5. Řešení problému vážené splnitelnosti booleovské formule pokročilou iterativní metodou

Ladislav Martínek

1 Problém

Je dána booleovská formule F proměnnných $X = (x_1, x_2, \ldots, x_n)$ v konjunktivní normální formě (tj. součin součtů). Dále jsou dány celočíselné kladné váhy $W = (w_1, w_2, \ldots, w_n)$. Najděte ohodnocení $Y = (y_1, y_2, \ldots, y_n)$ proměnných $x_1, x_2, \ldots, x_n$ tak, aby $F(Y) = 1$ a součet vah proměnných, které jsou ohodnoceny jedničkou, byl maximální.

Je přípustné se omezit na formule, v nichž má každá klauzule právě 3 literály (problém 3 SAT). Takto omezený problém je stejně těžký, ale možná se lépe programuje a lépe se posuzuje obtížnost instance (viz Selmanova prezentace v odkazech).

2 Zadání úlohy

Problém řešte některou z pokročilých lokálních heuristik (simulované ochlazování, genetické algoritmy, tabu prohledávání). Řešení jinými metodami prosím zkonzultovat se cvičícím nebo přednášejícím. Volby konkrétních parametrů heuristiky a jejích detailů (operace nad stavovým prostorem, kritérium ukončení, atd. atd.) proveďte sami, tyto volby pokud možno zdůvodněte a ověřte experimentálním vyhodnocením. Hodnocení Řešení této úlohy je podstatnou součástí hodnocení zkoušky (28 bodů ze 100). Hodnotí se především postup při aplikaci heuristiky, tj. postup a experimenty, jakým jste dospěli k výsledné podobě (parametry, konkrétní operátory apod.). Například, pokud máte v řešení nějaké hodně neortodoxní prvky a pokud máte jejich výhodnost experimentálně doloženou, těžko mohou vzniknout námitky. Méně významné jsou konkrétní dosažené výsledky. Nežádáme rozhodně, aby semestrální práce měla úroveň světové výzvy Centra diskrétní matematiky Rutgersovy univerzity.

Tato práce by měla sloužit jako ověření Vašich schopností používat zvolenou pokročilou iterativní metodu. Ideálním výstupem by měl být algoritmus schopný řešit co nejširší spektrum instancí s rozumnou chybou. To neznamená, že pokud se Vám některé instance "nepovedou", je vše špatně. Důležité je, abychom viděli, že jste se aspoň snažili. Někdy to prostě nejde...

3 Rozbor zadaného problému

Úkolem semestrální práce je vytvořit program řešící problém vážené splnitelnosti boolovské formule v konjunktivní normální formě. Tedy řešit SAT problém s váhami jednotlivých proměných.

Hlavním kritériem pro mě bude splnění takového problému, tedy splnění všech klauzulí. Toto se ne vždy může podařit proto se případně budu snažit zachytit procento splněných klauzulí. Dalším kritériem bude váha dané instance podle ohodnocení proměných, která by měla být co nejvyšší.

Problém budu řešit pomocí genetického algoritmu, který popíši v další kapitole. Aby algoritmus mohl s řešeními pracovat je nutné vytvořit fitness funkci, pomocí které bude možné jednotlivé řešení porovnávat a rozhodovat o tom, které je úspěšnější. Tuto funkci jsem nejprve vytvořil jako součet poměru počtu splněných klauzulí a váhy ku maximální možné váze. K tomuto jsem byl následně nucet přidat koeficient ovlivňující přínos každého z kritérií, protože jak jsem psal výše splnění

problému je primárním cílem. Vytvořená fitness funckce vypadá následovně:

$$fitness(S) = a * \frac{\text{počet splněných klauzulí}}{\text{celkový počet klauzulí}} + (1 - a) * \frac{\text{váha řešení}}{\text{maximání možná váha}}$$

kde S je jedno řešení a a koeficient určující poměr členů, kde vysoká hodnota a blízká k 1 bude upřednostňovat řešení s větším počtem splněných klauzulí.

V rámci řešení budu pracovat i s konfiguracemi, které nejsou řešením, protože prostor řešení je obrovský a jen pár konfigurací splňuje kritérium úspěšnosti řešení a při jeho oříznutí bych mohl dostat nespojitý prostor, kde bych mohl často uváznout v lokálním minimu.

4 Popis implementovaného genetického algoritmu

Implementovaný genetický algoritmus je algoritmus, který se skládá z hlavního generačního cyklu a příslušných method, mezi hlavní patří selekce, mutace, křížení. Algoritmus je možné rozšířit i o mnoho dalších method např. elitismus nebo obměnu populace. Algoritmus je iterativní, kde počet cyklů může určovat předem daný počet nebo může být zastaven nějakou heuristikou. V práci jsem použil pevný počet cyklů, který je možné zadat. V každém cyklu je spočítána hodnota fitness funkce pro každého jedince. Dále jsou pomocí selekce vybíráni jedinci do nové populace. Tito jedinci mohou být kříženi a další vliv na ně má i mutace. Pomocí těchto method je možné udržet divezifikovanou populaci, pomocí které je probledáván stavový prostor. Popis jednotlivých metod:

Elitismus Je výběr nejlepších jedinců, kteří jdou do nové populace. Tento princip může vést k rychlé konvergenci populace.

Selekce Selekci jsem implementoval pomocí turnaje. Selekční tlak je řízen velikostí turnaje, kde malé turnaje davájí šanci vybrat i slabé jednice. Tedy toto je výhodně spíše na začátku, proto jsem v algoritmu vytvořil možsnot nastavení postupného růstu velikosti turnaje s každou generací. Jedinci sou do turnaje vybíráni náhodně s opakování přes celou generaci.

Křížení Dva jedinci vybraní selekcí můžu být s nějakou pravděpodobností kříženi. Z křížení vzniknou nový potomci, kteří budou tvořit základ nové generace. Křížení je možné provádět více způsoby, kde jsem zvolil dvoubodové křížení.

Mutace Je určena pravděpodobností změny každého bitu jedince. Provádí se pro každého jedince v nové generaci.

5 Implementace

Algoritmus jsem implementoval v jazyce Python a pro hlavní cyklus genetického algoritmu jsem využil rožšiření Cython, kde lze psát kód Pythonu typovaný a je překládán do jazyka C pro spojení efektivity psaní v jazyce Python a rychlosti jazyka C.

Řešič příjímá v argumentech configurační soubor ve formátu yaml. V tomto souboru je podle vzoru (přiložen jako config.yml) nutné specifikovat složku s problémy, které bude algoritmus řešit a složku, kam bude ukládán výstup (csv soubor a grafy pro jednotlivé instance) a dále je také parametry samotného genetického algoritmu, které je možné zadávat jako počátek, konec a krok nebo jako jedinou hodnotu. Pro rozsah budou vyzkušeny všechny zadané hodnoty.

Jednotlivé generace ukládám jako 2D pole, takto ukládám i novou generaci, do které jsou přidáni buď vybraní jedinci nebo v případě křížení jejich potomci. Klauzule a váhy jsou také uloženy pomocí polí. Pro každou generaci tedy iteruji tak dlouho dokud nenaplním potřebný počet jedinců, který byli určeny v konfiguraci.

Turnaj je prováděn vybíráním počtu náhodných indexů, kde je z jedninců ponechán pouze nejlepší, který je vrácen. Přes všechny generace je také ponecháno nejlepší řešení, které ovšem nezasahuje do jednotlivých generací. Dále jsou také ukládány jednotlivé hodnoty fitness pro každého

jedince v každé generaci, tak aby bylo možné vykreslit grafy průběhu algortimu. Parametry, které je možné nastavit pro genetický algoritmus jsou nasledující:

generationsize Počet jedinců v jedné generaci.

generationcount Počet iterací algoritmu (počet generací).

mutation Pravděpodobnost mutace jednoho bitu.

crossover Pravděpodobnost křížení jedinců.

selection Procento určijící kolik jedinců z populace bude vybráno.

selection_add Počet jednců o kolik je v každé generaci zvětšena velikost turnaje. Není zadáno v procentech a je možné zadat pouze jednu hodnotu.

elitism Počet nejlepších jedinců, kteří jsou vybraní do nové generace. Není možné zadat intervalem s krokem.

fitness Hodnota koeficientu a popsaného v kapilole 3. Není možné zadat rozsahem hodnot a krokem.

6 Testovací data

Testovací data k problému nebyly poskytnuty globálně, ale bylo nutné si nějaké testovací instance vytvořit nebo sehnat. Já jsem se při řešení problému omezil na instance problému 3-SAT, které lze bez vah jednotlivých problému, jednoduše sehnat na stránkách prot SAT Competition nebo SAT Race. První sadu (v experimentech označovanou jako D1) instancí jsem použil instance ze SATLIB, kde mám k dispozici instance s počtem proměných od 91 do 175. Poměr klauzulí ku počtu proměných je přibližně 4.3, tedy tyto instance jsou nejtěžšími pro 3-SAT. Dále mám k dispozici instance 3-SAT náhodně generované (pomocí generátoru) s počtem proměných 50, ale s rozsahem poměru klauzulí a proměných od 3 do 8 (v experimentech označovanou jako D2). Všechny instance jsou ve formátu DIMACS.

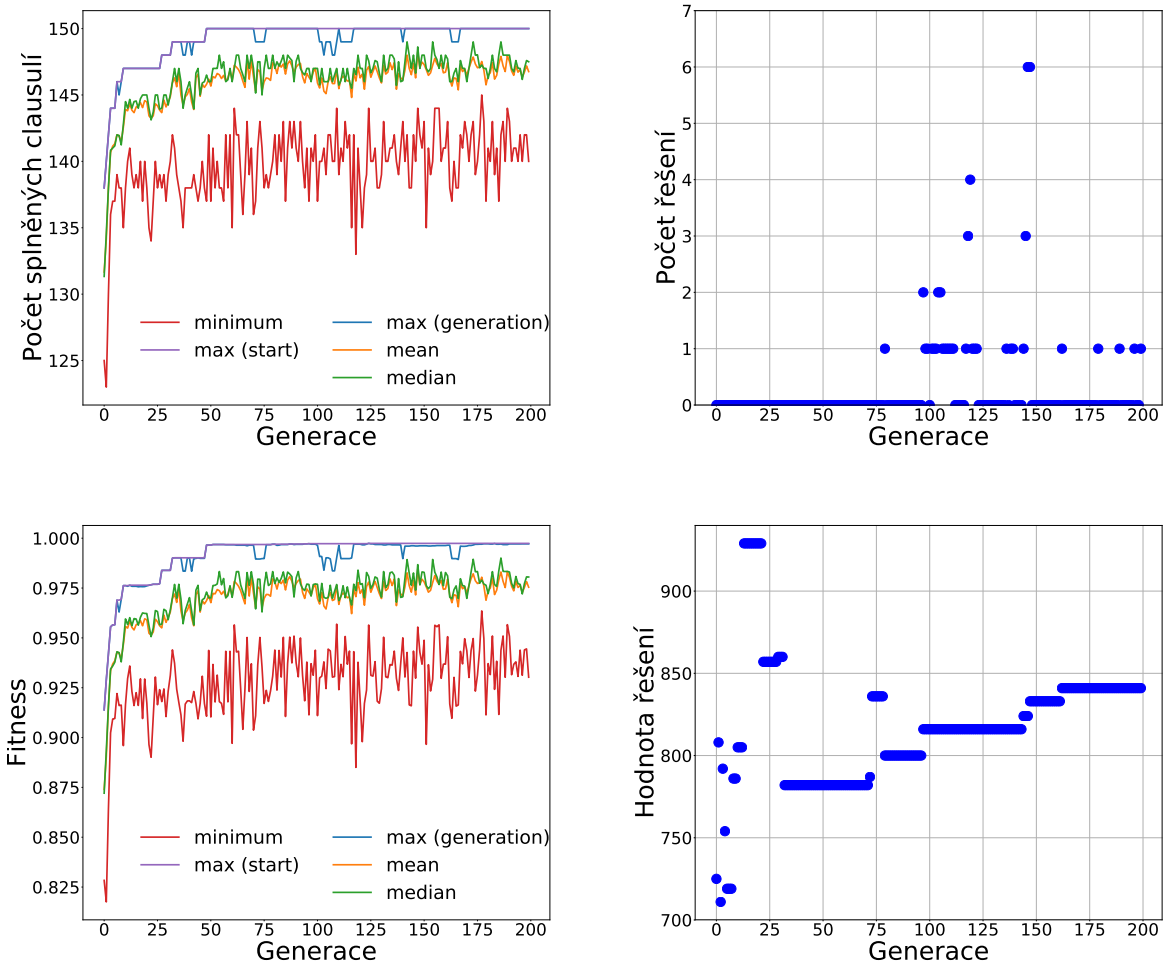
Pro splnění zadání bylo nutné dogenerovat váhy k jednotlivým proměným. Pro zachování formátu a možnosti využití těchto instancí i pro SAT řešiče, jsem váhy uložil jako komentář *"c weights 2 ..."*, aby nebyl porušen formát souboru. Váhy jsem generoval náhodně z rozsahu 1-50 jako celá čísla.

7 Testovací platforma

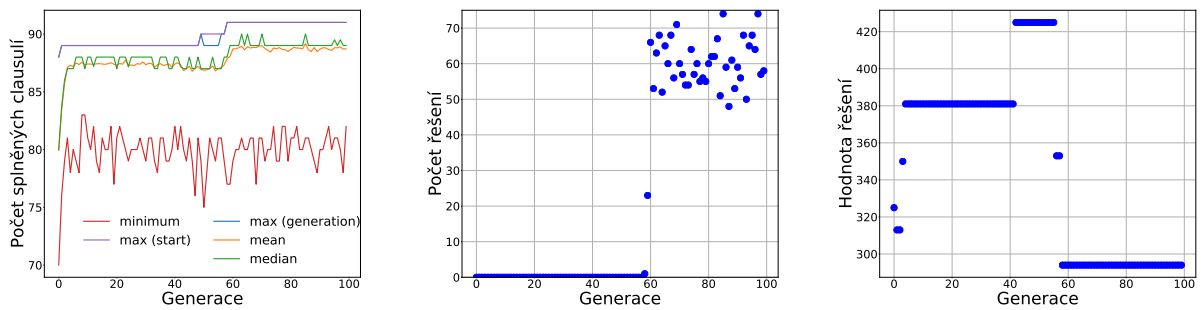
Řešení samotného genetického algoritmu bylo vytvořeno v Cythonu a přeloženo do jazyka C. Experimenty jsem prováděl ve virtuálním prostředí s operačním systémem Debian 9. Procesor Intel Core i5 5200U @ 2.20GHz a testy byly prováděné v režimu jednoho vlákna, při minimálním zatížení počítače dalšími programy.

7.1 Cíle

V genetickém algoritmu lze měnit jeho paramentry, cílem tedy je odzkoušet heuristiku v řešení vážené splnitelnosti SAT problému v závislosti na těchto parametrech. Cílem je prozkoumat nastavení genetického algoritmu tak, aby algoritmus byl schopen řešit co nejvíce problému nebo řešit problémy s co nejmenší chybou.



Obrázek 1: Na obrázcích jsou grafy generované při běhu algoritmu. Vlevo nahoře je graf, kde je v rámci generací zobrazen počet splněných klauzulí. Pod ním je stejný graf pouze s rozdílem, že zde jsou zobrazeny hodnoty fitness funkce. Graf vpravo nahoře ukazuje počet řešení v populaci v rámci generačního cyklu. Na grafu pod ním je vývoj maximální dosažené váhy proměnných v rámci generačního cyklu.



Obrázek 2: Na obrázcích je ukázán ne úplně ideální průběh. Na levém obrázku počet splněných klauzulí, uprostřed počet řešení v populaci a na pravém váha nejlepšího řešení.

8 Experimenty

V experimetech prozkoumám závislost genetického algoritmu na parametrech jako jsou počet generací, velikost populace, hodnota mutace, pravděpodobnost křížení a selekční tlak. Vždy se pokusím testovat jeden parametr a ostatní zafixuji, aby bylo možné měřit vliv tohoto parametru na průběh řešení. Zafixované hodnoty mají hodnotu z ukázky nastavení níže a pro hodnotu mutace je hodnota 0.05.

V rámci běhu algoritmu program vytváří grafy o běhu a jsou zaznámenávány statistiky do csv souboru. Na obrázku 1 jsem uvedl příklad normálního a úspěšného běhu iterativní heuristiky. Je také vidět, že hodnota fitness funkce a počet splněných klauzulí spolu velice korelují, což bylo předpokládáno již při sestavení vzorce pro výpočet fitness funkce. Proto v následujících experimentech budu vždy uvádět jen jeden z těchto dvou grafů, protože jsou téměř totožné. Grafy na obrázku 1 jsou vygenerovány s nastaveními níže, kde mutace je 0.05, tedy první hodnota ze seznamu. Konfigurační souborem ukaázaným níže, lze tedy řídit chování programu a algoritmu a tento soubor je nutné zadat při spuštění.

V experimentech jsem tedy měřil především dosaženou hodnotu fitness funkce a procento splněných klauzulí. Na těchto hodnotách budu také především testovat chování algoitmu na změnu parametrů za cílem nalezení nastavení, které bude při řešení problému nejúspěšnější.

RUN:

out: ./out/t

in: ./instCtV

GA:

generationcount: 200

generationsize: 200

mutation: 0.05 0.21 0.05

crossover: 0.7

selection: 0.1

selection_add: 2

elitism: 1

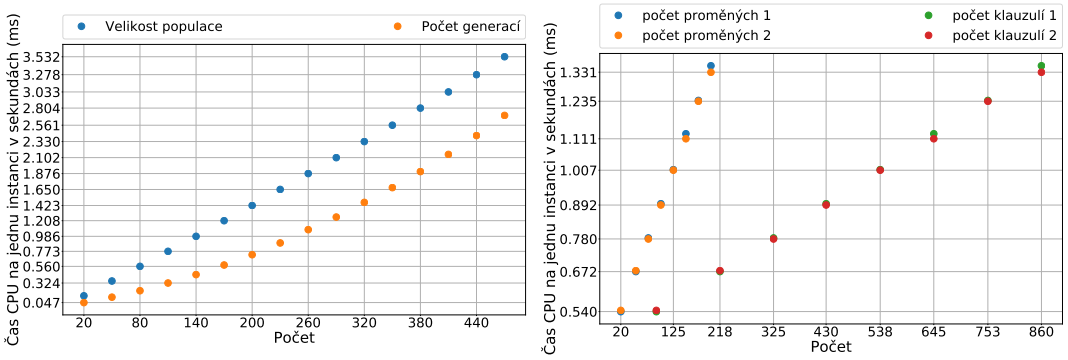
fitness: 0.99

Na obrázku 1 je ještě patrný vysoký koeficient ve fitness funkci, kde je hlavní podmínkou splnění klauzule. Jak je patrné, tak při naležení řešení poklesla hodnota řešení, protože paramentr hodnoty nebyl tak upřednostňvaný a hodnota se opět zlepšovala v dalších generací, kdy už v řešení byly přítomna řešení. Toto je velice markantní na obrázku 2, takto také dopadaly běhy algoritmu. Pří běhu bylo nalezeno řešení, hodnota řešení prudce klesla, ale již se ji nepodařilo vylepšit. Bohužel z důvodu neznalosti optimálního řešení nedokáži posoudit, zdali náhodou toto nebylo optimální řešení, pro splnění problému SAT.

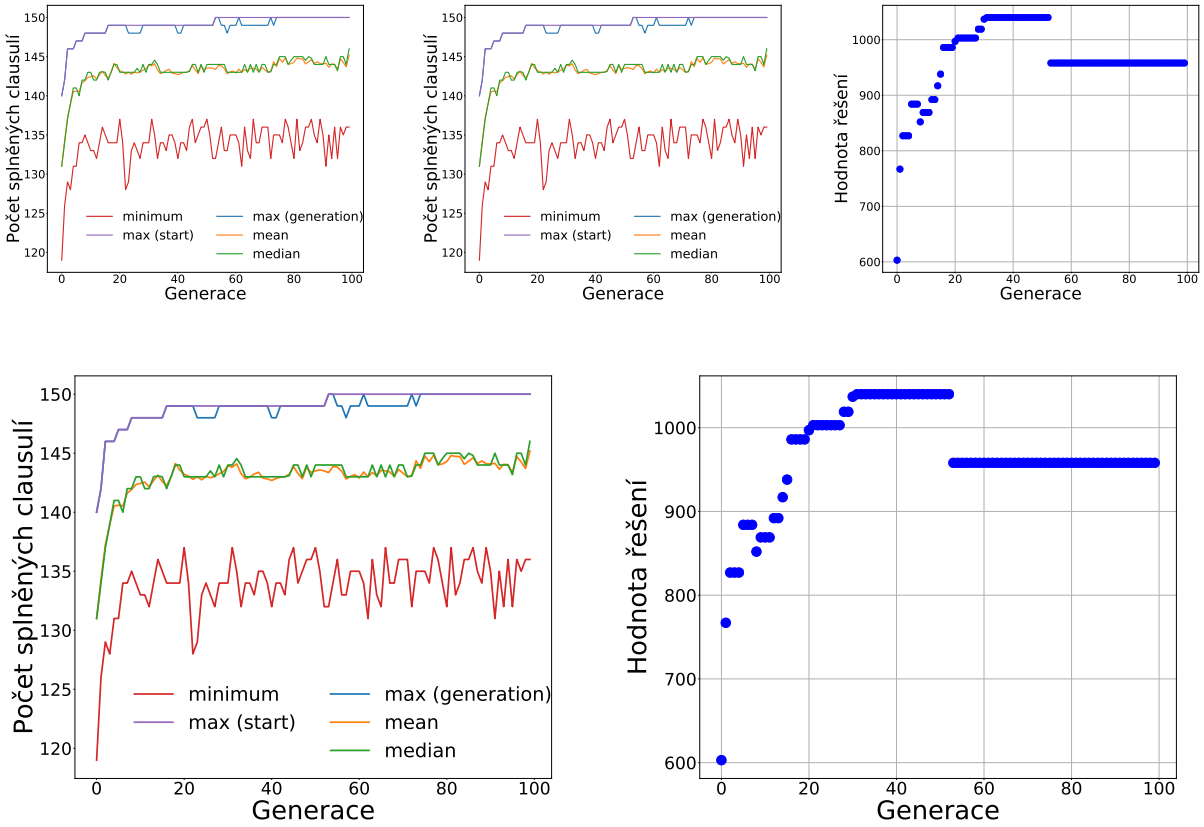
8.1 Závislosti některý parametrů instancí a algoritmu na výpočetním čase

8.2 Závislost na počtu generací

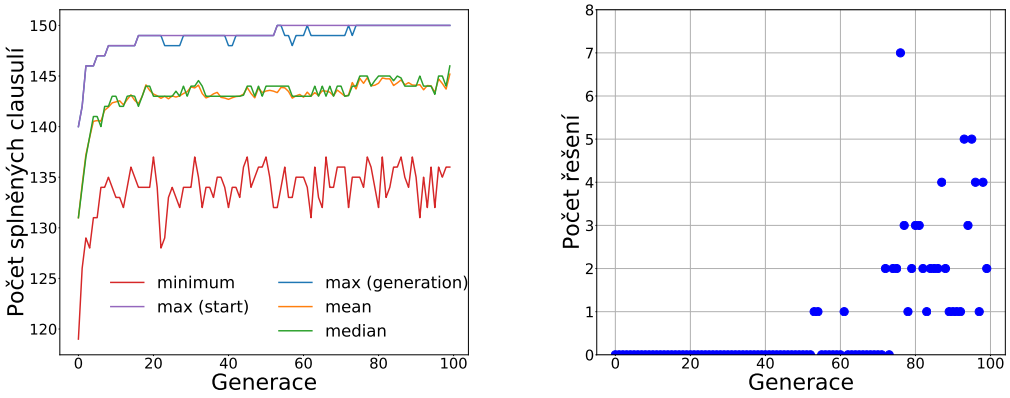
no asi na tom něco bude...



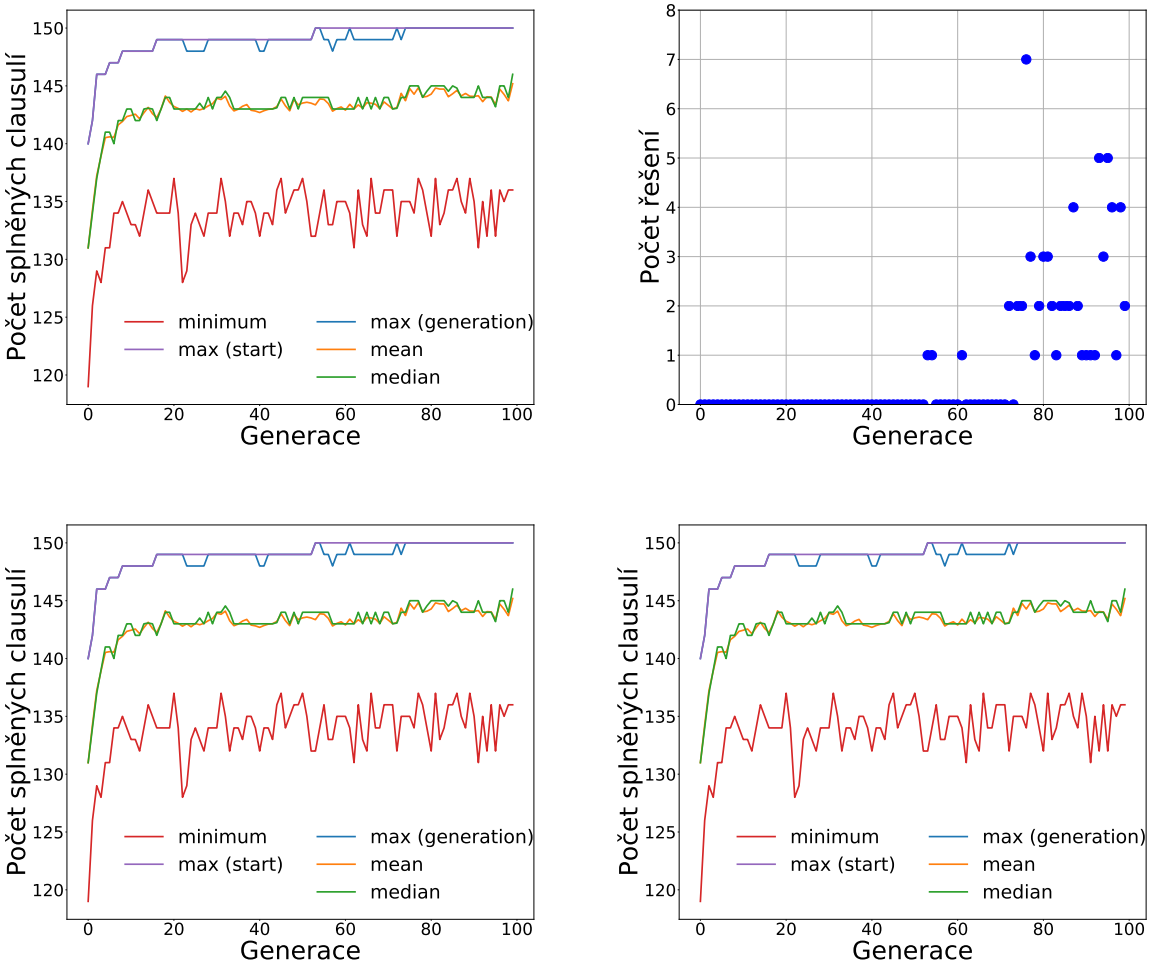
Obrázek 3: Na grech jsou vidět časové náročnosti. Na levém grafu je zobrazena časová náročnost na velikosti populace a počtu generací. Na pravém grafu je zobrazena časová závislos na počtu proměnných a počtu klauzulí v problému.



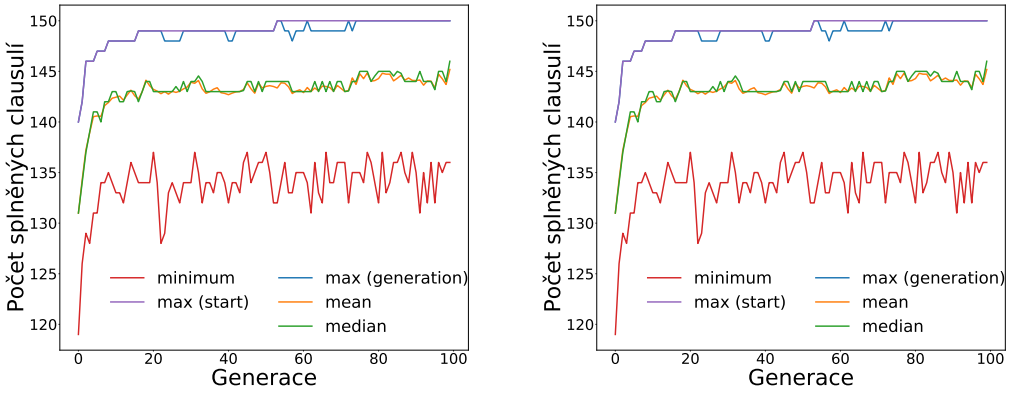
Obrázek 4: Zde jsou uvedené grafy vývoje řešení pro vybrané hodnoty počtu iterací na jedné teplotě. Konkrétně zleva pro hodnoty 30, 60, 120, 240, 300



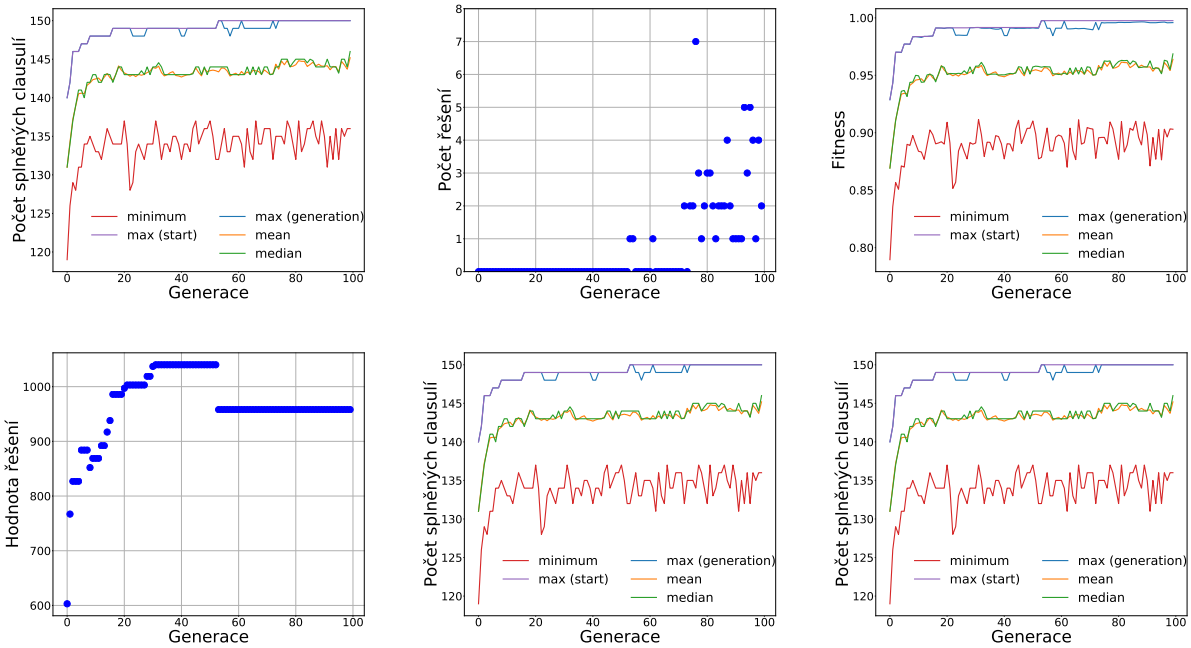
Obrázek 5: Na levém grafu je závislost relativní chyby na koeficientu ochlazování. Na pravém grafu je závislost výpočetního času na koeficientu ochlazování



Obrázek 6: Zde jsou uvedené grafy vývoje řešení pro vybrané hodnoty koeficientu ochlazovaní. Konkrétně zleva pro hodnoty 0.993, 0.995, 0.997, 0.999



Obrázek 7: Na levém grafu je závislost relativní chyby na počtu iterací na jedné teplotě. Na pravém grafu je závislost výpočetního času na počtu iterací na jedné teplotě



Obrázek 8: Zde jsou uvedené grafy vývoje řešení pro vybrané hodnoty počtu iterací na jedné teplotě. Konkrétně zleva pro hodnoty 30, 60, 120, 180, 240, 300

9 Závěr

Během experimentu jsem prozkoumal pokročilou iterativní heuristiku - simulované ochlazování. Ověřil a prozkoumal jsem závislosti této heuristiky na řídících parametrech. Parametry jsou určitě závislé na daných problémech či paramentech instancí, což je patrné i ze vzorců, kde přímo vystupuje rozdíl cen řešení. Tyto rozdíly se mohou pohybovat v různých intervalech a tomu je potřeba parametry také upravit, tedy hýbat s počáteční teplotou, která ve vzorci vystupuje jako druhý parametr.

Simulované ochlazování je randomizovaná heuristika sloužící k prochazení prostoru, a proto může při špatném nastavení mít tendenci uváznutí v lokálních extrémech. Heuristika kombinuje přístup diverzifikace na záčatku, a následující intenzifikace je snaha o nalezení optimálního řešení.

Ze závislostí zjištěných během experimentu je vidět, že počáteční teplota je důležitý parametr v závislosti na hodnotách ceny řešení. Pokud se ceny řešení pohybují ve velkých hodnotách bude mou snahou nastavit vyšší teploty, než pokud se budou pohybovat na nějakém intervalu a budou například normalizované.

Parametr ochlazování je důležitý k dostatečému na vzorkování rozsahu teploty a tedy i dostatečném počtu kroků v jednotlivých fázích a to především ve fázi intenzifikace.

Jednotlivé počty kroků na dané teplotě nám naopak dovolí prozkoumat dané okolí, ale tento parametr nemá lineární závislost na chybě, ale je dán nepřímou uměrou a volíme tak mezi relativní chybou a časovou náročností.

Časová náročnost algortitmu je daná nastavenými parametry a s konkrétními parametry provede algoritmus vždy stejný počet kroků, tedy pro mou implementaci tohoto přístupu. Simulované ochlazování je možné například implementovat s proměným počtem kroků na jedné teplotě, který může být opět řízen nějakou jednoduchou heuristikou.