EpiSolve: Programátorská dokumentace

Jan Beneš
MFF UK, obor Informatika
Programování 2

Obsah

1 Úvod a Architektura

Projekt EpiSolve je implementován v jazyce C# s využitím .NET platformy. Cílem je demonstrovat využití evolučního algoritmu (EA) k optimalizaci parametrů pro řízení epidemie v multiagentní simulaci.

Architektura je rozdělena do několika klíčových komponent:

- **Hlavní program (`Program.cs`):** Vstupní bod aplikace, zodpovědný za načtení konfigurace a spuštění evolučního algoritmu.
- **Evoluční algoritmus (`EvolutionaryAlgorithm.cs`):** Jádro optimalizačního procesu. Spravuje populaci, provádí selekci, křížení a mutaci. Pro ohodnocení jedinců volá simulační modul.
- **Simulační modul (`Simulation.cs`):** Spouští jednu kompletní simulaci epidemie s danou strategií a vrací výsledné statistiky.
- **Agent (`Agent.cs`):** Reprezentuje jednotlivce v simulaci s vlastním stavem (náchylný, nakažený, uzdravený, mrtvý), věkem a pozicí.
- **Fitness kalkulátor (`FitnessCalculator.cs`):** Statická třída, která na základě výsledků simulace a parametrů strategie vypočítá její fitness (kvalitu).
- **Datové struktury:** Třídy pro parametry (`EAParameters`, `SimulationParameters`), strategii (`MeasuresStrategy`), výsledky (`SimulationResult`) a pomocné struktury (`GridMap`, `GridPosition`).
- **Pomocné moduly:** Načítání konfigurace (`ConfigLoader.cs`) a vykreslování grafů (`GraphPlotter.cs`).

2 Struktura projektu a popis tříd

2.1 `Program.cs`

Obsahuje metodu `Main`, která orchestrací volání ostatních komponent spouští celý proces. Načte konfiguraci pomocí `ConfigLoader`, vytvoří instanci `EA` a zavolá její metodu `FindBestStrategy()`.

2.2 `EvolutionaryAlgorithm.cs`

Třída `EA` implementuje logiku evolučního algoritmu.

Vlastnosti: Uchovává parametry EA a simulace, aktuální populaci (`_population`), statistiky pro grafy a instanci generátoru náhodných čísel.

Klíčové metody:

- `FindBestStrategy()`: Hlavní smyčka evoluce. Inicializuje populaci a v cyklu provádí ohodnocení, selekci a tvorbu nové generace.
- `InitializePopulation()`: Vytvoří počáteční populaci jedinců s náhodnými strategiemi.
- `EvaluatePopulation()`: Pro každého jedince v populaci spustí simulaci (a to `NumberOfRunsForAveraging`-krát pro zprůměrování) a pomocí `FitnessCalculator` mu přiřadí fitness skóre. Výpočet probíhá paralelně pomocí `Parallel.ForEach`.

- `SelectParentTournament()`: Implementuje turnajovou selekci. Náhodně vybere `TournamentSize` jedinců a vrátí toho s nejlepším (nejnižším) fitness.
- `Crossover()`: Z dvou rodičovských strategií vytvoří potomka prostým zprůměrováním jejich odpovídajících parametrů.
- `Mutate()`: S pravděpodobností `MutationRate` aplikuje na strategii jedince mutaci. Mutace mírně změní hodnotu jednoho nebo více parametrů strategie. Zajišťuje, aby hodnoty zůstaly v platných mezích (např. `LockdownEndThreshold` nemůže být vyšší než `LockdownStartThreshold`).

2.3 `Simulation.cs`

Statická třída pro běh jedné simulace.

Klíčové metody:

- `Simulate()`: Hlavní metoda, která přijímá strategii a parametry. Vytvoří agenty, a poté v cyklu (přes časové kroky) provádí:
  1. Pohyb agentů (`agent.Move()`).
  2. Pokus o zotavení, ztrátu imunity a úmrtí (`agent.Try...()`).
  3. Pokus o infekci pro náchylné agenty (`agent.TryInfect()`).
  4. Vyhodnocení a případná změna stavu lockdownu na základě aktuálního počtu nakažených a prahů ve strategii.
  5. Sběr statistik (max. nakažených, délka lockdownu).

  Po skončení smyčky vrací objekt `SimulationResult` se sebranými daty.
- `InitAgents()`: Vytvoří pole agentů s náhodnými pozicemi, věkem a s jedním počátečním nakaženým.

2.4 `Agent.cs`

Reprezentuje osobu v simulaci.

Vlastnosti: `Status` (SIR), `Age`, `Position`, časové značky pro infekci/uzdravení.

Klíčové metody:

- `Move()`: Změní pozici agenta na náhodnou v okolí. V případě lockdownu je rozsah pohybu a pravděpodobnost pohybu omezena.
- `TryInfect()`: Klíčová metoda pro šíření nákazy. Pro náchylného agenta spočítá počet nakažených sousedů na stejném a na sousedních polích. Na základě těchto počtů, parametrů simulace a stavu lockdownu vypočítá výslednou pravděpodobnost nákazy a "hodí si kostkou".
- `TryRecover()`, `TryLoseImunity()`, `TryDie()`: Metody, které na základě času a pravděpodobností mění stav agenta.

2.5 `FitnessCalculator.cs`

Statická třída s jedinou metodou `GetFitness()`.

Logika výpočtu:

1. Normalizuje metriky z `SimulationResult` (např. počet mrtvých / celkový počet agentů), aby byly v rozsahu cca 0-1.
2. Vypočítá penalizaci za "extrémní"hodnoty ve strategii (např. příliš restriktivní lockdown, nebo příliš pozdní reakce).
3. Sečte normalizované metriky vynásobené jejich vahami (z `SimulationParameters`) a přičte penalizaci.

Cílem je minimalizovat tuto hodnotu.

2.6 `MeasuresStrategy.cs`

Jednoduchá datová třída (chromozom), která drží 4 klíčové parametry pro řízení lockdownu, které evoluční algoritmus optimalizuje.

3 Závislosti

Projekt využívá následující externí knihovny:

- **ScottPlot:** Pro vykreslování grafu vývoje fitness. Je integrována přes NuGet balíček.
- **NUnit & NUnit3TestAdapter:** Pro jednotkové testy. Testy jsou v souborech končících na `Tests.cs`.

4 Testování

Projekt obsahuje sadu jednotkových testů pro ověření správné funkčnosti klíčových komponent.

- `AgentTests.cs`: Testuje základní chování agenta, např. pohyb na hraně mřížky.
- `FitnessCalculatorTests.cs`: Ověřuje správnost výpočtu fitness pro různé scénáře, včetně těch s penalizacemi.
- `GridMapTests.cs` a `GridPositionTests.cs`: Testují logiku mřížky a pozic.

Testy lze spustit pomocí Test Exploreru ve Visual Studiu nebo přes příkazovou řádku příkazem `dotnet test`.

5 Možnosti rozšíření

- **GUI:** Nahrazení konzolové vizualizace plnohodnotným grafickým rozhraním (např. pomocí AvaloniaUI nebo MAUI).
- **Složitější chování agentů:** Implementace sociálních skupin, dojíždění do práce, různé modely pohybu.
- **Rozšíření strategie:** Přidání dalších opatření do strategie (např. nošení roušek, vakcinace) jako dalších genů pro optimalizaci.
- **Jiné EA operátory:** Implementace a porovnání jiných typů křížení (např. jednobodové) nebo selekce.