NI-KOP, 4. úkol

Jan Bittner (bittnja3)

Zadání úlohy

Úkolem je prozkoumat jednu z pokročilých iterativních metod na řešení problému batohu. Vybrána byla metoda genetických algoritmů. Cílem je seznámit se s algoritmem a citlivostí parametrů.

Použité prostředky

Programovací jazyky a software

Úloha byla řešena v jazyce C++ na operačním systému Windows 10.

Měření bylo spuštěno z **Bashe** prostřednictvím prostředí WSL 2 (**Windows Subsystem for Linux 2**), které využívá **Ubuntu 20.04.1 LTS**, nebylo tedy pro spuštění použito IDE.

Na zachycení aktuálního času bylo využito `std::chrono::high_resolution_clock::now()`.

Program je možné zkompilovat pomocí `make compile` a následně spustit s parametry souborů, např. `./program.out data/x.dat`.

Konfigurace testovacího stroje

Testování bylo provedeno na **Legion 5 15ARH05H**. Stroj obsahuje CPU **AMD Ryzen 7 4800H @ 2.90 GHz** a RAM **DDR4 16.00 GB**.

Algoritmus

Genetický algoritmus funguje na principu vylepšování populace v jednotlivých generacích. Chromosomy jsou pro problém batohu jednotlivé konfigurace, tedy bitový vektor pro přítomnost předmětu v batohu, a geny značí jednotlivou přítomnost věcí v konfiguraci batohu.

Pro prvotní generaci, která obsahuje náhodné věci s hodnotou větší než je průměr, s pravděpodobností přidání 40 %. Dále je přidána triviální kombinace "všechno jsou nuly" a "všechno jsou jedničky".

Pro každou generaci je pravděpodobnost pro křížení chromozomu a pro jeho mutaci, což do populace přidá další chromozomy. Následně jsou uchovány jen ty nejslibnější chromozomy (ty s větší cenou věcí v batohu, tj. fitness), zbytek je zahozen.

Po vytvoření X generací je řešením první jedinec seřazeného pole chromozomů dle fitness.

Rámcový popis postupu řešení

Algoritmus je implementován dle popisu předchozí kapitoly. Implementovány jsou objekty Chromosome, který obsahuje bitový vektor genů a metody pro křížení a mutaci chromozomu. Zbylý kód, tj. chování tvorby a selekce generací, jsou umístěny v `GaSolver.hpp`.

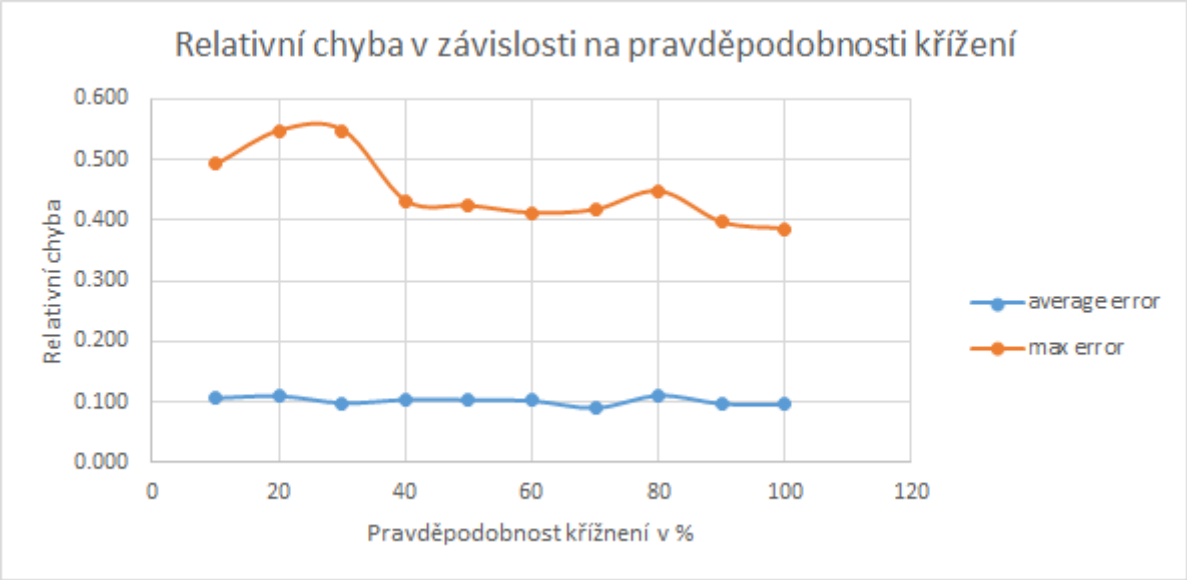
Naměřené výsledky

Naměřené výsledky jsou výsledky instancí ze souboru `data/x.dat`, který obsahuje 50 instancí o 40 věcech. Relativní chyba je porovnávána oproti řešení dynamickým programováním z předchozích úkolů. Zaměřeno bylo na prozkoumání závislosti na pravděpodobnost křížení, pravděpodobnost mutace a počtu generací. U všeho byla sledována relativní chyba i časová náročnost.

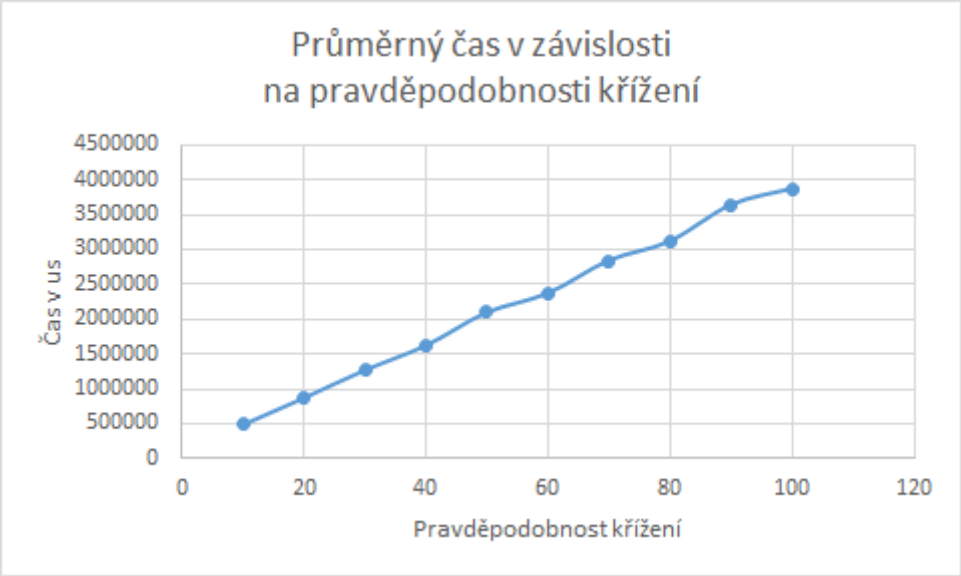
Závislost na pravděpodobnosti křížení

Pro toto měření byla nastavena velikost populace na 200, maximální počet generací na 100 a pravděpodobnost mutace na 10 %.

Z grafu níže je vidět, že průměrná chyba se udržuje po celou dobu okolo hodnoty 10 %, zatímco maximální chyba se snižuje z 55 % u 20% křížení na 40 % u 100% křížení.



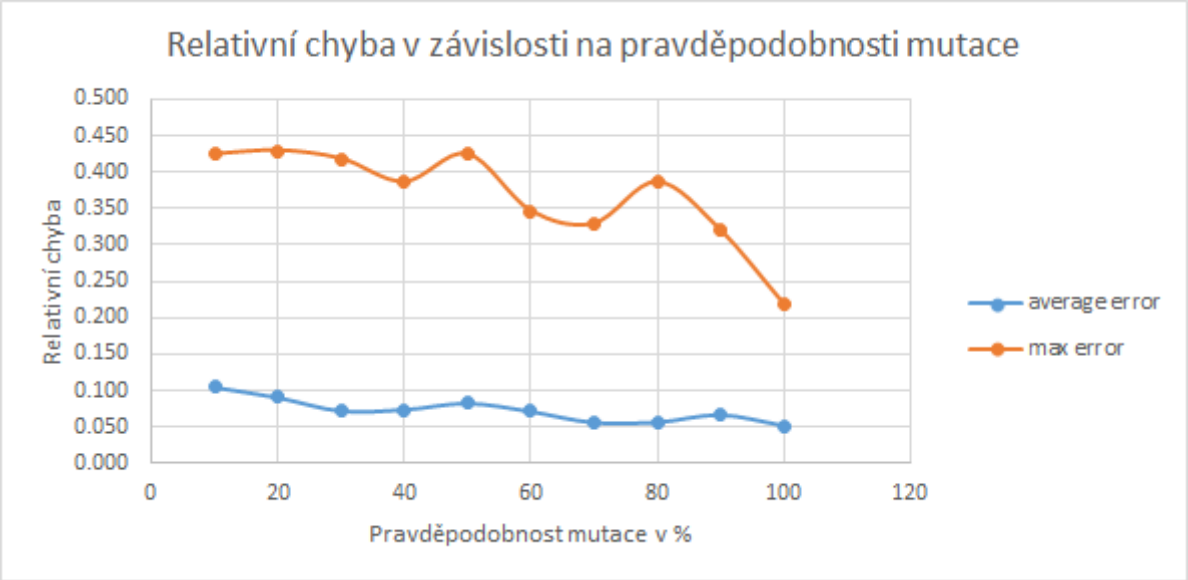
Z grafu níže je vidět, že čím je větší pravděpodobnost křížení, tím je výpočet náročnější, což je zřejmé z implementace, kdy se vytváří více chromozomů.



Závislost na pravděpodobnosti mutace

Pro toto měření byla nastavena velikost populace na 200, maximální počet generací na 100 a pravděpodobnost křížení na 25 %.

Z grafu níže je vidět, že průměrná chyba u 10% mutace je cca 10 % a snižuje se postupně až na průměrnou chybu 5% u 100 % mutace. Mutace dle algoritmu pomáhá lepšímu prozkoumání různých řešení. Jde vidět, že podobně se snižuje i maximální chyba z cca 45 % u 10% mutace na cca 22% chybu u 100 % mutace.

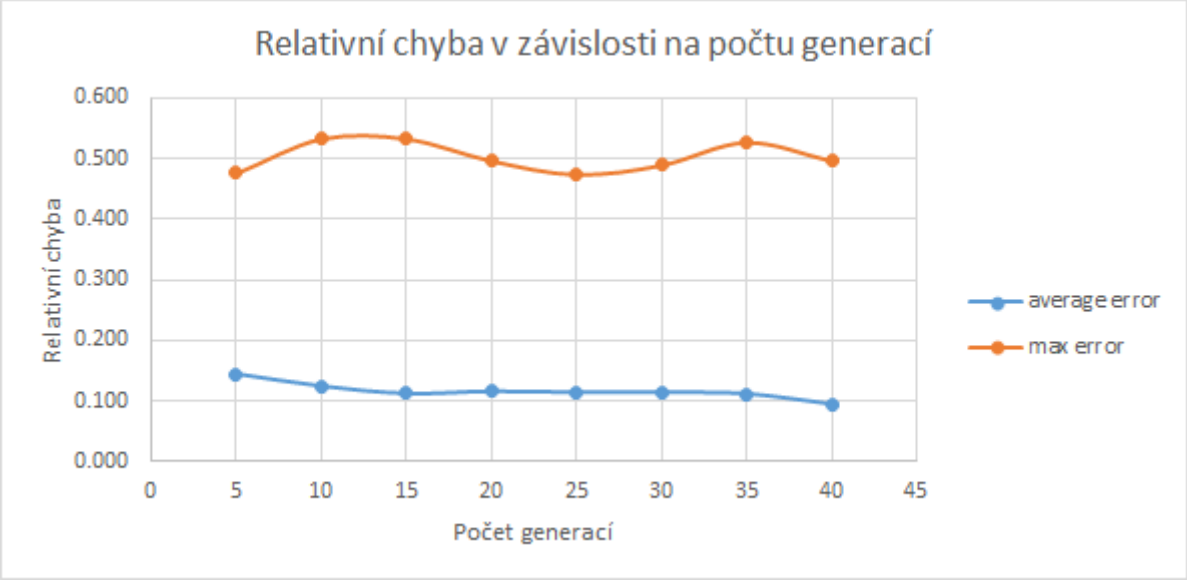


Z grafu níže je vidět, že algoritmus je nejméně časově náročný u 40% mutace, avšak rozdíly jsou velmi malé.

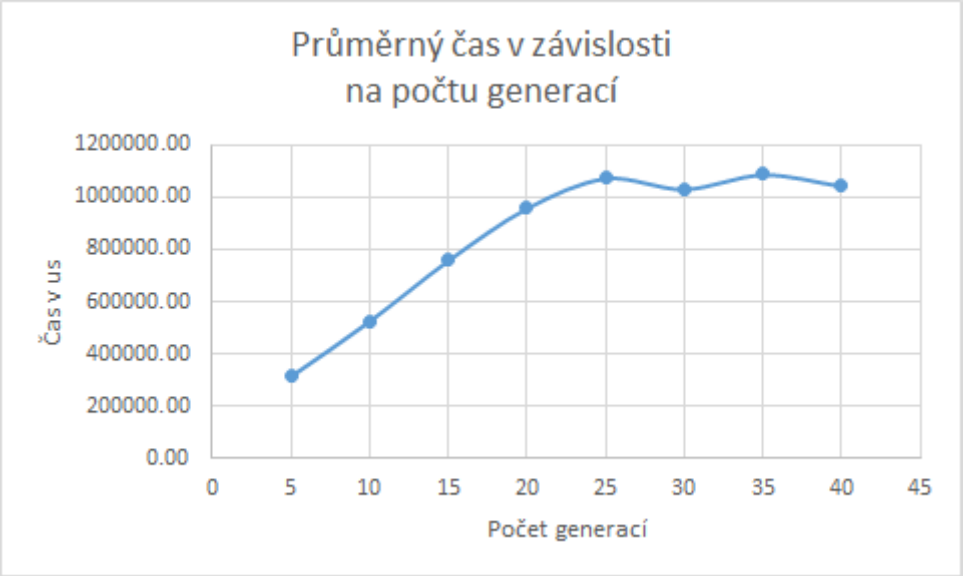
Závislost na počtu generací

Pro toto měření byla nastavena velikost populace na 200,pravděpodobnost křížení na 25 % a pravděpodobnost mutace na 10 %.

Z grafu níže je vidět, že relativní chyba se snižuje z cca 15 % u 5 generací na 10 % u 40 generací. Relativní chyba je naopak nejlepší okolo 25 generací.



Z grafu níže je vidět, že nejméně časově náročný je algoritmus u 5 generací a postupně se zvyšuje náročnost až do 25 generací, kde od tohoto bodu se pro měřené instance časová náročnost nezvyšuje.



Závěr

Byl implementován algoritmus genetického programování pro problém batohu a byly prozkoumány závislosti výsledků algoritmu na změně parametrů počtu generací a pravděpodobnosti křížení a mutace.

Algoritmus je pomalejší než algoritmus dynamického programování, oproti kterému byla testována relativní chyba. Tento algoritmus je ale univerzálnější a snadnější na implementaci.