Evolucijsko računanje

Vaje 2025/2026

2 Integracija algoritma v ogrodje EARS

V tej nalogi boste izbrali in implementirali izbrani evolucijski algoritem znotraj ogrodja EARS. Po implementaciji boste napisali poročilo. Zahteva se oddaja izvorne kode vašega algoritma, članka, izvorne kode izvirnega algoritma ter poročila.

Vaš izbrani algoritem ne sme biti starejši od 3. let (leto objave ne sme biti pred letom 2022) in mora biti primeren za zvezne enokriterijske optimizacijske probleme. Preverite, ali algoritem že obstaja v ogrodju EARS! Izbrani algoritem mora imeti izvorno kodo od avtorja. Preden začnete z implementacijo, moram odobriti vaš izbrani algoritem, zato zagotovite potrebne informacije na seznamu.

Pomoč pri izbiri algoritma:

- Preverite revije na temo evolucijskega računanja, kot so: Applied Soft Computing, IEEE Transactions on Evolutionary Computation, Swarm and Evolutionary Computation, Evolutionary Computation, Natural Computing, Expert Systems with Applications, Soft Computing ali uporabite Google Scholar.
- Ključne besede za iskanje ustreznega evolucijskega algoritma: unconstrained, evolutionary algorithm, single-objective optimization, metaheuristic . . .
- Repozitorij s seznamom evolucijskih algoritmov
- Članki:
  - A Contemporary Systematic Review on Meta-heuristic Optimization Algorithms with Their MATLAB and Python Code Reference
  - A review of nature-inspired algorithms on single-objective optimization problems from 2019 to 2023
  - Performance assessment and exhaustive listing of 500+ natureinspired metaheuristic algorithms

Delo z ogrodjem EARS:

- Prenesite zadnjo različico ogrodja EARS.
- Dodajte nov paket za vaš algoritem v direktorij: org.um.feri.ears.algorithms.so. Paket naj nosi kratico algoritma (npr. za Moth Search Algorithm: org.um.feri.ears.algorithms.so.msa).
- Za zgled si vzemite algoritem GWO, ki se nahaja v paketu org.um.feri.ears.algorithms.so.gwo.
- Za generiranje naključnih števil **obvezno** uporabite obstoječe metode, kot so RNG.nextInt(), RNG.nextDouble(), RNG.shuffle(), RNG.randomPermutation(), RNG.nextGaussian(), RNG.nextBoolean() itd.
- Če želite nastaviti fiksno začetno seme, uporabite ukaz RNG.setSeed(seed). Privzeti generator naključnih števil je Mersenne Twister.
- Za preverjanje veljavnosti ustvarjenih rešitev (ali je rešitev znotraj spodnje in zgornje meje) uporabite metodo task.problem.isFeasible(). Po potrebi popravite rešitev z metodo task.problem.setFeasible().
- Preden ovrednotite rešitev z metodo task.eval(solution), preverite, ali ste že porabili vsa ovrednotenja z metodo task.isStopCriterion(), da ne sprožite izjeme StopCriterionException. Preverjanje je potrebno na vsakem mestu, kjer ovrednotimo rešitve.
- Uporabite metodo task.isFirstBetter(solution1, solution2), da preverite, katera rešitev je boljša.
- Primer preprostega zagona algoritma lahko najdete v razredu SOSingleRun, ki se nahaja v paketu org.um.feri.ears.examples.

Implementacija in primerjava z izvorno kodo

Izbrani algoritem lahko implementirate v Javi ali Kotlinu. Algoritem naj ima privzeti konstruktor, v katerem so parametri nastavljeni s priporočenimi nastavitvami. V algoritem dodajte parameter isDebug. Če je parameter nastavljen na true, naj algoritem izpiše vsako rešitev, kadar pride do izboljšave. Na zagovoru boste morali dokazati, da dosežete **enake rezultate** kot algoritem z izvorno kodo. To lahko zagotovimo le, če vzporedno zaženemo izvorno kodo članka in lastno implementacijo. Tukaj se pojavi težava zaradi stohastičnosti evolucijskih algoritmov. Torej moramo v obeh algoritmih imeti isto zaporedje naključnih števil. To lahko dosežemo, tako da uporabljamo isti generator naključnih števil in isto začetno seme. Drugi način, kako lahko zagotovimo enaka naključna števila, je, da uporabimo vnaprej generiran seznam naključnih števil. Ogrodje EARS že ima vgrajen takšen generator v razredu PredefinedRandom, ki se nahaja v paketu org.um.feri.ears.util.random. V razredu lahko tudi najdete primer uporabe generatorja. Šele ko dobite popolnoma enako populacijo po nekaj generacijah, ste lahko prepričani, da sta implementaciji identični. Za pomoč pri primerjavi izvorne kode si lahko preberete članek Primerjava evolucijskih algoritmov implementiranih v različnih programskih jezikih.

Zagon algoritma na primerjalnem testu

Ko ste prepričani, da je vaša implementacija identična izvorni kodi, lahko zaženete algoritem na primerjalnem testu (ang. Benchmark). Za zagon primerjalnega testa si pomagajte z razredom SOBenchmarkExample, ki se nahaja v paketu org.um.feri.ears.examples. Algoritem vključite v benchmark tako, da pokličete algorithms.add(new ImeAlgoritma()). Najprej poženite privzeti benchmark RPUOed30Benchmark, ki vsebuje preproste probleme in se bo posledično izvedel v krajšem času. Če se benchmark uspešno izvede, se bo na koncu prikazal graf z Rating intervali. Nato zaženite zahtevnejši benchmark iz tekmovanja CEC 2015, tako da RPUOed30Benchmark zamenjate z CEC2015Benchmark. Zagon tega benchmarka lahko traja nekaj ur. Vaš izbrani algoritem mora premagati vsaj algoritem RandomSearch, v nasprotnem primeru je zelo velika verjetnost, da je implementacija algoritma napačna.

Poročilo

Pripravite poročilo implementacije in rezultatov v obliki Word dokumenta, ki mora vsebovati:

- Kratek povzetek delovanja algoritma.
- Opis pomanjkljivosti in razlik med izvorno kodo in člankom. Izvorne kode ne spreminjajte!
- Rezultate (Rating intervali) primerjalnega testa CEC2015Benchmark, v katerem je vključen izbrani algoritem, RandomSearch, ABC, PSO, GWO in JADE. Število neodvisnih zagonov nastavite na **10**!
- Rezultate članka in primerjalnega testa. Poskusite pripraviti kar se da podoben primerjalni test, tako da ustvarite lasten Benchmark, ki vsebuje seznam problemov, ki so bili uporabljeni v članku. V primerjavo vključite tiste algoritme, ki so omenjeni v članku in so implementirani v ogrodju EARS. Primerjate dobljene rezultate s tistimi v članku. V poročilu vključite rezultate in ugotovitve primerjave.

Točkovanje:

- **Implementacija algoritma (25 točk):** Popolnost in pravilnost implementacije v ogrodju EARS, uporaba pravilnih metod ogrodja (RNG, task.eval, zaustavitveni pogoj, itd.) (**15 točk**) ter identičnost z izvorno kodo – enaki rezultati pri fiksnem semenu (**10 točk**).
- **Rezultati benchmarka (8 točk):** Uspešen zagon primerjalnega testa CEC2015Benchmark (**5 točk**) in primerjava z algoritmi iz članka na podobnem testnem setu (**3 točke**).
- **Poročilo (7 točk):** Kratek povzetek delovanja algoritma, opis pomanjkljivosti in razlik med izvorno kodo in člankom ter predstavitev in interpretacija rezultatov.

---

* Naloga je **obvezna** in vredna **40 točk**.