedinců, což vede k novému jedinci. Mutace zavádí náhodné změny do genetického kódu jedince, což podporuje diverzitu v populaci.

2.2. Mravenčí algoritmus

Mravenčí algoritmus (Ant Colony Optimization, ACO) simuluje chování mravenců hledajících cestu k potravě. Algoritmus je účinný při hledání optimalizovaných cest v grafech, což ho činí vhodným pro TSP.

Mravenci komunikují a směřují své chování pomocí feromonů, chemických stop, které zanechávají na cestě. V ACO, vyšší koncentrace feromonů na cestě znamená, že cesta byla často používána a je potenciálně výhodná.

Po každé iteraci se množství feromonů na cestách aktualizuje na základě kvality řešení, které mravenci našli. Tím se podporuje průzkum efektivnějších cest v budoucích iteracích.

3. Teoretické porovnání algoritmů

V této kapitole poskytujeme teoretické porovnání tří různých přístupů k řešení problému obchodního cestujícího (TSP): Hopfieldovy sítě, genetického algoritmu a mravenčího algoritmu. Každý z těchto algoritmů přináší specifické charakteristiky, které ovlivňují jejich výkonnost, adaptabilitu a efektivitu v různých scénářích.

Hopfieldova síť je model neuronové sítě vhodný pro řešení optimalizačních problémů díky své schopnosti uchovávat informace v dynamických vzorech a hledat energetická minima. Efektivní je především pro menší instance TSP, ale může narazit na problémy s uvíznutím v lokálních minimech u větších a složitějších úloh.

Genetický algoritmus, založený na principu přírodní selekce a evoluce, je silný v prozkoumávání velkého prostoru řešení a je schopen efektivně nalézat optimální nebo suboptimální řešení pro rozsáhlé a komplexní instance TSP. Díky možnostem nastavení operátorů křížení a mutace je robustní a vysoce adaptabilní.

Mravenčí algoritmus, který simuluje chování mravenců při hledání potravy, je založen na principu optimálního vyhledávání cesty pomocí feromonových stop. Podporuje kooperativní průzkum a optimalizaci, což jej činí velmi účinným při hledání řešení pro TSP. Je dobře adaptovatelný na různé varianty TSP a efektivní při dynamickém aktualizování feromonových stop na základě kvality nalezených řešení.

Hopfieldova síť nabízí rychlou konvergenci pro malé instance a schopnost přechodu mezi různými řešeními pro nalezení energetického minima, ale má problémy s uvíznutím v lokálních minimech a obtížnější nastavení parametrů pro velké instance TSP. Genetický algoritmus umožňuje efektivní prozkoumávání velkého prostoru řešení s vysokou adaptabilitou, avšak může být výpočetně náročný a riskuje nalezení lokálního minima. Mravenčí algoritmus exceluje v nalezení optimálních cest a adaptaci na změny, ale vyžaduje složitou parametrizaci a může být časově náročný v závislosti na velikosti problému a počtu mravenců.

Hopfieldova síť, genetický algoritmus a mravenčí algoritmus poskytují robustní řešení pro širší spektrum problémů s různou mírou flexibility a adaptability. Genetické algoritmy jsou účinné v rozšiřování hledání řešení, zatímco mravenčí algoritmy se osvědčují v nalezení optimálních cest díky své kooperativní a adaptivní strategii. Výběr nejlepšího algoritmu závisí na specifických požadavcích problému, dostupných zdrojích a požadované přesnosti řešení.

4. Implementace Hopfieldovy sítě

Tato kapitola se zaměřuje na implementaci Hopfieldovy sítě pro řešení problému obchodního cestujícího (TSP) v jazyce Python. Uvedený kód demonstruje, jak může být Hopfieldova síť použita k nalezení nejkratší možné cesty mezi skupinou měst (hospod) s použitím konceptu energetických funkcí.

Implementace začíná definicí třídy `HopfieldovaSit`, která obsahuje několik klíčových metod pro inicializaci, aktualizaci stavů a hledání nejlepší cesty.

Metoda __init__ inicializuje třídu s potřebnými parametry, včetně správce hospod (objekt, který uchovává informace o městech), velikosti problému (počtu měst) a počtu iterací pro proces aktualizace. Také obsahuje konfiguraci, která obsahuje hodnoty pro penalizaci a krok pro gradient descent.

```
import numpy as np

class HopfieldConfiguration:
   A: float = 500  # Penalizace za aktivaci více neuronů ve stejném řádku
   B: float = 500  # Penalizace za aktivaci více neuronů ve stejném sloupci
   C: float = 200  # Penalizace za aktivaci více nebo méně neuronů než je měst
   D: float = 500  # Penalizace za vzdálenost
   u0: float = 0.02  # Základní hodnota pro aktivaci neuronů, použitá v sigmoideální aktivační
funkci
   step: float = 1e-6  # Krok pro gradient descent algoritmus, použitý k aktualizaci vah sítě

class HopfieldovaSit:
      def __init__(self, spravce_hospod, iterace, config=HopfieldConfiguration):
      self.spravce_hospod = spravce_hospod
      self.velikost = len(spravce_hospod.hospody)
      self.iterace = iterace
      self.config = config
      self.energie = np.zeros((self.velikost, self.velikost))  # Matice energií pro každý pár uzlů
      self.neurony = np.random.uniform(0, 1, (self.velikost, self.velikost))  # Inicializace neuronů
náhodnými hodnotami
```

Zdrojový kód 1 HopfieldovaSit __init__

Metoda inicializuj_energie nastavuje energetickou matici sítě, kde elementy matice odpovídají záporné vzdálenosti mezi městy (hospodami). Tím síť "preferuje" kratší cesty. Diagonální prvky jsou nastaveny na nekonečno, aby se zabránilo výběru cesty, kde by město navštívilo samo sebe.

```
   def inicializuj_energie(self):
      for i in range(self.velikost):
         for j in range(self.velikost):
            souradnice1 = self.spravce_hospod.ziskej_souradnice(i)
            souradnice2 = self.spravce_hospod.ziskej_souradnice(j)
            if i != j:
               self.energie[i, j] = -vypocti_vzdalenost(souradnice1, souradnice2)
            else:
```

```
            self.energie[i, j] = float('inf')
```
Zdrojový kód 2 HopfieldovaSit metoda inicializuj_energie

Metoda update slouží k aktualizaci energetických stavů sítě. V této implementaci se pro každou iteraci aktualizují všechny prvky energetické matice a provádí se pokus o zlepšení energetického stavu sítě prostřednictvím posunu (rotace) energií. Pro každý prvek matice se vypočítá změna energetického stavu na základě vzdálenosti a penalizací. Nakonec je použita skoková funkce pro aktivaci neuronu.

```
    def update(self):
    for _ in range(self.iterace):
      for i in range(self.velikost):
        for j in range(self.velikost):
          if i != j:
            u_ij = self.neurony[i, j]
            delta_u_ij = -self.config.D * self.energie[i, j]
            delta_u_ij += -self.config.A * (np.sum(self.neurony[i, :]) - self.neurony[i, j])  #
Penalizace pro řádky
            delta_u_ij += -self.config.B * (np.sum(self.neurony[:, j]) - self.neurony[i, j])  #
Penalizace pro sloupce
            delta_u_ij += -self.config.C * (np.sum(self.neurony) - self.velikost)  # Penalizace pro
počet aktivací

            u_ij += self.config.step * delta_u_ij
            # Skoková aktivační funkce
            if u_ij >= 0:
              self.neurony[i, j] = 1
            else:
              self.neurony[i, j] = -1
```
Zdrojový kód 3 HopfieldovaSit metoda update

Metoda najdi_nejlepsi_cestu vyhledává nejlepší cestu na základě energetické matice, hledáním sloupce s nejnižší celkovou energií, což indikuje optimální pořadí měst v cestě.

```
  def najdi_nejlepsi_cestu(self):
    cesta = []
    navstiveno = set()
    for i in range(self.velikost):
      max_index = np.argmax(self.neurony[i, :])
      while max_index in navstiveno:
        self.neurony[i, max_index] = 0  # Vynulovat opakovaně navštívené místo
        max_index = np.argmax(self.neurony[i, :])
      cesta.append(max_index)
      navstiveno.add(max_index)
    return cesta
```
Zdrojový kód 4 HopfieldovaSit metoda najdi_nejlepsi_cestu

Metoda solve integruje všechny kroky dohromady, inicializuje energie, provádí aktualizace, a nakonec vrací nejlepší nalezenou cestu spolu s celkovou vzdáleností a identifikací metody.

```
    def solve(self):
        self.inicializuj_energie()
        self.update()
        nejlepsi_cesta = self.najdi_nejlepsi_cestu()
        return (
            self.spravce_hospod.dej_informace_o_nejlepsi_cesta(nejlepsi_cesta),
            self.spravce_hospod.vypocti_celkovou_vzdalenost(nejlepsi_cesta),
            'Hopfieldova síť'
        )
```

Zdrojový kód 5 HopfieldovaSit metoda solve

Tato implementace ukazuje základní aplikaci Hopfieldovy sítě pro TSP a ilustruje, jak lze neuronové sítě využít pro řešení optimalizačních problémů.

5. Implementace ostatních algoritmů

5.1. Genetický Algoritmus

Genetický algoritmus využívá principy evoluční biologie, jako jsou selekce, křížení a mutace, k optimalizaci řešení složitých problémů, jako je problém obchodního cestujícího (TSP). Algoritmus začíná s náhodně generovanou populací možných cest a postupně aplikuje fitness funkci, která hodnotí jednotlivé cesty na základě jejich celkové délky. Cesty s kratší délkou mají vyšší fitness hodnotu. Nejlepší jedinci jsou následně vybráni pro reprodukci, při které dochází ke kombinaci jejich genetických informací a následnému křížení. Pro zajištění genetické diverzity a přizpůsobení novým podmínkám je do procesu zahrnuta mutace, při které jsou náhodně zaměněny dvě města v cestě. Tento cyklus selekce, křížení a mutace se opakuje po určený počet generací nebo dokud není nalezeno uspokojivé řešení.

5.2. Algoritmus Mravenčí Kolonie

Algoritmus mravenčí kolonie napodobuje chování mravenců při hledání potravy a využívá feromonové stopy k vedení vyhledávání optimálních cest. Mravenci začínají na náhodných pozicích a průzkumem prostoru generují cesty mezi městy. Intenzita feromonů na cestě mezi dvěma městy zvyšuje pravděpodobnost, že další mravenci tuto cestu zvolí, přičemž feromony se postupně odpařují, čímž se snižuje jejich vliv pokud nejsou obnovovány. Po dokončení cesty mravenci aktualizují feromony na této cestě na základě celkové délky cesty; kratší cesty získají vyšší koncentraci feromonů, což zvyšuje šanci, že budou vybrány dalšími mravenci v následujících iteracích. Tento proces se opakuje v několika iteracích, přičemž algoritmus postupně zlepšuje nalezené řešení.

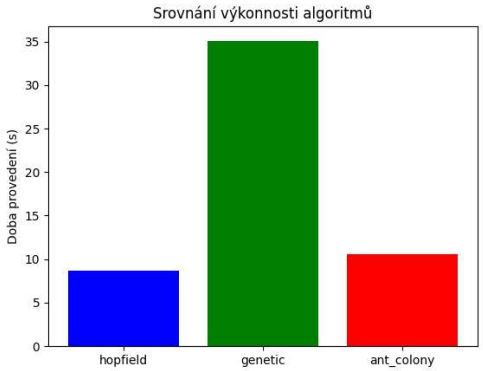
5.3. Souhrn

Oba algoritmy jsou implementovány tak, aby iterativně vylepšovaly řešení problému a byly schopny adaptovat se na dynamicky se měnící podmínky problému. Genetický algoritmus je vhodný pro rychlé prozkoumání velkého prostoru možností, zatímco algoritmus mravenčí kolonie je efektivní v postupném zlepšování řešení pomocí kooperativních strategií.

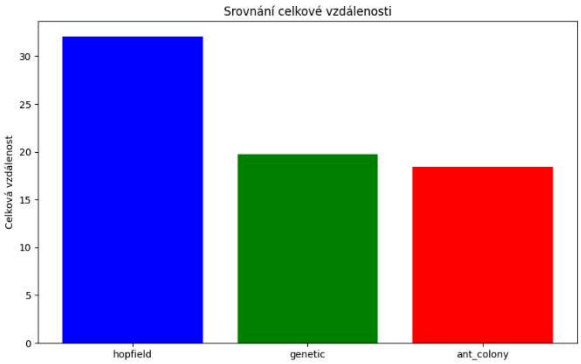
6. Experimenty a hodnocení

Tato kapitola se zaměřuje na experimentální vyhodnocení a srovnání tří algoritmů – Hopfieldovy sítě, genetického algoritmu a mravenčího algoritmu – pro řešení problému obchodního cestujícího pomocí skupiny hospod. V experimentech byly algoritmy hodnoceny podle dvou klíčových metrik: doby provedení a celkové vzdálenosti nalezené cesty.

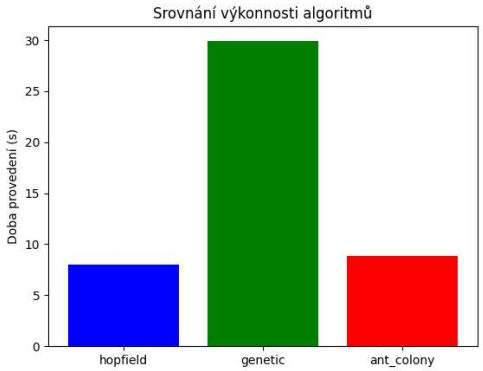
K experimentům byla využita data hospod, na kterých byly testovány algoritmy. Každý algoritmus byl spuštěn s předem definovanými parametry, jako je počet iterací, velikost populace u genetického algoritmu, počet mravenců a parametry feromonů u mravenčího algoritmu.



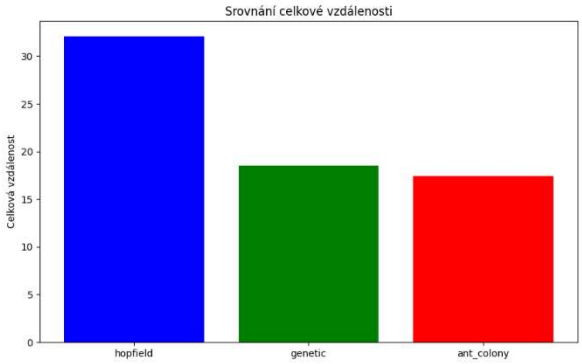
Obrázek 1 Graf srovnání rychlostí algoritmů pokus 1



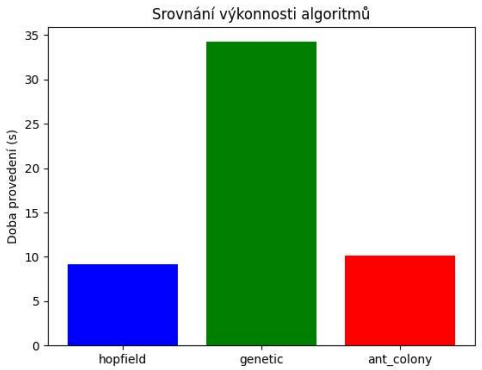
Obrázek 5 Graf srovnání kvality řešení (délka cesty) pokus 1



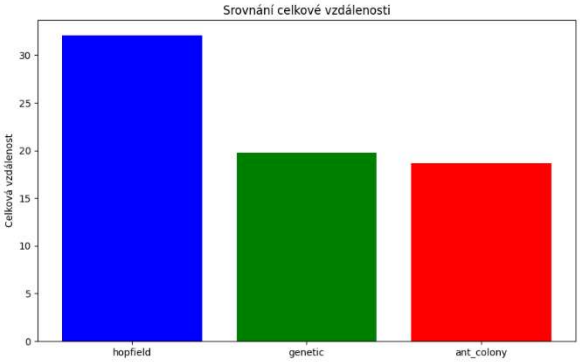
Obrázek 3 Graf srovnání rychlostí algoritmů pokus 2



Obrázek 2 Graf srovnání kvality řešení (délka cesty) pokus 2



Obrázek 4 Graf srovnání rychlostí algoritmů pokus 3



Obrázek 6 Graf srovnání kvality řešení (délka cesty) pokus 3

Výsledky experimentů jsou vizualizovány pomocí dvou grafů. Pro přesnost experimentu byl výpočet proveden třikrát. První graf ukazuje dobu provedení jednotlivých algoritmů a druhý graf srovnává celkové vzdálenosti, které algoritmy našly jako řešení problému obchodního cestujícího.

Z grafu je zřejmé, že genetický algoritmus má nejdelší dobu provedení, ale zároveň vrací lepší výsledky z hlediska kvality nalezených cest než Hopfieldova síť. Mravenčí algoritmus má dobu provedení podobnou Hopfieldově síti, ale nalezené cesty jsou výrazně kratší než u Hopfieldovy sítě. Hopfieldova síť je nejrychlejší, avšak vrací nejdelší nalezené cesty, což naznačuje její tendenci uvíznout v lokálních minimech.

Mravenčí algoritmus, který trvá podobně jako Hopfieldova síť, nachází nejkratší cesty, což z něj činí nejlepší volbu mezi rychlostí a kvalitou výsledků. Hopfieldova síť, přestože rychle generuje řešení, má tendenci uvíznout v lokálních minimech, což vede k delším nalezeným cestám.

Experimenty ukázaly, že výběr algoritmu by měl záviset na specifických požadavcích problému a dostupných zdrojích. Pro aplikace, kde je prioritou rychlost, by mohl být preferován mravenčí algoritmus, zatímco pro aplikace, kde může být důležitější průzkum prostoru řešení, by mohl být vhodnější genetický algoritmus. Výsledky také poukazují na potřebu dalšího výzkumu a optimalizace, zejména u Hopfieldovy sítě a genetického algoritmu, pro zlepšení jejich výkonnosti v reálných aplikacích. Pro pokus byly zvoleny následující hodnoty:

- Počet iterací: 1000
- Velikost populace: 1000
- Generace: 500
- Pravděpodobnost mutace: 0.1
- Počet mravenců: 50
- Alfa: 1
- Beta: 2
- Odpařování: 0.95
- Penalizace za aktivaci více neuronů ve stejném řádku: 500
- Penalizace za aktivaci více neuronů ve stejném sloupci: 500
- Penalizace za aktivaci více nebo méně neuronů než je měst: 200
- Penalizace za vzdálenost: 500
- Základní hodnota pro aktivaci neuronů, použitá v sigmoideální aktivační funkci: 0.02
- Krok pro gradient descent algoritmus, použitý k aktualizaci vah sítě: 1e-6

7. Diskuze

Porovnávání Hopfieldovy sítě, genetického algoritmu a mravenčího algoritmu z hlediska teorie a praxe odhaluje, že každý z nich má specifické silné a slabé stránky v závislosti na povaze problému, který řeší. Hopfieldova síť je teoreticky atraktivní pro její schopnost dynamicky aktualizovat stavy sítě s cílem najít energetická minima, ale v praxi čelí výzvám, jako je uvíznutí v lokálních minimech a špatná škálovatelnost pro větší problémy TSP. Genetický algoritmus je ve srovnání flexibilní a robustní, efektivně prozkoumává velký prostor řešení a adaptuje se na různé varianty problému, což jej činí vhodným pro praxi. Mravenčí algoritmus, při správné konfiguraci nacházel v krátké časové době nejlepší výsledky ze všech třech algoritmů. Genetický algoritmus sice trvá nejlépe ale při experimentech se stejným počtem iterací, má druhé nejlepší výsledky.

7.1. Možné problémy při implementaci a jejich řešení

Při implementaci těchto algoritmů může dojít k několika problémům, které vyžadují specifické řešení:

- Hopfieldova síť: Hlavní problém spočívá ve správném nastavení parametrů a překonání tendence uvíznout v lokálních minimech. Pro zlepšení výsledků se doporučuje použití technik simulovaného žíhání nebo hybridních přístupů. Simulované žíhání je optimalizační metoda inspirovaná procesem žíhání v metalurgii, která umožňuje přijímat i horší řešení během procesu optimalizace, aby se zabránilo uvíznutí v lokálních minimech a zvýšila se šance na nalezení globálního minima. Hybridní přístupy kombinují různé optimalizační techniky do jednoho rámce, využívají silné stránky každé metody a snaží se minimalizovat jejich slabiny, což zlepšuje efektivitu a účinnost v řešení složitých optimalizačních problémů.
- Genetický algoritmus: Potýká se s vysokou výpočetní náročností a rizikem předčasné konvergence. Efektivní selekční a mutační strategie jsou klíčové pro zvýšení diverzity populace a předejití uvíznutí v lokálních optimách.
- Mravenčí algoritmus: Čelí výzvám spojeným s časovou náročností a komplexností nastavení parametrů, jako jsou aktualizace feromonů a volba heuristiky. Efektivitu tohoto algoritmu je možné zvýšit dynamickým nastavováním parametrů podle předběžných výsledků a využitím paralelních výpočtů.

7.2. Doporučení pro konkrétní typy aplikací TSP

Volba algoritmu pro konkrétní aplikace TSP závisí na několika faktorech, jako je velikost problému, požadavky na rychlost řešení a flexibilita v reakci na změny v problému.

- Malé až střední instance problému obchodního cestujícího (TSP): Hopfieldova síť může být velmi efektivním řešením. Díky své schopnosti rychle najít energetická minima je vhodná pro situace s menším množstvím měst, kde je nižší riziko uvíznutí v lokálních minimech.
- Velké instance TSP: Lépe se hodí genetický algoritmus, který díky své flexibilitě a schopnosti efektivně prozkoumávat rozsáhlé prostory řešení dokáže adaptovat na různé problémové domény, což jej činí vhodným pro složitější a rozmanitější sady údajů.

- Dynamicky se měnící prostředí: Ideální je mravenčí algoritmus. Jeho adaptivní průzkum prostoru a schopnost aktualizovat feromony na základě nově získaných informací umožňují efektivní reakci na změny, což ho činí vynikajícím nástrojem pro aplikace vyžadující pružnou a dynamickou optimalizaci.

8. Závěr

V této práci jsme provedli podrobné srovnání a analýzu tří rozdílných algoritmů – Hopfieldovy sítě, genetického algoritmu a mravenčího algoritmu – aplikovaných na problém obchodního cestujícího (TSP). Každý z algoritmů byl posouzen z teoretického hlediska, hodnocen na základě experimentů a byly diskutovány potenciální aplikace a výzvy spojené s jejich implementací.

Výsledky experimentů ukázaly, že mravenčí algoritmus poskytuje nejvyváženější výsledky z hlediska rychlosti a efektivity, což jej činí vhodným kandidátem pro různorodé aplikace vyžadující rychlé a efektivní plánování cest. Na druhou stranu, Hopfieldova síť, i když je užitečná pro rychlé řešení menších problémů, se potýká s omezeními ve škálovatelnosti a efektivitě pro větší instance TSP. Genetický algoritmus je ideální, pokud potřebujete nalézt optimální řešení v relativně krátkém čase.

Tato práce také identifikovala a navrhla řešení pro typické problémy spojené s implementací každého algoritmu, což poskytuje cenné informace pro další výzkum a vývoj v oblasti optimalizačních algoritmů. Doporučení pro specifické typy aplikací TSP poskytují praktický návod pro výběr nejvhodnějšího algoritmu v závislosti na konkrétních požadavcích a podmínkách.

Závěrem, tato studie přináší hlubší porozumění o silných a slabých stránkách každého z prozkoumaných algoritmů, což umožňuje lepší informovanost při rozhodování o nejvhodnějších metodách pro řešení problémů obchodního cestujícího. Výzkum by měl pokračovat s cílem dále optimalizovat existující algoritmy a rozvíjet nové strategie pro zlepšení výkonu v reálných aplikacích.