Projektová dokumentace BIN

Efektivní evoluční návrh celulárního automatu

Autor:
Tomáš Beránek (xberan46)

E-mail: **xberan46@stud.fit.vutbr.cz**
Datum vytvoření: **8. května 2022**

1 Zadání

Cílem tohoto projektu je návrh a implementace **efektivního** evolučního algoritmu pro hledání počáteční konfigurace 2D celulárního automatu. Celulární automat se řídí pravidly Game of Life[1]. Hledaná počáteční konfigurace musí vést v 1 až N krocích do požadované, předem dané, konfigurace.

2 Evoluční algoritmus

Implementovaný EA se řídí běžným schématem:

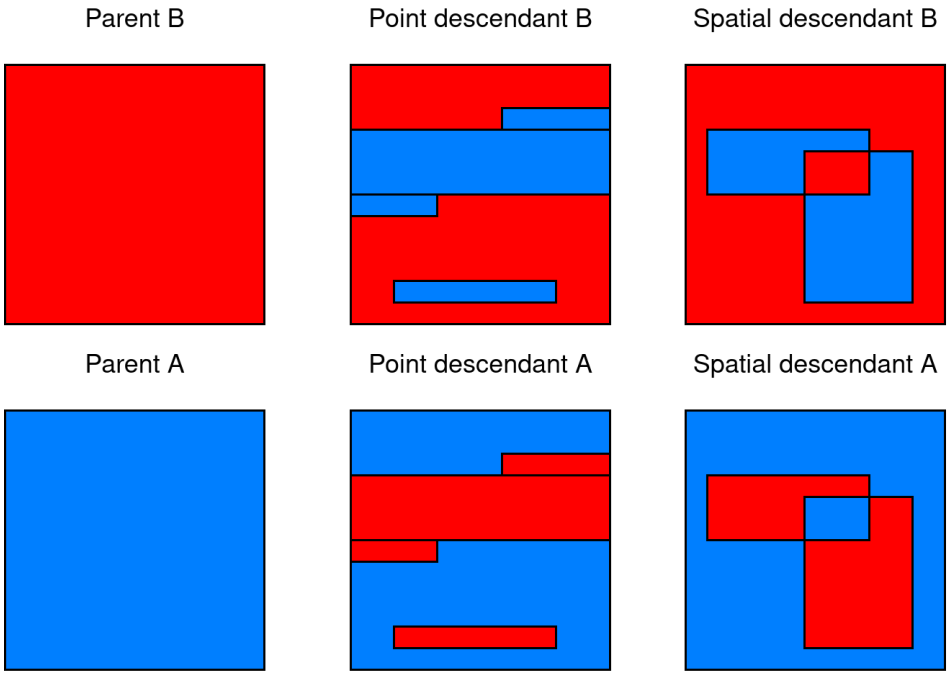
1) náhodná inicializace jedinců
2) ohodnocení jedinců
3) vybrání rodičů
4) křížení rodičů pro získání nových jedinců
5) mutace jedinců a skok na 2)

Každý **jedinec** je instance třídy `GameOfLife`. **Ohodnocení** každého jedince probíhá zavoláním jeho metody `makeStep`, která jako argument bere počet kroků, které má CA udělat a vrací **fitness** – počet rozdílných buněk mezi hledanou konfigurací a konfigurací jí nejvíce podobnou v průběhu dělání kroků. Fitness je tudíž chyba, kterou se snažíme **minimalizovat**.

Výběr rodičů probíhá prahováním – vybere se pouze N nejlepších jedinců, kteří se stanou rodiči další populace. Počet rodičů je nastavitelným parametrem simulace, viz. kapitola 6. Bylo experimentováno i s výběrem rodičů podle jejich relativních pořadí, aby se předešlo degradaci populace, ovšem tento způsob naopak ještě zhoršil výsledky experimentů. Nejlepší jedinec každé generace je automaticky a beze změny vybrán do další generace (**elitismus**). Je využito **generační EA** – nová generace je složena pouze z vytvořených potomků (a nejlepšího jedince).

Každý potomek je vytvořen **křížením** ze dvou náhodně vybraných rodičů. Výstupem každého křížení jsou, kvůli optimalizaci, dva potomci. Na každého takto vygenerovaného potomka je poté aplikována **mutace**. Veškeré parametry jsou opět nastavitelné.

[1] Conway's Game of Life – https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life.



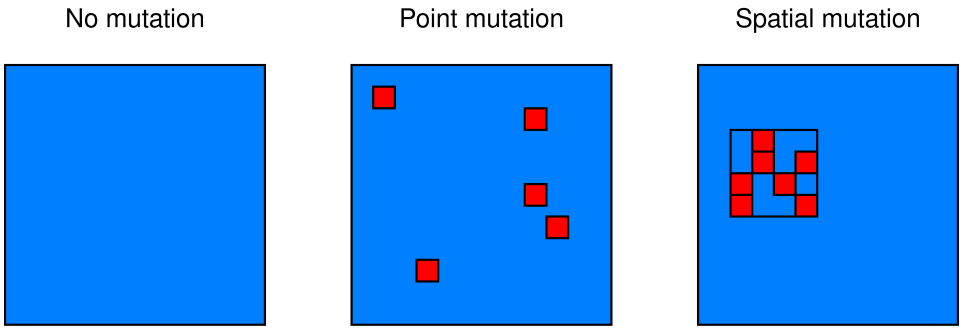
Obrázek 1: Porovnání bodového a prostorového křížení.

2.1 Prostorové genetické operátory

Bodové křížení je primárně určeno pro křížení vektorů. A i když konfigurace 2D CA je možné reprezentovat jako jediný vektor (takto je také uložen v paměti), tak není vhodné na tuto 2D mapu takto pohlížet. Hlavně z toho důvodu, že při bodovém křížení není možné vybrat např. pouze levou část mapy. Vždy se při bodovém křížení vybere buď pouze část řádku nebo více celých řádků, viz. obrázek 1.

Z tohoto důvodu bylo navrhnuto a implementováno **prostorové genetické křížení**, které respektuje 2D povahu Game of Life. Jedná o jednoduché vygenerování náhodného obdélníku, který reprezentuje oblast ve které se vymění informace od obou rodičů. Počet obdélníků je opět nastavitelným parametrem. Pokud se překrývá více obdélníků je jejich efekt podobný negaci, viz. obrázek 1.

Vzhledem k povaze pravidel Game of Life není **bodová mutace** moc úspěšná, protože např. osamocená živá buňka ihned v dalším kroku zemře. Při experimentování jsem narazil na problém vymírání buněk – EA poměrně rychle zkonvergoval do stavu, kdy byla naprostá většina buněk většiny konfigurací mrtvá a aby bodové mutace dokázaly tyto konfigurace oživit, bylo by nutné jich naakumulovat velké množství a protože jsou tyto mutace rozprostřené, tak při velkém množství už by se EA blížil náhodnému procházení stavového prostoru.



Obrázek 2: Porovnání bodové a prostorové mutace.

Z tohoto důvodu byla vytvořena **prostorově koncentrovaná mutace**, která vybere náhodně oblast ve které se s velkou pravděpodobností (např. 50 %, viz. kapitola 4) znegují stavy buněk. Toto umožní obživnutí určité oblasti mapy. Velikost oblasti byla nastavena na čtverec o ploše 1 % plochy mapy. Ukázka prostorové mutace je na obrázku 2.

Statistické vyhodnocení bodového a prostorového přístupu je uvedeno v kapitole 4.

3 Optimalizace

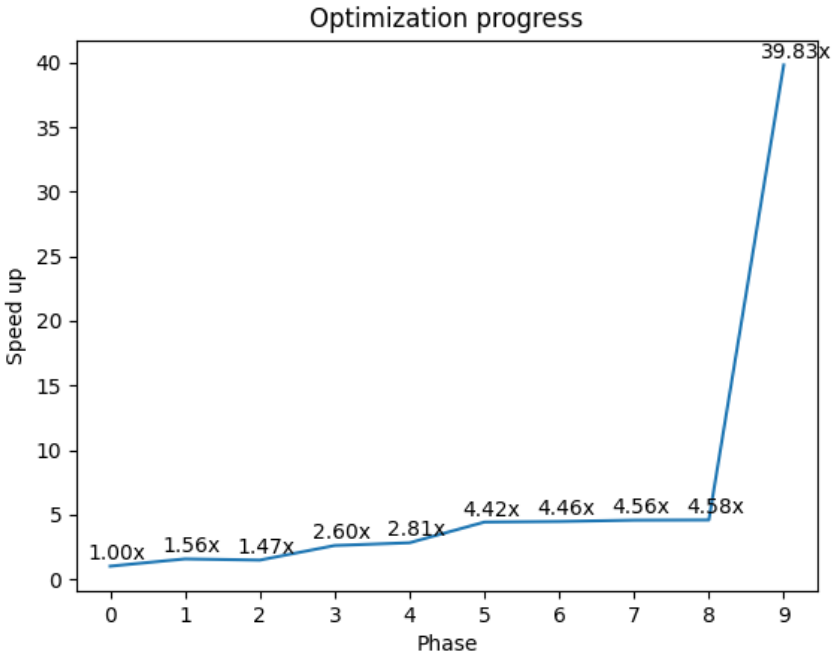
Jedním z hlavním cílů tohoto projektu byla optimalizace EA. Pro měření rychlosti EA byla zvolena metrika **počet updatů buněk / sekundu**. Počet updatnutých buněk lze vypočítat pomocí vzorce:

$$N_{updates} = G * P * M^2 * S$$

kde G je počet generací, P je velikost populace, M strana čtvercové mapy a S je počet kroků provedených z počáteční konfigurace.

Tato metrika byla zvolena tak, aby byla nebyla ovlivněna např. změnou velikosti mapy (kvůli dobrému zarovnání do cache), atp.

Pro měření času byla použita utilita `time`. Za hlavní čas byl považován čas strávený v user space. Po implementaci paralelizace se však muselo přejít na real time. Každý čas byl vypočítám jako průměr ze tří spuštění.



Obrázek 3: Graf zrychlení jednotlivých úprav.

Optimalizace probíhala v několika fázích, viz. obrázek 3. Jednotlivé fáze obsahovaly změny:

0) originální řešení – už zde byla samozřejmě snaha o psaní rozumně rychlého kódu (243 299 u/s)
1) zarovnání mapy na 64x64 – lepší zarovnání v paměti a lepší využití cache (380 090 u/s)
2) zpomalení díky přidání prostorových operací (358 114 u/s)
3) optimalizace pomocí `g++ -O1` (632 769 u/s)
4) optimalizace pomocí `g++ -O2` (684 318 u/s)
5) optimalizace pomocí `g++ -O3` (1 074 342 u/s)
6) využití rychlejších, ale nepřesných matematických operací (1085623 u/s)
7) optimalizace algoritmu – odebrání zbytečných kopírování polí (1 110 159 u/s)
8) přidání `const` pro lepší informaci pro `g++` optimalizátor, inlinování funkcí (1 114 071 u/s)
9) přidání paralelizace pomocí 12 vláken (9 690 340 u/s)

4 Statistické vyhodnocení

Hlavním cílem statistického vyhodnocení bylo **porovnání bodových a prostorových operací**. Byla provedena sada experimentů s různým nastavením křížení a mutace. Cílem by zjistit, které operace dokáží hledat nejpřesnější řešení. Výsledky experimentů s bodovými operacemi jsou na obrázku 4, prostorovými operacemi pro koncentraci mutace 30 % na obrázku 5, 50 % na obrázku 6 a 70 % na obrázku 7. Každý bod v grafu udává průměrnou minimální (nejlepší) fitness z 15 běhů na 3 náhodně vygenerovaných hledaných konfiguracích. Hledané konfigurace mají existující řešení, tudíž minimální fitness je 0 a maximální fitness je 4096 (dáno velikostí mapy). Každý běh měl 36 jedinců, 10 rodičů, 500 generací a aplikoval se elitismus.

Z experimentů s **prostorovými operacemi** lze vidět, že výsledky pro různé procento mutace jsou velmi podobné, nicméně nejlepších výsledků dosahuje mutace 30 %, viz. obrázek 5. Konkrétně fitness 825 pro nastavení – křížení se dvěma obdélníky a koncentrovaná mutace jedné oblasti.

Z experimentů s klasickými **bodovými operacemi** lze vidět, že nejlepší výsledky jsou pro nízké procento mutací, viz. obrázek 4. Také je vidět, že příliš nezáleží na počtu bodů křížení. V průměru dosahují bodové operace lepších výsledků než prostorové, ovšem zdaleka nejlepší fitness je pro prostorové operace – 825 oproti nejlepší fitness bodových operací – 832.

Nejlepší nalezené nastavení je dále použito pro experimenty s **fixním počtem vyhodnocených fitness** a měnícím se poměrem – počet generací/velikost populace. Cílem tohoto experimentu je zjistit, zda např. velký počet jedinců a menší počet generací není výhodnější pro tuto úlohu než nízký počet jedinců a mnoho generací.

Výsledky experimentu jsou na obrázku 8. Každý bod je průměrem nejlepších fitness z 15 běhů na 3 náhodných konfiguracích. Z grafu lze vyčíst, že pro tuto úlohu je rozhodně **lepší využívat generace s větším počtem jedinců** (jednotky stovek).

5 Programovací jazyk a knihovny

Samotná simulace je napsána v jazyku C++, zejména z důvodu jednoduché paralelizace a velmi dobrého optimalizátoru v `g++`. Pomocné skripty pro měření, statistické vyhodnocení a vytvoření grafů jsou napsány pro Python3.

Implementace EA byla inspirována kódem prezentovaným na BIN cvičení[2]. Bylo využíváno standardních knihoven a knihoven `jsoncpp` a `pthread` pro C++. Pro Python3 pouze standardních modulů.

[2] https://github.com/mrazekv/bin-lab-ca/

6 Návod na spuštění

Pro pohodlnější změnu parametrů simulace jsou všechna nastavení uložena v souboru `config.json` v kořenové složce adresáře. Simulaci je tak možné vyhodnocovat pro různá nastavení bez nutnosti opětovné kompilace. `config.json` obsahuje:

- **TargetMap** – soubor s požadovanou konfigurací (jeho formát si lze prohlédnout ve složce `patterns/`)
- **OutputMap** – výstupní soubor s nejlepším nalezeným řešením
- **MapSize** – velikost strany čtvercové mapy (cyklické okraje)
- **StepsGOL** – počet kroků pro každé vyhodnocení fitness
- **Population** – velikost populace (optimálního výkonu se dosáhne, pokud je velikosti populace dělitelná 12)
- **Elitism** – informace, zda používat elitismus
- **ParentPopulation** – počet vybíraných rodičů
- **MaxMutationPercetange** –

  0 – žádná mutace

  > 0 – maximální pravděpodobnost na bodovou mutaci

  < 0 – maximální počet oblastí pro prostorově koncentrovanou mutaci
- **Generations** – počet generací
- **Crossover** –

  0 – uniformní křížení

  > 0 – počet obdélníků pro prostorové křížení

  < 0 – počet bodů pro bodové křížení
- **ConcentratedMutation** – pravděpodobnost na mutaci buňky v oblasti prostorové mutace

Pro automatizaci jednotlivých kroků řešení byl napsán `Makefile` s cíly:

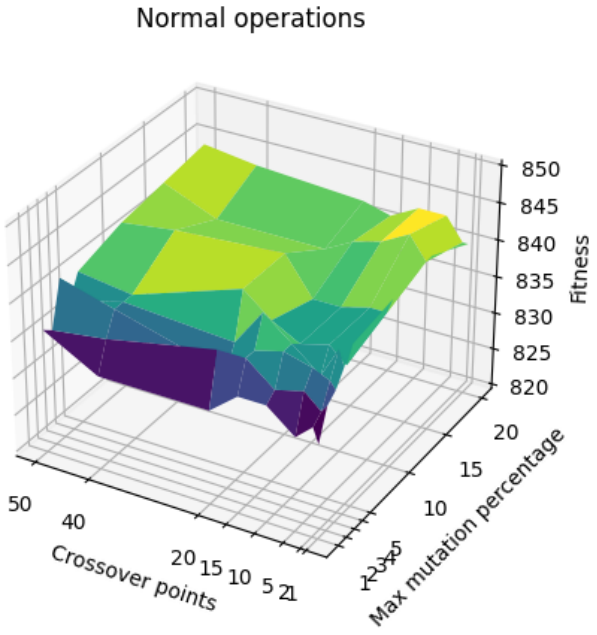
- **run/all** – spustí vytvořenou binárku simulace s nastavením v `config.json`
- **compile/install** – vytvoří binárku simulace
- **clean** – odstranění dočasných souborů – včetně defaultního souboru s nejlepším řešením!

- **test** – vypočítá metriku updates/s pro nově přeloženou simulaci
- **statistics** – znovu vyhodnotí veškeré experimenty – výstupem jsou soubory:
  - `statistics.csv` – výsledky experimentů pro porovnání bodových a prostorových operací
  - `statistics2.csv` – výsledky experimentů pro určení vhodné velikosti populace
- **graph** – vytvoření grafů ze souborů `statistics.csv` a `statistics2.csv` a jejich uložení do složky `graphs/`

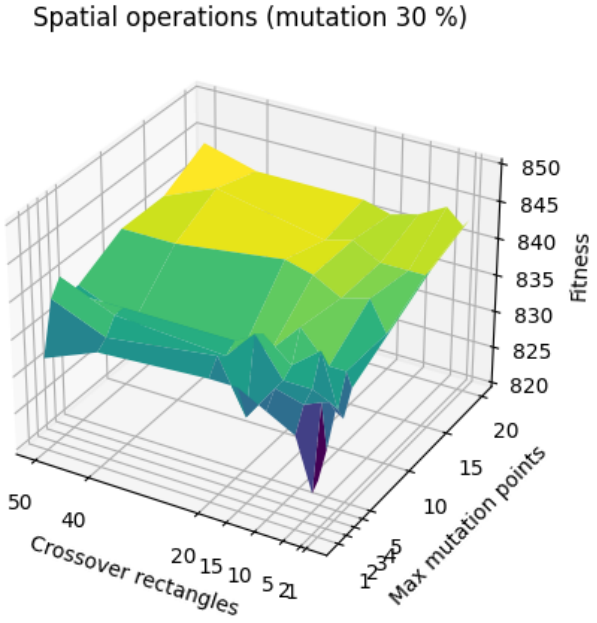
7 Závěr

Cílem tohoto projektu bylo navržení a implementace **efektivního** EA pro hledání počáteční konfigurace Game of Life. Vylepšení efektivity bylo rozděleno do tří kroků – zrychlení kódu, vylepšení genetických operací a vyladění hyperparametrů EA. Kód byl zrychlen téměř **40x** oproti baseline řešení. Pomocí statistického vyhodnocení bylo ukázáno, že nejlepších výsledků dosahují navržené a implementované **prostorové operátory**. Nejlepší nastavení bylo poté použito pro nalezení vhodné velikosti populace pro tuto úlohu. Ukázalo se, že na náhodně vygenerovaných hledaných konfiguracích je **nejlepší používat křížení pomocí dvou obdélníků s jedním místem koncentrované mutace** a **používat vysoké počty** (v řádu stovek) **jedinců** v každé populaci.

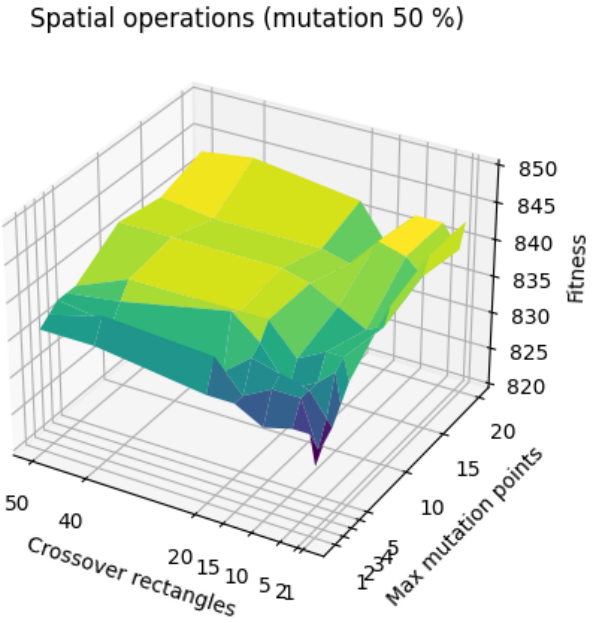
Navržené řešení ovšem není ani zdaleka na vrcholu efektivity. Určitě je možné dále optimalizovat kód i hyperparametry simulace. Pro další optimalizaci kódu by bylo např. možné využít konvoluce pro rychlé počítání živých/mrtvých buněk v okolí a využití GPU pro rychlé provedení konvoluce. Tato úprava by měla velmi velký vliv na zrychlení, protože právě děláním kroků je stráveno 98 % času simulace (podle nástroje `valgrind` a `kcachegrind`). Pro nalezení vhodnějších hyperparametrů by bylo nutné provést více experimentů, ty jsou ovšem časově velmi náročné – experimenty v této dokumentaci zabraly přibližně 35 hodin na procesoru Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz s 6 jádry a hypertrhreadingem.



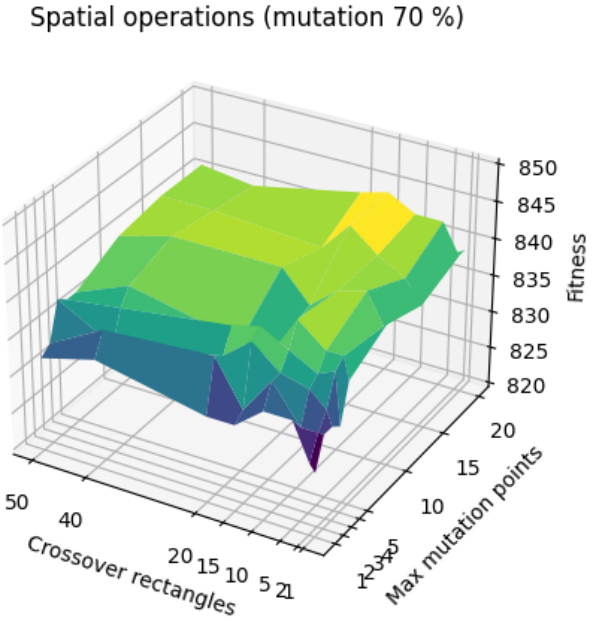
Obrázek 4: Výsledky experimentů pro bodové operace.



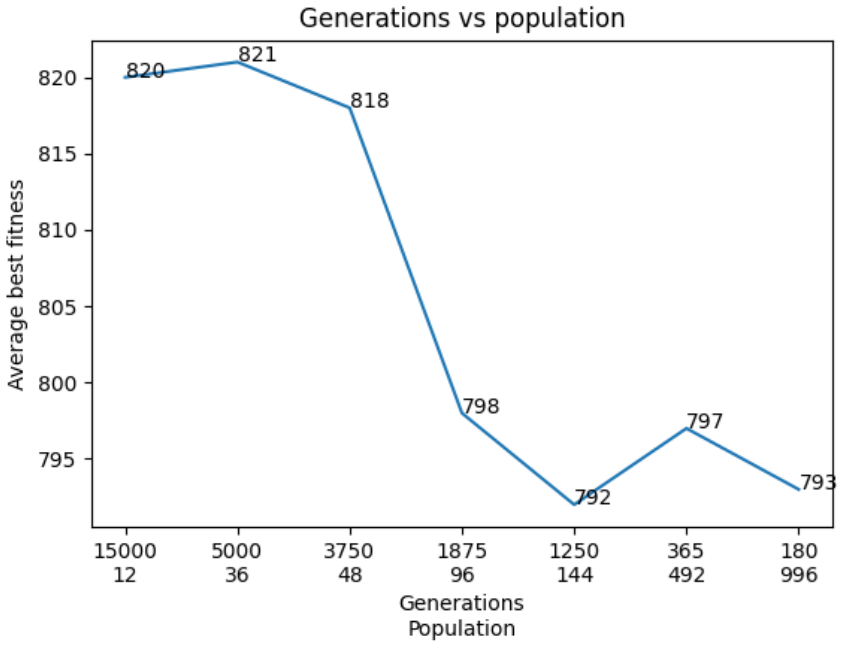
Obrázek 5: Výsledky experimentů pro prostorové operace s pravděpodobností na mutaci 30 %.



Obrázek 6: Výsledky experimentů pro prostorové operace s pravděpodobností na mutaci 50 %.



Obrázek 7: Výsledky experimentů pro prostorové operace s pravděpodobností na mutaci 70 %.



Obrázek 8: Výsledky experimentů pro fixní počet vyhodnocení fitness funkce (180 000 vyhodnocení) a proměnlivý počet generací/velikosti populace.